The Stair Sketch: Bringing more Clarity to Memorize Recent Events

Yikai Zhao*, Yubo Zhang*, Pu Yi*, Tong Yang*, Bin Cui*, Uhlig Steve†
*Department of Computer Science, Peking University, China
†School of Electronic Engineering and Computer Science, Queen Mary University of London, United Kingdom

Abstract—**Data stream processing has become fundamental in computer science, with a wide range of applications, such as in databases, data mining, and security. Memorizing when an item appears in the data stream is one important task in stream processing. Because the older data is, the less value it has, memorizing recent events with higher accuracy is desirable. To achieve this, we propose a novel data stream processing structure named the Stair sketch. Our key idea is to organize the memory used by different time periods in the shape of stairs. We deploy the Stair sketch on Bloom filters, CM sketches, and CU sketches as case studies. Experiment results show that our approach outperforms state-of-the-art algorithms by more than $5\times$ in accuracy while providing comparable efficiency. The source code of the Stair sketch is available at GitHub.**

I. Introduction

A. Background and Motivation

Data stream processing consists in collecting, analyzing, or answering queries about the data at any prior time and the current time. Specifically, given a data stream and a time period $[T_1, T_2]$, one is usually interested in whether an item occurs, or how many times it does. However, not all occurrences are equally important: one often cares more about recent occurrences than older ones. For example, the Youtube recommendation system [1] ranks the popularity of videos according to the number of hits, especially recent hits. As another example, when a network firewall [2] detects that a server is under attack, the users who have visited the server recently are more suspicious. Due to the high volume of data in today's streams, people tend to use sublinear data structures, namely *sketches*, because of their high efficiency in terms of both time and space. Typical sketches include the Bloom filter [3], the CM sketch [4], and others [5]–[8].

Given the higher importance of recent occurrences and the efficiency of sketches, this paper aims to devise a new sketch that can memorize recent events with higher accuracy. We call this property of our scheme ***time gradualness***. There are two aspects to this *time gradualness*.

- ***Error gradualness***. Our scheme needs to be more accurate when performing estimation over more recent periods. This makes better use of the memory available.
- ***Time stability***. The estimation error for the recent past needs to be stable within a well-defined bound. This supports the long-term deployment and use of the scheme.

These two concepts are formally defined in Section III.

B. Prior Art and Their Limitations

Hokusai [9] is the foundational work that achieved *time gradualness* for data stream processing. It builds a fixed size sketch for each new time period, such as a CM sketch or a Bloom filter, and compresses some old ones to control the memory usage. It is simple and easy to deploy in practice. However, Hokusai also comes with several shortcomings. First, the load of each sketch is imbalanced. In practical data streams, the number of items in each period may vary, and so does the accuracy of sketches built during each period. As a result, the accuracy will sometimes be very low and sometimes very high. Second, to allow compression, the sketches used in Hokusai cannot rely on the optimal parameters to achieve the highest accuracy. For instance, as pointed by [10], to build a compressed Bloom filter [3], only 2 hash functions should be used, much less than the optimal number of hash functions. Third, compressing a sketch requires traversing the whole sketch, which is time consuming. To address these shortcomings, the successors of Hokusai have fallen into two categories. The first category keeps *error gradualness* but discards *time stability*. The typical solution is the Ada-sketch [11]. Its key idea is to set higher weights for more recent items. The second category keeps *time stability* but discards *error gradualness*. Typical solutions are persistent data sketching [12] and persistent Bloom filter [13]. They share sketches for different time periods to achieve time stability. However, they are memory consuming, because they memorize all events with the same accuracy. None of the above solutions can achieve both aspects of *time gradualness*. Our design goal is to achieve the *time gradualness* while significantly improving accuracy, by focusing on higher accuracy for more recent items.

C. Our Solution

To achieve the design goal, we propose a novel data stream processing structure called the Stair sketch. The Stair sketch can meet both requirements of *time gradualness*, and is much more accurate than the state-of-the-art solutions in terms of *time gradualness*. Our Stair sketch is generic, in that it can be applied to many sketches. We choose the Bloom filter [3], the CM sketch [4], and the CU sketch [5] as case studies.

Our data structure consists of many atomic sketches, each of which can be a Bloom filter, a CM sketch, or others [5]–[8]. We build several atomic sketches for each time period. Given a time period, the more atomic sketches it uses, the higher accuracy it achieves. The key idea of the Stair sketch is to organize the memory in a stair shape: the number of atomic sketches used for each period increases just like the height of each stair. We use an example in Figure 1 to show the

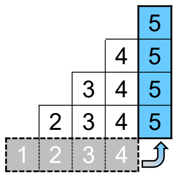


Fig. 1: A Stair sketch with 10 atomic sketches.

basic version of the Stair sketch. Time periods 1, 2, 3, 4 have passed. The numbers of atomic sketches used for them are 1, 2, 3, and 4, respectively. Therefore, the accuracy increases gradually. Now time period 5 arrives. We build four atomic sketches for it, and delete one atomic sketch from each time period 1, 2, 3, 4. The deleted sketches will be persistent but on a slow memory or disk. In this way, we organize our data structure in a stair shape, and achieve a fixed memory usage.

Similarly to Hokusai, our basic version is imbalanced. From a design point of view, sharing is an effective strategy to reduce load imbalance. Therefore, we propose **horizontal sharing**, which lets atomic sketches at the same horizontal line share memory. Horizontal sharing is effective because each sharing spans across multiple time periods. We also show that the Stair sketch is a generic structure. By using different kinds of atomic sketches, such as Bloom filters, CM sketches, or CU sketches, the Stair sketch can handle different data stream tasks.

Contributions: Our contributions are the following: 1) We propose the concept of time gradual data stream processing, which has two aspects: *time stability* and *error gradualness*. 2) We propose a novel scheme named the Stair sketch, which is a general data stream processing structure achieving *time gradualness*. 3) We apply our Stair sketch to Bloom filters, CM sketches, and CU sketches as case studies. 4) We conduct extensive experiments, the results show that the Stair sketch is time gradual, and outperforms state-of-the-art algorithms by more than $5\times$ in accuracy while providing comparable efficiency. The source code of the Stair sketch is available at GitHub [14].

II. Related Work

A. Typical Sketches for Data Streams

In recent years, sketches have been extensively used because of their reasonable trade-off among accuracy, time efficiency, and memory usage [15]. Typical sketch algorithms, including the Bloom filter [3], the CM sketch [4], the CU sketch [5], and others [6]–[8], [16]–[23], form the basis of various applications in the fields of databases [24]–[30], data mining [31]–[37], and information security [38]–[43]. Among those sketches, many share the same hash-based scheme. They use very similar data structures, as shown in Figure 2. The data structures use d hash functions and a cell array. Upon each item's arrival, it is mapped by d pair-wise independent hash functions to d cells, which are then updated. The algorithms then utilize the information stored in the cells to offer an estimation of the item.

For a Bloom filter (BF), each cell is essentially a bit, which is set to 0 at the beginning. For each incoming item, its d mapped bits are set to 1. For a membership query of an item, *i.e.*, querying whether an item occurs in the data stream, the

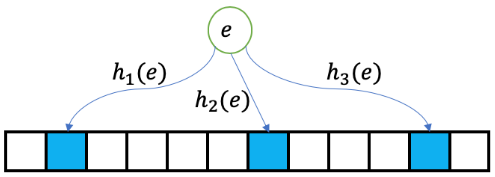


Fig. 2: A common hash-based probabilistic data structure ($d = 3$ in this figure).

BF checks its d mapped bits to see if all of them have been set to 1. Clearly, there is a false positive issue for Bloom filters due to hash collisions, but the False Positive Rate (FPR) is fairly small. Indeed, it is pointed out that with a memory of fewer than 10 bits per item, the BF can achieve a 1% FPR [44].

For the CM and CU sketch, each cell is a counter. For the CM sketch, each counter is initially 0, and each time the counter is mapped, it is incremented by 1. The frequency of an item is estimated by the minimum value among its d mapped counters. The CU sketch closely resembles the CM sketch. The only difference is that when inserting an item, the CU sketch only increments the mapped counters that have the minimum value in the d counters. The CM and CU sketch both have one-sided errors. For comparison, the CU sketch has a lower error rate.

Besides the above three typical algorithms, many other sketch algorithms share the same hash-based scheme in Figure 2, such as the Count sketch [7], the CSM sketch [8], the Bitmap [45], and [6], [46], [47].

B. Time Sensitive Sketch Schemes

The algorithms mentioned earlier have gained wide acceptance due to their simple implementation and great efficiency. One recent research direction has been to enhance them to support time-sensitive queries. The motivation for this is that in various scenarios, we may want to ask questions about data at a prior time instance, which is not supported by the algorithms mentioned above. For instance, a Bloom filter can answer questions like *"Has this item arrived before?"*. However, in many scenarios, one might also ask time-sensitive queries like *"Has this item arrived in the last five minutes?"* or *"Has this item come between 9:30 and 10:00?"* that a Bloom filter cannot answer. Besides membership queries, this issue applies to frequency estimation tasks as well. To support time-sensitive queries as well as make an efficient usage of space, many algorithms have been proposed based on the above typical sketch algorithms.

Hokusai [9] tries to allocate a CM sketch for each time period, and the CM sketches for more recent time periods consume more space. To query the frequency of an item during a certain time period, Hokusai simply queries the corresponding CM sketches. The key idea is to make an efficient trade-off between accuracy and space. By splitting the history into dyadic time periods and compressing the sketches recording older information by half on a regular basis, it saves memory for recording newer information. In this way, Hokusai gets a more accurate frequency estimation for more recent history.

Inspired by the pre-emphasis and de-emphasis technique used in Dolby noise reduction, Ada-CMS [11] gives a time-adaptive estimation of frequency (*i.e.*, more accurate estimation for recent statistics). The Ada-CMS has a similar data structure to the CM sketch. When inserting an item e that arrived in time period t, Ada-CMS increases all the mapped counters of the tuple (e, t) by $f(t)$, which is a monotonically increasing function. To query the frequency of e during period t, Ada-CMS gets the minimum value of all mapped counters of the tuple (e, t) and divides them by $f(t)$, taking the final result as the estimated value. Due to the monotonicity of function f, the effect of older counters on recent ones is reduced.

Persistent data sketching [12] improves the CM sketch into a persistent data structure, so that it can support queries for history frequency. It uses a piecewise-linear function to approximate the historical value of each counter. For membership queries, the persistent Bloom filter [13] (PBF) extends the Bloom filter. There are two versions of the PBF in its paper, the PBF-1 and PBF-2. The PBF-1 splits the entire time interval into smaller periods with a size that is a power of two, which is similar to the interval splitting of the segment tree. For each period, it builds a Bloom filter. The PBF-2 combines some small Bloom filters in the PBF-1 to save space, which improves its overall performance but makes it time unstable.

III. Problem Statement

We introduce the data stream model used in this paper, and formally define *time gradualness*. A data stream $\mathcal{S}$ is defined as follows,

$$\mathcal{S} = \{\langle e_1, t_1\rangle, \langle e_2, t_2\rangle, \cdots, \langle e_n, t_n\rangle, \cdots\} \tag{1}$$

where e_i is an item belonging to the set $\mathcal{U} = \{1, 2, \cdots, N\}$, and $t_i \in \mathbb{Z}^+$ is a monotonically increasing timestamp indicating in which period item e_i occurs. Let $\mathcal{N}_k = \sum_{i=1} 1_{\{t_i=k\}}$ be the number of items in the k-th period. We assume that there is an upper bound for these n_k, *i.e.*, $\mathcal{N}_k \leqslant \mathcal{N}$. It is worth noting that although the number of items in each period does not exceed $\mathcal{N}$, it varies dramatically in different periods.

Given a data stream $\mathcal{S}$ and an estimation solution, the estimation error $\mathcal{E}(k, k')$ refers to the average error about the k'-th period estimated by the scheme, and it is currently in the k-th period. For example, for membership queries, $\mathcal{E}(5, 2)$ indicates the average error of estimating whether item e has occurred in the second period and we are in the fifth period. Given the memory budget $\mathcal{M}$, we use the estimation error $\mathcal{E}$ to formally define error gradualness and time stability as follows.

***Error Gradualness*:** A property indicating that a scheme has a lower estimation error for the more recent period, *i.e.*,

$$\mathcal{E}(k, k') \leqslant \mathcal{E}(k, k'') \qquad \forall\ k'' \leqslant k' \leqslant k \tag{2}$$

***Time Stability*:** A property indicating that the estimation error of a scheme for the most recent period (*i.e.*, current period) is stable within a well-defined nontrivial bound that only relates to $\mathcal{N}$ and $\mathcal{M}$, *i.e.*,

$$\mathcal{E}(k, k) \leqslant f(\mathcal{N}, \mathcal{M}) \qquad \forall\ k \tag{3}$$

A scheme has time gradualness *if and only if* it satisfies both error gradualness and time stability.

IV. The Stair Sketch

The Stair sketch is a generic structure and can be applied to many existing sketches such as Bloom filters [3], CM sketches [4], CU sketches [5], and more [6]–[8]. To illustrate the rationale of our Stair sketch, we show its application with Bloom filters as a case study. Then we will show how to apply the Stair sketch data stream processing structure to many other existing typical sketches.

A. Stair Bloom Filter: Basic Version

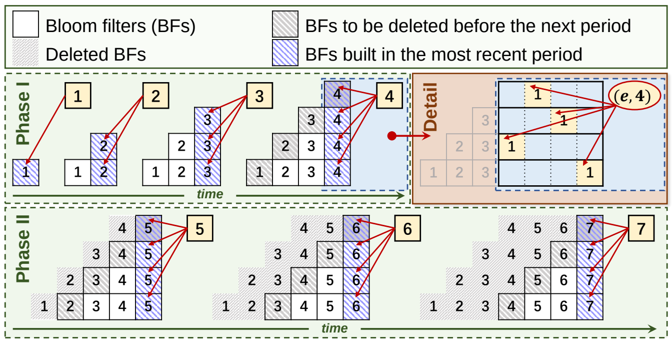

Bloom filters (BFs)
Deleted BFs
BFs to be deleted before the next period
BFs built in the most recent period
Phase I
time
Detail
(e,4)
Phase II
time

Fig. 3: The basic version of the Stair sketch ($m = 4$). Each square is a sketch (a Bloom filter) with a number in it indicating when it was built. This figure shows the structure of the Stair sketch after the first 7 time periods.

Suppose that we want to keep the data stream information of the latest m periods in memory. The Stair Bloom filter consists of $\frac{m(m+1)}{2}$ Bloom filters of the same memory size w. Therefore, the total memory consumption is $\frac{m(m+1)}{2}w$. Given a fixed memory size, we can adjust the parameters w and m to meet the limits. As shown in Figure 3, the Stair Bloom filter is dynamically constructed and updated.

Construction Phase I: In the very first few periods when the space consumption of the Stair Bloom filter has not reached the memory limit, we fill the memories with new Bloom filters (line 5-8 in Algorithm 1 and phase I in Figure 3). For the k-th period where $k \leq m$, we create k Bloom filters. Each Bloom filter uses d pairwise-independent hash functions.

Insertion: For each incoming item e in the k-th period, we perform a standard Bloom filter insertion of e in all the k new Bloom filters (line 13-17 in 1 and Figure 3). Let $\mathcal{A}_{k(i)}$ be the i-th Bloom filter of the k-th period, and $\mathcal{A}_{k(i)}.h_j$ be the j-th hash function of $\mathcal{A}_{k(i)}$. We set the bit $\mathcal{A}_{k(i)}.h_j(e)$ in $\mathcal{A}_{k(i)}$ to 1, for all $i \in \{1, 2 \cdots k\}$ and $j \in \{1, 2 \cdots d\}$.

Construction Phase II: When the Stair Bloom filter has reached its memory limit, we enter the second phase (line 9-12 in Algorithm 1 and phase II in Figure 3). In this phase, when the k-th period begins ($k > m$), we delete one Bloom filter for each i-th period ($i \in \{k-m, k-m+1, \cdots, k-1\}$), and use the released memory block obtained to construct m new Bloom filters for the k-th period.

Insertion: To insert an incoming item e in the k-th period, we perform a standard Bloom filter insertion for each of the m new Bloom filters.

Algorithm 1: Construction of the Stair, basic version

```
foreach time period $k$ arrives do
    Initialization($k$);
    foreach $\langle e, k\rangle \in \mathcal{S}$ do
        Insertion($\langle e, k\rangle$);
Function Initialization($k$):
    if $k \leqslant m$ then
        for $i = 1 \to k$ do
            $\mathcal{A}_{k(i)} \leftarrow$ new Bloom Filter($w, d$);
    else
        for $i = 1 \to m$ do
            delete $\mathcal{A}_{\{k-(m+1)+i\}(i)}$;
            $\mathcal{A}_{k(i)} \leftarrow$ new Bloom Filter($w, d$);
Function Insertion($\langle e, k\rangle$):
    $m' \leftarrow \min(k, m)$;
    for $i = 1 \to m'$ do
        for $j = 1 \to d$ do
            $\mathcal{A}_{k(i)}\left[\mathcal{A}_{k(i)}.h_j(e)\right] = 1$;
```

Query (Algorithm 2): To query the occurrence of one item e in the k'-th period, we perform a query for each Bloom filter associated with the k'-th period. Specifically, if it is currently in the k-th period ($k \geq k'$), we can derive the number m'' of Bloom filters recording the period k' (line 2-7 in Algorithm 2). For each Bloom filter $\mathcal{A}_{k'(i)}, i \in \{1, 2 \cdots m''\}$, we check the mapped bits of item e, namely $\mathcal{A}_{k'(i)}.h_j(e)$ for $j \in \{1, 2, \cdots d\}$. If all these $m''d$ bits have been set to 1, then we report that item e occurred in the k'-th period.

Algorithm 2: Query of the Stair, basic version

```
1  Function Query($k, \langle e, k'\rangle$):
2      if $k' \leqslant k - m$ then
3          return Unknown;
4      else if $k \leqslant m$ then
5          $m'' \leftarrow k'$;
6      else
7          $m'' \leftarrow k' + m - k$;
8      occurrence $\leftarrow 1$;
9      for $i = 1 \to m''$ do
10         for $j = 1 \to d$ do
11             occurrence &= $\mathcal{A}_{k'(i)}\left[\mathcal{A}_{k'(i)}.h_j(e)\right]$;
12     return occurrence;
```

Analysis: In the Stair Bloom filter, false positives are inevitable due to the use of Bloom filters. Assuming that the number of incoming items in a period is below a constant upper bound, the false positive rate of each Bloom filter of the same size is expected to have a constant upper bound $\mathcal{P}$. Since we essentially query m'' Bloom filters and take the bit-wise AND of their results, the false positive rate of the Stair Bloom filter is bounded by $\mathcal{P}^{m''}$. As m'' is larger for larger k', the information of more recent periods is more accurate, which suggests the Stair Bloom filter satisfies ***error gradualness*** (defined in Section III.). Similarly, if we query the information of the latest period k, as we have m Bloom filters for period k, the false positive rate is bounded by $\mathcal{P}^m$, which is fairly small. Therefore, the Stair Bloom filter also satisfies ***time stability***.

B. Stair Bloom Filter: Optimized Version

The basic version of the Stair Bloom filter already satisfies the two aspects of time gradualness: error gradualness and time stability, but it still has two limitations. We propose two optimization techniques to address them.

Vertical Sampling: For a fixed fast memory size $\mathcal{M}$, the parameters m and w satisfy $\frac{m(m+1)}{2} \cdot w \leqslant \mathcal{M}$. If we want to record a longer history (*i.e.*, increase the parameter m), the number of the needed Bloom filters increases quadratically, so we have to significantly reduce w, leading to a severe loss of accuracy. If the number of the needed Bloom filters only increases linearly with the increase of m, we may more easily keep a long and relatively accurate history.

Therefore, we propose the *vertical sampling* technique. In Figure 3, we call a set of Bloom filters in a horizontal line a layer. In the example of this figure, from top to bottom, four layers have 1, 2, 3 and 4 Bloom filters, respectively. We find that in order to keep the relatively accurate information as well as meet the requirements of *time gradualness*, it suffices to only keep those layers whose ID is a power of two. For example, if $m = 8$, instead of keeping all 8 layers, we can sample only the layers numbered 1, 2, 4 and 8 (Figure 4). In this way, if we keep the size of each one Bloom filter unchanged, the total space consumption of the Stair Bloom filter only grows linearly with m, so that we can record a longer history with accuracy.

Horizontal Sharing: The basic version of Stair Bloom filter suffers from *load imbalance* easily. For one specific Bloom filter, we call the number of items inserted into it as its *load*. Ideally, the load of Bloom filters is balanced in the Stair Bloom filter, (*i.e.*, all Bloom filters have the same load), so that the memory for each Bloom filter is made full use of. However, as suggested in [11], the data stream is "bursty" in most cases, leading to the severe problem of *load imbalance*. For example, there might be very few items in the k-th period, but in the basic version we still create m Bloom filters for this period (in phase II). Consequently, these $m \cdot w$ memories record very little information and cause waste.

Though the data stream may be "bursty", according to the *Law of Large Numbers*, the average number of items in a number of periods has a much less variation and is more stable than the number of items in one period. This inspires us to use the *sharing* technique. Instead of recording only one period in a fixed-size Bloom filter, we can combine several Bloom filters to a larger compound Bloom filter and record many periods in it. In this compound Bloom filter, we use the **shifting** technique proposed by [48] to distinguish the item e in different periods (line 13-16 in Algorithm 3 and Figure 5). Simply put, **shifting** is to add an offset of k (the ID of the period) to the original hash value of item e for both insertion and query operation. For example, to insert an item e in period k to a Bloom filter A with hash function h and size w, instead of setting the bit $h(e)$ to 1, we set the bit $(h(e) + k)\%w$ to 1. When querying the membership of an item e in period k', we also add an offset of k', so that we can distinguish the item occurrences in different periods.

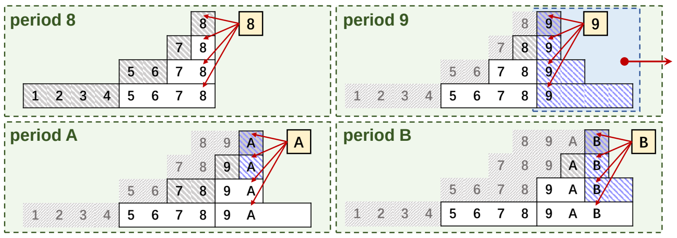
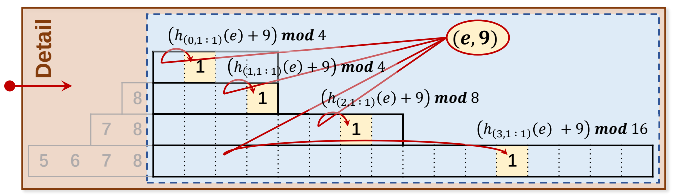


Fig. 4: A Stair sketch ($m = 4$) using the optimizations of vertical sampling and horizontal sharing. Each square is a sketch (a shifting Bloom filter), and each rectangle is a compound sketch (a combination of several shifting Bloom filters). A compound sketch contains several numbers, which indicate the time periods it records.


Detail
$(h_{(0,1:1)}(e)+9)$ mod 4
$(h_{(1,1:1)}(e)+9)$ mod 4
$(h_{(2,1:1)}(e)+9)$ mod 8
$(h_{(3,1:1)}(e)+9)$ mod 16
$(e,9)$
Fig. 5: The detailed process of inserting an item into a shifting Bloom filter. The positions of the mapped bits are the hashes plus the time period number 9 as offsets.

Data Structure (Figure 4): The optimized version of the Stair Bloom filter is summarized in Figure 4 and Algorithm 3-4. We have m layers. In the first layer (from top to bottom in Figure 4), we have one Bloom filter of size w_1 named $\mathcal{A}_{(1,1)}$ which have d_1 hash functions. At the i-th layer, we have two Bloom filters of size w_i namely $\mathcal{A}_{(i,0)}$ and $\mathcal{A}_{(i,1)}$, both of which have d_i hash functions and record $2^{(i-2)}$ periods. In the pseudo code, $\mathcal{A}_{(i,j)}$.begin and $\mathcal{A}_{(i,j)}$.end represent that the Bloom filter $\mathcal{A}_{(i,j)}$ records the data stream from period $\mathcal{A}_{(i,j)}$.begin to $\mathcal{A}_{(i,j)}$.end.

Algorithm 3: Construction of the Stair, optimized

```
1  foreach time period $k$ arrives do
2      Initialization($k$);
3      foreach $\langle e,k\rangle \in \mathcal{S}$ do
4          Insertion($\langle e,k\rangle$);
5  Function Initialization($k$):
6      clear $\mathcal{A}_{(1,1)}$;
7      for $i = 2 \to m$ do
8          if $k > \mathcal{A}_{(i,1)}.end$ then
9              delete $\mathcal{A}_{(i,0)}$;
10             $\mathcal{A}_{(i,0)} \leftarrow \mathcal{A}_{(i,1)}$;
11             clear $\mathcal{A}_{(i,1)}$;
12             $\mathcal{A}_{(i,1)}$.begin $\leftarrow k$;
13             $\mathcal{A}_{(i,1)}$.end $\leftarrow k + 2^{(i-2)}$-1;
14 Function Insertion($\langle e,k\rangle$):
15     for $i = 1 \to m$ do
16         for $j = 1 \to d_i$ do
17             $\mathcal{A}_{(i,1)}\left[\left(\mathcal{A}_{(i,1)}.h_j(e) + k\right) \bmod w_i\right] = 1$;
```

Insertion: To insert an item e in the k-th period, we insert tuple $\langle e,k\rangle$ to Bloom filter $\mathcal{A}_{(i,1)}$ in each layer (the right one in Figure 5) with shifting technique. When a new period begins, if for some i, $\mathcal{A}_{(i,1)}$ is full (*i.e.*, it has recorded 2^{i-1} periods), we then reset its left Bloom filter $\mathcal{A}_{(i,0)}$ and swap their positions (line 5-12 in Algorithm 3).

Query: To query the occurrence of an item e in period k', we can derive an interval $[m', m]$, so that $\forall i \in [m', m], k' \in [\mathcal{A}_{(i,p_i)}.\text{begin}, \mathcal{A}_{(i,p_i)}.\text{end}]$. We query these $(m - m' + 1)$ relevant Bloom filters and take the bit-wise *AND* of their results (see Algorithm 4). Since we still have more Bloom filters for more recent periods, and a constant number of Bloom filters for the latest period, *time gradualness* still holds for the Stair Bloom filter.

Algorithm 4: Query of the Stair, optimized

```
1  Function Query($k, \langle e,k'\rangle$):
2      if $k' < \mathcal{A}_{(m,0)}.begin$ then
3          return Unknown;
4      occurrence $\leftarrow 1$;
5      if $k'$=$k$ then
6          for $j = 1 \to d_i$ do
7              occurrence & =
                   $\mathcal{A}_{(1,1)}\left[\left(\mathcal{A}_{(1,1)}.h_j(e) + k'\right) \bmod w_1\right]$;
8      for $i = 2 \to m$ do
9          if $k' \geqslant \mathcal{A}_{(i,0)}.begin$ then
10             if $k' \geqslant \mathcal{A}_{(i,1)}.begin$ then
11                 $p_i \leftarrow 1$;
12             else
13                 $p_i \leftarrow 0$;
14             for $j = 1 \to d_i$ do
15                 occurrence & =
                       $\mathcal{A}_{(i,p_i)}\left[\left(\mathcal{A}_{(i,p_i)}.h_j(e) + k'\right) \bmod w_i\right]$;
16     return occurrence;
```

The remaining issue is how to configure the parameters of the data structure. In general, larger w_i and d_i ($2 \leqslant i \leqslant m$) can improve query accuracy about the last time period, while larger w_m and d_m can improve query accuracy about earlier time periods, and we advise that users configure reasonable parameters in practice. We also give a recommended parameter configuration method in the mathematical analysis section.

C. The Stair Sketch Usage in Other Typical Sketches

The Stair Sketch is a generic structure, and can be applied to many other typical sketches besides the Bloom filter. To illustrate that, we first introduce the abstract concept of *atomic sketch* which depicts the key properties of various existing sketches such as Bloom filters [3], CM sketches [4], CU sketches [5], and [6]–[8]. Then we show in more detail how the Stair Sketch applies to CM and CU sketches.

An **atomic sketch** has an array of w cells, and d independent hash functions $h_i(\cdot)$. Each hash function maps an item into one of w cells uniformly, and the atomic sketch maps an item to d cells through these hash functions. Atomic sketches have two operators. The updater$(\cdot)$ takes an item, maps it to d cells with hash functions, and updates each of these cells. The aggregator$(\cdot)$ summarizes the statistic of an item according to its mapped cells. The updater is used to insert an item, and the aggregator is used for queries.

Many existing sketches are essentially atomic sketches with concrete updater and aggregator. For example, cells in the

Bloom filters are bits. For an incoming item, the updater of the Bloom filter is to set those mapped bits as 1, and the aggregator of it is the bit-wise AND result of all mapped bits. We have seen how the Stair Bloom filter works, now by just changing the insertion and query process, the Stair Sketch can apply to other sketches featuring the atomic sketch. We show the Stair CM sketch and the Stair CU sketch for example.

The Stair CM Sketch : For CM sketches as atomic sketches, the cells are essentially counters. For an incoming item, the updater increments all its mapped counters by 1, and the aggregator gets the minimum value of its mapped counters. The Stair CM sketch has a set of CM sketches, and manipulates them with the same approach as the Stair Bloom filter manipulates Bloom filters. To insert an item e in the period k, the Stair CM sketch performs a CM sketch insertion with shifting for each CM sketch that records the k-th period. To query the frequency of an item e in the period k', the Stair CM sketch query each CM sketch that records the k'-th period with the aggregator, and take the minimum result of them.

The Stair CU Sketch : For CU sketches as atomic sketches, the only difference from the CM sketches is that the updater performs conservative updates. More specifically, for an incoming item, the updater of the CU sketch only increments those counters that have the minimum value among all its mapped counters. The Stair CU sketch consists of a set of CU sketches. The query process is the same as the Stair CM sketch. To insert an item e in the period k, instead of performing a conservative update for each CU sketch locally, the Stair CU sketch performs *global conservative update* for all relevant CU sketches. That is, for an incoming item, the Stair CU sketch first performs a query process to get the minimum value of all counters it is mapped to in all CU sketches, and increment those counters with the minimum value. In this way, the frequency estimation will be more accurate.

We do not change the way the Stair Sketch manipulates the atomic sketches. The more atomic sketches we use for recording a period, the more accurate the estimation will be. Therefore, by similar analysis as we do for the Stair Bloom filter, the Stair Sketch in general is a *time gradual* estimation structure. We shall present our experiment data to further show that in the experiment section.

D. The Stair Sketch for Range Query

In the optimized Stair Sketch, we can also perform *range query* with little time overhead. The range query is to ask the aggregate information of item e in a number of consecutive periods. For example, for membership query, the range query is to ask questions like whether the item e has occurred between l-th period and r-th period ($l \leq r$). For frequency estimation, the range query is to ask the frequency of an item e between l-th period and r-th period. We can see that range query is a generalized version of single-period query.

Since we use the shifting technique, the information of an item e in consecutive periods is stored in a sequence of consecutive cells. By memory locality, it would rarely cause more overhead to access some consecutive cells than to access just one cell. Therefore, we only need to select the atomic sketches that record the information of the period range, then we can answer range queries relatively fast.

V. Mathematical Analysis

In this section, we continue to use Bloom filters, CM sketches, and CU sketches as case studies, and analyze the error bounds of the Stair sketch when applied to them. We first introduce some preliminaries.

A. Preliminary

Given a data stream $\mathcal{S}$. Let $f_{(e,k)}$ be the statistic of item e in the k-th period, and $\hat{f}_{(e,k)}$ be the estimate of $f_{(e,k)}$ given by the data stream processing algorithms. In addition, let $f_{(e,[k_l,k_r])}$ be the statistic of item e within periods k_l to k_r, and $\hat{f}_{(e,[k_l,k_r])}$ its estimate. For a membership query, $f_{(e,k)}$ indicates whether item e occurs in the k-th period, and $f_{(e,[k_l,k_r])}$ indicates whether an item e occurs in periods from k_l to k_r. Formally,

$$f_{(e,k)} = \bigvee_{i=1}^{n} 1_{\{e_i=e \wedge t_i=k\}}; \quad f_{(e,[k_l,k_r])} = \bigvee_{k=k_l}^{k_r} f_{(e,k)}. \quad (4)$$

For frequency estimation, $f_{(e,k)}$ indicates the number of times the item e occurs in the k-th period, and $f_{(e,[k_l,k_r])}$ indicates the number of times item e occurs in periods from k_l to k_r. Formally,

$$f_{(e,k)} = \sum_{i=1}^{n} 1_{\{e_i=e \wedge t_i=k\}}; \quad f_{(e,[k_l,k_r])} = \sum_{k=k_l}^{k_r} f_{(e,k)}. \quad (5)$$

B. The Stair Bloom Filter

We first give the probability that the estimation $\hat{f}_{(e,k)}$ of $f_{(e,k)}$ is a false positive when using the shifting Bloom filter as the compound sketch.

Theorem 1. *Given a data stream $\mathcal{S}$ and a compound sketch M (i.e., a shifting Bloom filter) with w bits and d hash functions, M records the data from the L-th to the R-th period. Given a query $\langle e, k\rangle$, where $e \in \mathcal{U}$ and $k \in [L, R]$, we have*

$$\Pr\left\{\hat{f}_{(e,k)} \neq f_{(e,k)}\right\} \leqslant \left(1 - e^{-\frac{d}{w}\cdot\left(\sum_{k'=L}^{R}\sum_{e'=1}^{N} f_{(e',k')}\right)}\right)^d. \quad (6)$$

Proof. *Since only false positives will occur in the query when using the Bloom filter as the compound sketch, we have*

$$\Pr\left\{\hat{f}_{(e,k)} \neq f_{(e,k)} \mid f_{(e,k)} = 1\right\} = 0. \quad (7)$$

When item e does not occur in the k-th period, i.e., $f_{(e,k)} = 0$, $\forall 1 \leqslant i \leqslant d$, we have

$$\begin{aligned} &\Pr\left\{M\left[(h_i(e)+k) \bmod w\right] = 0 \mid f_{(e,k)} = 0\right\} \\ &= \prod_{e'=1}^{N}\left(\frac{w - \sum_{k'=L}^{R} f_{(e',k')}}{w}\right)^d \\ &= \prod_{e'=1}^{N} e^{-\frac{d}{w}\cdot\left(\sum_{k'=L}^{R} f_{(e',k')}\right)} = e^{-\frac{d}{w}\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}. \quad (8)\end{aligned}$$

To simplify the form, we follow the convention [3] to write $\approx$ to $=$ in the above formula. Since $\hat{f}_{(e,k)} = 1$ requires all d mapped bits to be 1, we have

$$\begin{aligned}
&\Pr\left\{\hat{f}_{(e,k)} \neq f_{(e,k)} \mid f_{(e,k)} = 0\right\} \\
=&\Pr\left\{\hat{f}_{(e,k)} = 1 \mid f_{(e,k)} = 0\right\} \\
=&\prod_{i=1}^{d} \Pr\left\{M\left[(h_i(e)+k) \bmod w\right] = 1 \mid f_{(e,k)} = 0\right\} \\
=&\left(1 - e^{-\frac{d}{w}\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}\right)^{d}. \qquad (9)
\end{aligned}$$

To sum up, we have

$$\begin{aligned}
&\Pr\left\{\hat{f}_{(e,k)} \neq f_{(e,k)}\right\} \\
=&\begin{aligned}&\Pr\left\{\hat{f}_{(e,k)} \neq f_{(e,k)} \mid f_{(e,k)} = 0\right\} \cdot \Pr\left\{f_{(e,k)} = 0\right\} \\ +&\Pr\left\{\hat{f}_{(e,k)} \neq f_{(e,k)} \mid f_{(e,k)} = 1\right\} \cdot \Pr\left\{f_{(e,k)} = 1\right\}\end{aligned} \\
\leqslant&\left(1 - e^{-\frac{d}{w}\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}\right)^{d}. \qquad (10)
\end{aligned}$$

Considering that $\sum_{e'=1}^{N} f_{(e',k')}$ has an upper bound $\mathcal{N}$, we can derive a looser error bound $\left(1 - e^{-\frac{d}{w}\cdot(R-L+1)\cdot\mathcal{N}}\right)^{d}$, which can be minimized by setting $d = \frac{w\cdot\ln 2}{(R-L+1)\cdot\mathcal{N}}$. Then we give the probability that the estimation $\hat{f}_{(e,[k_l,k_r])}$ is a false positive when using the Bloom filter as the compound sketch.

Theorem 2. *Given a data stream $\mathcal{S}$ and a compound sketch M with w bits and d hash functions, M records the data from the L-th to the R-th period. Given a range query $(e,[k_l,k_r])$, where $e \in \mathcal{U}$ and $[k_l,k_r] \subset [L,R]$, we have*

$$\begin{aligned}
&\Pr\left\{\hat{f}_{(e,[k_l,k_r])} \neq f_{(e,[k_l,k_r])}\right\} \\
\leqslant&\left(1 - e^{-\frac{d}{w}\cdot\left((R-L+k_r-k_l+1)\cdot\sum_{e'=1}^{N} f_{(e',[L,R])}\right)}\right)^{d}. \qquad (11)
\end{aligned}$$

Proof. *We first analyze the false positives of a range query.*

$$\begin{aligned}
&\Pr\left\{\hat{f}_{(e,[k_l,k_r])} = 1 \mid f_{(e,[k_l,k_r])} = 0\right\} \\
=&\Pr\left\{\exists\, k \in [k_l,k_r], \hat{f}_{(e,k)} = 1 \mid \forall\, k \in [k_l,k_r], f_{(e,k)} = 0\right\} \\
=&\Pr\left\{\begin{array}{c}\exists\, k \in [k_l,k_r], \forall\, i \in [1,d], \\ M\left[(h_i(e)+k) \bmod w\right] = 1 \\ \mid \forall\, k \in [k_l,k_r], f_{(e,k)} = 0\end{array}\right\} \\
\leqslant&\Pr\left\{\begin{array}{c}\forall\, i \in [1,d], \exists\, k_i \in [k_l,k_r], \\ M\left[(h_i(e)+k_i) \bmod w\right] = 1 \\ \mid \forall\, k \in [k_l,k_r], f_{(e,k)} = 0\end{array}\right\} \\
=&\prod_{i=1}^{d} \Pr\left\{\begin{array}{r}\exists\, k_i \in [k_l,k_r], M\left[(h_i(e)+k_i) \bmod w\right] = 1 \\ \mid \forall\, k \in [k_l,k_r], f_{(e,k)} = 0\end{array}\right\} \\
\leqslant&\prod_{i=1}^{d}\left(1 - \left(\prod_{e'=1}^{N} \frac{w - (R-L+k_r-k_l+1)\cdot f_{(e',[L,R])}}{w}\right)^{d}\right) \\
=&\left(1 - e^{-\frac{d}{w}\cdot\left((R-L+k_r-k_l+1)\cdot\sum_{e'=1}^{N} f_{(e',[L,R])}\right)}\right)^{d}. \qquad (12)
\end{aligned}$$

We can get the following inequality through the conditional probability formula.

$$\begin{aligned}
&\Pr\left\{\hat{f}_{(e,[k_l,k_r])} \neq f_{(e,[k_l,k_r])}\right\} \\
=&\begin{aligned}&\Pr\left\{\hat{f}_{(e,[k_l,k_r])} = 0 \mid f_{(e,[k_l,k_r])} = 1\right\} \cdot \Pr\left\{f_{(e,[k_l,k_r])} = 1\right\} \\ +&\Pr\left\{\hat{f}_{(e,[k_l,k_r])} = 1 \mid f_{(e,[k_l,k_r])} = 0\right\} \cdot \Pr\left\{f_{(e,[k_l,k_r])} = 0\right\}\end{aligned} \\
\leqslant&\left(1 - e^{-\frac{d}{w}\cdot\left((R-L+k_r-k_l+1)\cdot\sum_{e'=1}^{N} f_{(e',[L,R])}\right)}\right)^{d}. \qquad (13)
\end{aligned}$$

According to Theorem 1, we can derive the error probability of the Stair Bloom filter when querying $\langle e,k'\rangle$ in the k-th time period, and show that it satisfies the time gradualness.

Theorem 3. *Given a data stream $\mathcal{S}$ and a Stair Bloom filter of m layers. Given a query $\langle e,k'\rangle$, and assuming it is currently in the k-th period. The compound sketch M_{l,p_l} in the l-th ($m' \leqslant l \leqslant m$) layer records the data from the L_l-th to the R_l-th period, with m' and p_l as defined in Section IV-B. Its parameters are $\{w_l, d_l\}$. We have*

$$\Pr\left\{\hat{f}_{BF} \neq f\right\} \leqslant \prod_{l=m'}^{m}\left(1 - e^{-\frac{d_l}{w_l}\cdot\left(\sum_{k''=L_l}^{R_l}\sum_{e'=1}^{N} f_{(e',k'')}\right)}\right)^{d_l}. \qquad (14)$$

Using $\mathcal{N}$ instead of $\sum_{e'=1}^{N} f_{(e',k'')}$, we can derive a looser error bound $\prod_{l=m'}^{m}\left(1 - e^{-\frac{d_l}{w_l}\cdot\left((R_l-L_l+1)\cdot\mathcal{N}\right)}\right)^{d_l}$, where m' is determined by k and k'. We can infer the following properties.

- ***error gradualness***. The smaller k' is, the larger m' is, and thus the larger the error bound.
- ***time stability***. When $k' = k$, there is $m' = 1$, and thus the error bound is only related to $\mathcal{N}$ rather than k.

Since the Stair Bloom filter satisfies both error gradualness and time stability, it satisfies time gradualness.

Recommended parameter configuration: We first propose the following two principles. 1) Let $w_l \propto d_l \times (M_{l,0}.\texttt{end} - M_{l,0}.\texttt{begin}+1)$, so that the bits in different compound sketch will be set to 1 with approximately equal probability p. 2) Let $d_l = 1$ $(1 \leqslant l \leqslant m-1)$, so that the estimation error $\mathcal{E}(k,k')$ is approximately proportional to the time interval $(k-k'+1)$. With the constraints of these two principles, we can set the only variable parameter d_m to make $p \approx \frac{1}{2}$, so as to make the best use of memory.

Comparison with the Hokusai: Suppose both data structures record $\mathcal{L} = 2^{m-1}$ periods, and it is currently in the $\mathcal{L}$-th period. By setting memory budget $\mathcal{M} = \frac{m\cdot\mathcal{L}\cdot\mathcal{N}}{\ln 2}$, according to the recommendation, $d_m = m-1$ and $p = \frac{1}{2}$. So we have

$$\mathcal{E}(\mathcal{L},k') = 2^{-2\cdot(m-1)+\left\lceil\log\left(\mathcal{L}-k'+1\right)\right\rceil} \qquad (15)$$

For Hokusai Bloom filter (see Section VI-A for details) using 2 hash functions as recommended [10], we have

$$\mathcal{E}'(\mathcal{L},k') = \left(1 - 2^{-\frac{m-1}{m}\cdot 2^{-(m-2)+\left\lfloor\log\left(\mathcal{L}-k'+1\right)\right\rfloor}}\right)^{2} \qquad (16)$$

There is always $\mathcal{E}(\mathcal{L},k') < \mathcal{E}'(\mathcal{L},k')$ when $m \geqslant 4$, *i.e.*, Stair Bloom filter has a smaller error bound than Hokusai.

C. The Stair CM Sketch

We first give the probability that the estimation error exceeds a certain ε when using the shifting CM sketch as the compound sketch.

Theorem 4. *Given a data stream $\mathcal{S}$ and a compound sketch M (i.e., a shifting CM sketch) with w counters and d hash functions. M records the data from the L-th to the R-th period. Given a query $\langle e, k\rangle$, where $e \in \mathcal{U}$ and $k \in [L, R]$, we have*

$$\Pr\left\{\left|\hat{f}_{(e,k)} - f_{(e,k)}\right| > \varepsilon\right\} \leqslant \left(\frac{d\cdot\left(\sum_{k'=L}^{R}\mathcal{N}_{k'}\right)}{w\cdot\varepsilon}\right)^{d}. \quad (17)$$

Proof. *Considering the expectation of the error in a mapped counter, $\forall 1 \leqslant i \leqslant d$, we have*

$$\begin{aligned}&E\left(M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)}\right)\\ &= \sum_{e'=1, e'\neq e}^{N}\left(\sum_{k'=L}^{R}\frac{1}{w}\cdot f_{(e',k')}\right)\cdot d\\ &\leqslant \frac{d\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}{w}.\end{aligned} \quad (18)$$

According to the Markov inequality, we have

$$\begin{aligned}&\Pr\left\{M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)} > \varepsilon\right\}\\ &\leqslant \frac{E\left(M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)}\right)}{\varepsilon}\\ &\leqslant \frac{d\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}{w\cdot\varepsilon}.\end{aligned} \quad (19)$$

Considering that the CM sketch has only one-sided errors, in summary, we have

$$\begin{aligned}&\Pr\left\{\left|\hat{f}_{(e,k)} - f_{(e,k)}\right| > \varepsilon\right\}\\ &= \Pr\left\{\left(\min_{1\leqslant i\leqslant d} M\left[(h_i(e)+k) \bmod w\right]\right) - f_{(e,k)} > \varepsilon\right\}\\ &= \prod_{i=1}^{d}\Pr\left\{M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)} > \varepsilon\right\}\\ &\leqslant \left(\frac{d\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}{w\cdot\varepsilon}\right)^{d}.\end{aligned} \quad (20)$$

Considering that $\mathcal{N}_{k'}$ has an upper bound $\mathcal{N}$, we can also derive a looser error bound $\left(\frac{d\cdot(R-L+1)\cdot\mathcal{N}}{w\cdot\varepsilon}\right)^{d}$. Then we give the probability that the range query error exceeds a certain ε when using the shifting CM sketch as the compound sketch.

Theorem 5. *Given a data stream $\mathcal{S}$ and a compound sketch M with w cells and d hash functions. M records the data from the L-th to the R-th period. Given a range query $(e, [k_l, k_r])$, where $e \in \mathcal{U}$ and $[k_l, k_r] \subset [L, R]$, we have*

$$\begin{aligned}&\Pr\left\{\left|\hat{f}_{(e,[k_l,k_r])} - f_{(e,[k_l,k_r])}\right| > \varepsilon\right\}\\ &\leqslant \left(\frac{d\cdot(k_r-k_l+1)\cdot\left(\sum_{k'=L}^{R}\mathcal{N}_{k'}\right)}{w\cdot\varepsilon}\right)^{d}.\end{aligned} \quad (21)$$

Proof. *We first find an upper bound of this probability, that is*

$$\begin{aligned}&\Pr\left\{\hat{f}_{(e,[k_l,k_r])} - f_{(e,[k_l,k_r])} \geqslant \varepsilon\right\}\\ &= \Pr\left\{\left(\sum_{k=k_l}^{k_r}\min_{1\leqslant i\leqslant d} M\left[(h_i(e)+k) \bmod w\right]\right) - f_{(e,[k_l,k_r])} \geqslant \varepsilon\right\}\\ &\leqslant \Pr\left\{\left(\min_{1\leqslant i\leqslant d}\sum_{k=k_l}^{k_r} M\left[(h_i(e)+k) \bmod w\right]\right) - f_{(e,[k_l,k_r])} \geqslant \varepsilon\right\}\\ &= \prod_{i=1}^{d}\Pr\left\{\sum_{k=k_l}^{k_r}\left(M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)}\right) \geqslant \varepsilon\right\}\end{aligned} \quad (22)$$

According to Equation 18, we have

$$\begin{aligned}&E\left(\sum_{k=k_l}^{k_r}\left(M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)}\right)\right)\\ &= \sum_{k=k_l}^{k_r} E\left(M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)}\right)\\ &\leqslant (k_r-k_l+1)\cdot\left(\frac{d\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}{w}\right)\end{aligned} \quad (23)$$

Through the Markov inequality, we have

$$\begin{aligned}&\Pr\left\{\hat{f}_{(e,[k_l,k_r])} - f_{(e,[k_l,k_r])} \geqslant \varepsilon\right\}\\ &\leqslant \prod_{i=1}^{d}\frac{E\left(\sum_{k=k_l}^{k_r}\left(M\left[(h_i(e)+k) \bmod w\right] - f_{(e,k)}\right)\right)}{\varepsilon}\\ &\leqslant \left(\frac{d\cdot(k_r-k_l+1)\cdot\left(\sum_{e'=1}^{N}\sum_{k'=L}^{R} f_{(e',k')}\right)}{w\cdot\varepsilon}\right)^{d}.\end{aligned} \quad (24)$$

According to Theorem 4, we can derive the error bound of the frequency estimation of the Stair CM sketch when querying $\langle e, k'\rangle$ in the k-th time period.

Theorem 6. *Given a data stream $\mathcal{S}$ and a Stair CM sketch of m layers. Given a query $\langle e, k'\rangle$, and assuming it is currently in the k-th period. The compound sketch M_{l,p_l} in the l-th ($m' \leqslant l \leqslant m$) layer records the data from the L_l-th to the R_l-th period, where m' and p_l are defined in Section IV-B. The parameters of M_{l,p_l} are $\{w_l, d_l\}$. We have*

$$\Pr\left\{\left|\hat{f}_{CM} - f\right| \geqslant \varepsilon\right\} \leqslant \prod_{l=m'}^{m}\left(\frac{d_l\cdot\left(\sum_{k''=L_l}^{R_l}\mathcal{N}_{k''}\right)}{w_l\cdot\varepsilon}\right)^{d_l}. \quad (25)$$

By replacing $\mathcal{N}_{k''}$ with $\mathcal{N}$, we can use the same analysis as described in Section V-B to obtain that the Stair CM sketch satisfies both error gradualness and time stability, thus having time gradualness.

The Stair CU Sketch. Since the CU sketch is a conservative version of the CM sketch, the error bound of the Stair CM sketch is also applicable to the Stair CU sketch. This means that the Stair CU sketch also has time gradualness.

Recommended parameter configuration: We follow the two principles of parameter configuration proposed in Section V-B, and we set the only variable parameter $d_m = 4$, just as many sketches recommend to achieve the best accuracy in practice.

VI. Experimental Results

In this section, we conduct extensive experiments to test the properties of the Stair sketch and compare it with state-of-the-art algorithms. Our comparison covers mainly three aspects: time gradualness, estimation accuracy, and query delay.

A. Experimental Setup

We now describe our experimental setup, including the state-of-the-art algorithms we compare against, the datasets used, the evaluation metrics, and default settings.

1) Comparison with State-of-the-art:

The Stair sketch can perform both membership queries and frequency estimation, while state-of-the-art algorithms only handle one of them. Therefore, for comparison, we select specifically among the many state-of-the-art algorithms for membership queries and frequency estimation.

Frequency Estimation: For frequency estimation, we choose the Hokusai item aggregation CM sketch [9] (HCM) and Time-Adaptive CM sketch [11] (ADA-CM). For ADA-CM, we use the increasing sequence $f(j) = j$ for the j-th time period, which was proposed in the original paper. We compare the Stair CM sketch (SCM) and the Stair CU sketch (SCU) with them. We do not choose persistent data sketching in our comparison since its memory consumption is dynamic when running. We actually implemented and tested it, and found that it can consume as much as 10 GB of memory on our datasets.

Membership Query: We implement PBF-2 [13] with a minor optimization: checking more relevant Bloom Filters for queries. This optimization greatly improves its accuracy at the cost of a slightly lower query speed. Apart from PBF, few algorithms deal with time-sensitive membership queries. As claimed in Section I, only Hokusai satisfies the two aspects of time gradualness, but it can only handle frequency estimation. We find that the core operation of the item aggregation Hokusai is the CM sketch compression, and the Bloom filter can easily be compressed as well [10]. Therefore, by changing the CM sketch in Hokusai into a Bloom filter, we can extend the Hokusai algorithm to support membership queries, as one of our baseline algorithms. We call this algorithm the Hokusai Bloom filter (HBF). We compare the Stair Bloom filter (SBF) with PBF and HBF.

2) Datasets:

We use the following datasets.

- **CAIDA:** We use CAIDA [49], a public dataset that includes anonymous real-world network traces from high-speed Internet Backbone links. More specifically, we use network packets coming from an 16-minute consecutive trace from CAIDA as the data stream. Consequently, there are 1.6×10^6 different items in the 4.2×10^8 items in total.
- **Web page:** The Web page dataset is built from a number of web HTML documents [50]. There are 1×10^6 distinct items in a total of around 3.2×10^7 items.
- **Synthetic Datasets:** Using open-source performance testing tool Web Polygraph [51], we generate a number of datasets following the Zipf [52] distribution. There are 952888 distinct items in a total of 3.2×10^7 items.

Since the original Web page and synthetic datasets do not include arrival timestamps of the items, we generate n (the number of items) independent random variables uniformly distributed on $(0, 16)$, sort them in ascending order, and use them as the arrival time of each item in turn. In this way, these datasets cover exactly 16 minutes and can be treated as timestamped data streams, just like the CAIDA dataset. Sorting them in ascending order ensures that these timestamps have the same distribution as n arrival times of a Poisson process if the number of arrivals in the first 16 minutes is exactly n.

3) Evaluation Metrics:

Three typical metrics to measure the accuracy of data streams estimation are shown below. In the following, we use T to represent the total number of time periods, $\mathcal{S}$ to represent the data stream, and $\langle e, k\rangle \in \mathcal{S}$ means that item e arrived in the k-th time period at least once.

- **FPR** (False Positive Rate): For each time period k, we define the FPR_k as $\frac{1}{|E_k|}\sum_{e\in E_k}\hat{f}_{(e,k)}$. Where $E_k = \{e|\langle e, k\rangle \notin \mathcal{S} \wedge \exists k', \langle e, k'\rangle \in \mathcal{S}\}$ and $\hat{f}_{(e,k)}$ represents the membership query result of item e in the k-th time period.
- **AAE** (Average Absolute Error): For each period k, we define the AAE_k as $\frac{1}{|W_k|}\sum_{e\in W_k}|f_{(e,k)} - \hat{f}_{(e,k)}|$. Where $W_k = \{e|\langle e, k\rangle \in \mathcal{S}\}$, and $f_{(e,k)}$ and $\hat{f}_{(e,k)}$ represent the actual and estimated frequency of item e in the k-th time period respectively.
- **ARE** (Average Relative Error): For each period k, we define the ARE_k as $\frac{1}{|W_k|}\sum_{e\in W_k}\frac{|f_{(e,k)}-\hat{f}_{(e,k)}|}{f_{(e,k)}}$. Where $W_k = \{e|\langle e, k\rangle \in \mathcal{S}\}$, and $f_{(e,k)}$ and $\hat{f}_{(e,k)}$ represent the actual and estimated frequency of item e in the k-th time period respectively.

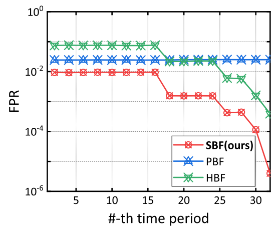
To test accuracy as well as time gradualness of the algorithms, besides typical metrics above, we also adopt weighted error metrics, which add a penalty for estimation errors on more recent time periods. We use two kinds of queries to measure the performance, queries of a single time period and multiple time periods. For the k-th time period, the weight is $\frac{1}{T-k+1}$. For a query of multiple times periods from l to r, its weight is assigned as the summary of weights of time period $l, l+1, \cdots r$.

Since weighted error metrics follow the same weighting rule, we only give the formal definition of Weighted False Positive Rate (WFPR) below due to space limitations. **WAAE** and **WARE** are defined similarly as **WFPR**.

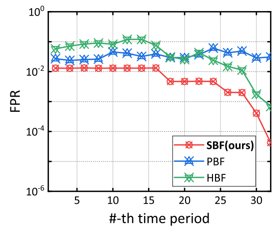
- **WFPR** (Weighted FPR, single time period): WFPR is defined as $\sum_{k=1}^{T} \frac{\text{FPR}_k}{T-k+1}$.
- **WFPR** (Weighted FPR, multiple time periods): For all $1 \leq l \leq r \leq T$, we define $\text{FPR}_{[l,r]}$ as $\frac{1}{|E_{[l,r]}|}\sum_{e\in E_{[l,r]}} \hat{f}_{(e,[l,r])}$. Where $E_{[l,r]} = \{e|\forall l \leq k \leq r, \langle e,k\rangle \notin \mathcal{S} \wedge \exists k', \langle e,k'\rangle \in \mathcal{S}\}$. WFPR is defined as $\sum_{l=1}^{T}\sum_{r=l}^{T} \text{FPR}_{[l,r]} \sum_{k=l}^{r} \frac{1}{T-k+1}$.
- **AMA** (Average Memory Access): AMA is defined as $\frac{1}{n}\sum_{i=1}^{n} MA(e_i,k_i)$. Where n is the number of testing samples and $MA(e_i,k_i)$ denotes the number of memory accesses required to query the information of item e_i within time period k_i. We use AMA to evaluate the memory latency of queries.

4) Default Settings:

By default, we split the data stream into 32 consecutive time periods of equal length, and the parameters of all the algorithms are set to maintain the information of 32 time periods only. All the algorithms are implemented in C++. By default, the SCM, SCU, and SBF have four layers. For SCM, SCU, and SBF, the sketches in the bottom layer (according to Fig. 4) use 4 hash functions while others use 1. For other algorithms, each sketch uses 2 hash functions. The hash function we adopted in these algorithms is the Bob hash [53]. To keep the error rate of the algorithms at a moderate level (*i.e.*, not too high or too low, suitable for practical usage and comparison), we set a reasonable memory size for each set of experiments, which we specify in the caption of the corresponding figure. All programs are run on a server with 64 GB system memory and dual 6-core CPUs (24 threads, Intel(R) Xeon(R) CPU E5-2620 @2.00GHz).

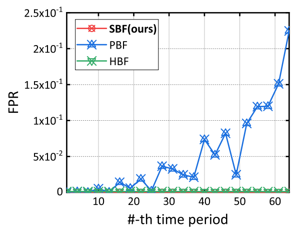


(a) CAIDA, 10 MB.

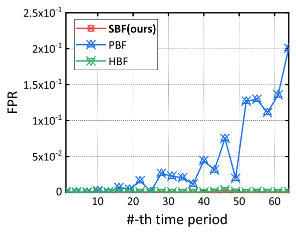


(b) Web page, 4 MB.

Fig. 6: Error gradualness for membership query task.



(a) CAIDA, 6 MB.

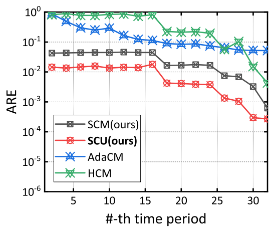


(b) Web page, 2 MB.

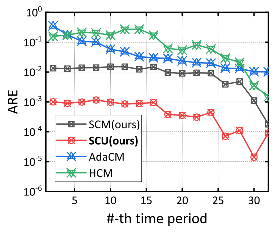
Fig. 7: Time stability for membership query task.

B. Evaluation of Time Gradualness

In this subsection, we evaluate time gradualness. As described in Section I, time gradualness has two aspects: error gradualness and time stability. To evaluate error gradualness, we build the data structures over the whole data stream and check their accuracy in estimating each time period. We expect that an error gradual algorithm would have a more accurate estimation in recent periods. To evaluate time stability, for each algorithm, each time it finishes processing the items in one period, we check its estimation accuracy for this period. We expect that a time stable algorithm would have a bounded error rate in estimating the latest period.

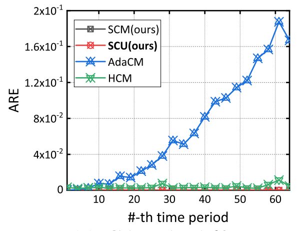


(a) CAIDA, 160 MB.

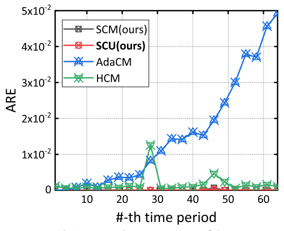


(b) Web page, 100 MB.

Fig. 8: Error gradualness for frequency estimation task.



(a) CAIDA, 160 MB.



(b) Web page, 60 MB.

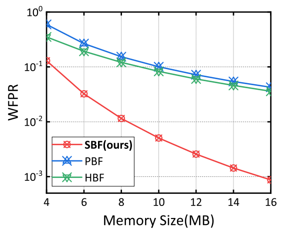
Fig. 9: Time stability for frequency estimation task.

Error gradualness for membership queries (Fig. 6): After letting the PBF, HBF, and SBF process the whole data stream, we test their FPR (False Positive Rate) in estimating items in 32 time periods respectively. We find that the HBF and SBF have error gradualness, but the PBF does not. This result corresponds with our previous expectation. Besides, the SBF achieves the best result among them.

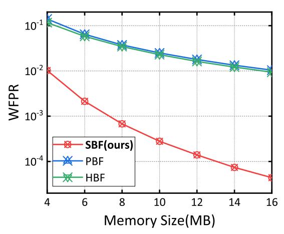
Time stability for membership queries (Fig. 7): We split the datasets into 64 consecutive time periods of equal length. After each time period comes to an end, we test the FPR (False Positive Rate) of PBF, HBF, and SBF when estimating the items in the latest time period. As expected, we find that despite fluctuations, the FPR of the SBF and HBF stays below a relatively small upper-bound, so they are time stable. The FPR of the PBF increases over time, so it is time unstable. The SBF achieves the best results among them.

Error gradualness for frequency estimation (Fig. 8): After letting the HCM, ADA-CM, and our SCM and SCU process the whole data stream, we test their ARE (Average Relative Error) when estimating items in 32 time periods. As expected, we find that the four algorithms all have error gradualness. Besides, the SCM and SCU achieve the best results among them.

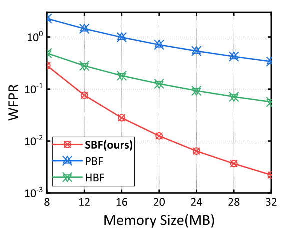
Time stability for frequency estimation (Fig. 9): We split the datasets into 64 consecutive time periods of equal length. After each time period comes to an end, we test the ARE (Average Relative Error) of the HCM, ADA-CM, SCM, and SCU at estimating the items in the latest time period. As expected, we find that the HCM, SCM, and SCU have time stability, despite periodical fluctuations. The ARE of the SCM and SCU stay below a relatively small upper-bound. ADA-CM is not time stable, hence it would be difficult to deploy ADA-CM in practice. Among the four algorithms, the SCM and SCU achieve the best results.



(a) CAIDA dataset

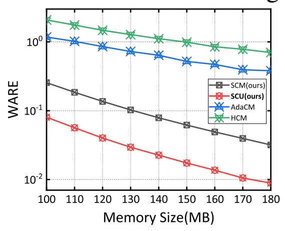


(b) Web page dataset

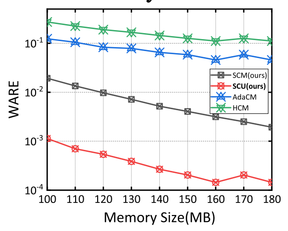


(c) Synthetic dataset

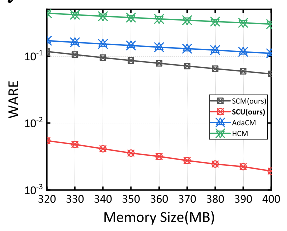
Fig. 10: WFPR *vs.* memory for membership query task.



(a) CAIDA dataset

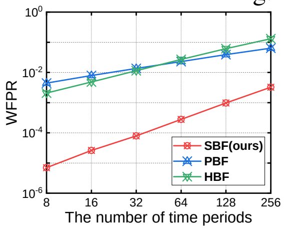


(b) Web page dataset

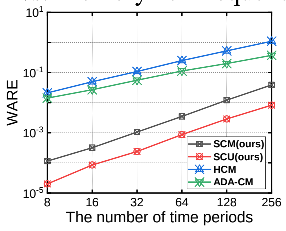


(c) Synthetic dataset

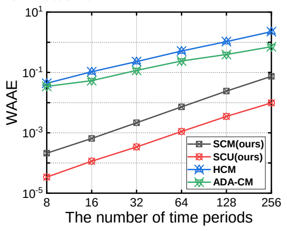
Fig. 11: WARE *vs.* memory for frequency estimation task.



(a) Membership query, 20 MB.

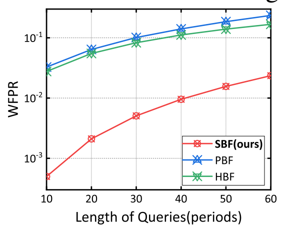


(b) Frequency estimation, 300 MB.

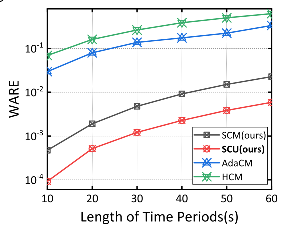


(c) Frequency estimation, 300 MB.

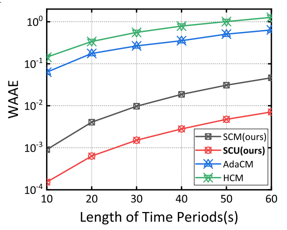
Fig. 12: Weighted Error *vs.* the number of time periods.



(a) Membership query, 10 MB.



(b) Frequency estimation, 300 MB.



(c) Frequency estimation, 300 MB.

Fig. 13: Weighted Error *vs.* length of time periods.

C. Evaluating Accuracy from The Perspective of Error Gradualness

Now, we test and compare the estimation accuracy of different algorithms. Since recent information is more valuable as claimed in Section I, we want to test the accuracy of the algorithms from the perspective of error gradualness. Therefore, instead of common metrics such as the FPR for membership queries and the AAE, the ARE for frequency estimation, we use their weighted versions, defined earlier in this section. The weighted error metric can be seen as the aggregate error of algorithms in many time periods, adding more weight to more recent time periods.

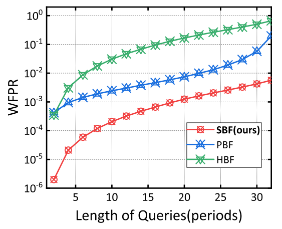
In the following three sets of experiments, we test the weighted error of the algorithms with various memory sizes.

WFPR *vs.* memory (Fig. 10): We vary the total memory size of the PBF, HBF and SBF, and test their WFPR on three different datasets. We find that the SBF achieves the highest accuracy for various total memory consumptions. Generally, the WFPR of the SBF is two orders of magnitude lower than the other algorithms.

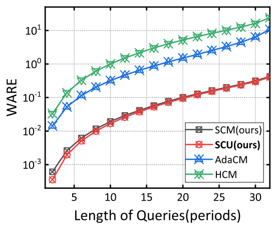
WARE *vs.* memory (Fig. 11): We vary the total memory size of the HCM, ADA-CM, SCM, and SCU, and test their WARE on three different datasets. We find that the SCU and SCM achieve the highest and the second highest accuracy across the different amounts of total memory. Generally, the WARE of the SCM is one to two orders of magnitude lower than that of the ADA-CM and HCM, and the WARE of the SCU is two orders of magnitude lower than that of the ADA-CM and HCM.

In the following three sets of experiments, we vary either the number or the length of the time periods, and test the performance of the algorithms in these different scenarios[1]. Furthermore, we try range queries across multiple time periods. Since the algorithms perform similarly for the different datasets, we only use the CAIDA dataset in the following three sets of experiments.

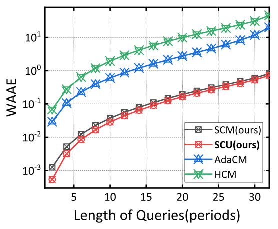
[1]The length of the whole input data stream may vary with the change of one of the two parameters.



(a) Membership query, 30 MB.



(b) Frequency estimation, 300 MB.



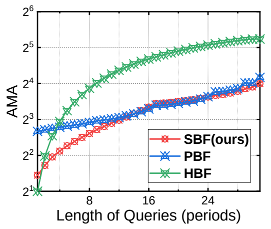
(c) Frequency estimation, 300 MB.

Fig. 14: Weighted Error *vs.* length of queries.

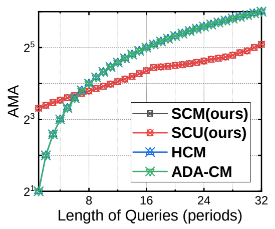
Weighted error *vs.* the number of time periods (Fig. 12): We vary only the number of time periods, and test the weighted error of all the algorithms above. We find that the estimation error of the algorithms increases with the number of time periods. This is because a larger number of time periods considered increases the amount and the granularity of the information recorded in the data structure. For a fixed memory size, the accuracy naturally decreases for larger amounts of information recorded in the data structure. For the number of time periods considered, the weighted error of the Stair sketch is always at least 9.5 times lower than other state-of-the-art algorithms.

Weighted error *vs.* length of time periods (Fig. 13): We vary only the length of the time periods, and test the weighted error of all the algorithms. We find that generally, the estimation error increases the longer the time period. Again, this is due to the increased amount and granularity of the information recorded in the data structure. For a fixed memory size, the accuracy decreases as the amount of information recorded in the data structure increases. For the considered time period lengths, the weighted error of the Stair sketch is always at least 7 times lower than other state-of-the-art algorithms.

Weighted error for range queries of multiple time periods (Fig. 14): For different query lengths (*i.e.*, the number of time periods the range query covers), we test the weighted error for multiple time periods (defined at the beginning of this section). We find that the estimation error increases as the number of time periods covered increases. This is because the estimation error of each single time period contributes to the total error of the query, both for membership queries and frequency estimation. Across the different time periods covered, the weighted error of the Stair sketch is always at least 5 times lower than other state-of-the-art algorithms.



(a) Membership query, 15 MB



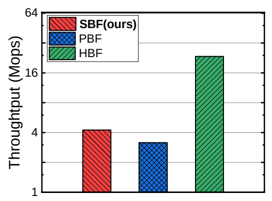
(b) Frequency estimation, 120 MB

Fig. 15: AMA *vs.* length of queries.

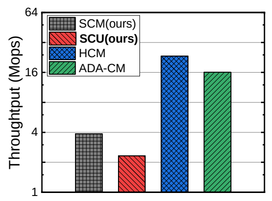
D. Evaluation of Efficiency

Here, we compare the query delay of different algorithms. We use the average number of memory accesses (AMA) as the metric. Since the algorithms perform similarly across different datasets, we only use the CAIDA dataset in the following experiments.

AMA for range queries of multiple time periods[2] (Fig. 15): For different query lengths (*i.e.*, the number of time periods the range query covers), we test the AMA of different algorithms. As expected, we find that the AMA increases as the number of considered periods increases. Among the algorithms, the AMA of the Stair sketch increases the least, and generally, the Stair sketch needs fewer memory accesses than other algorithms.



(a) Membership query, 15 MB



(b) Frequency estimation, 120 MB

Fig. 16: Throughput.

Throughput (Fig. 16): We find that the throughput of Stair sketches is higher than that of the PBF, but lower than that of Hokusai sketches and ADA-CM. This is because Stair sketches and PBF focus on reducing range query delay. When recording $\mathcal{L}$ time periods, the insertion complexity and range query complexity of Stair sketches and PBF are both $O(\log \mathcal{L})$. In contrast, the insertion complexity of Hokusai sketches and the ADA-CM are $O(1)$, but their range query complexity are $O(\mathcal{L})$.

VII. Conclusion

Memorizing when an item appears is an important task in data stream processing. Also, one tends to pay more attention to the recent appearance of items than to older ones. We propose a novel data stream processing structure named the Stair sketch to memorize recent events with more accuracy. The key idea is to organize the memory in a stair shape. We show that our scheme achieves *time gradualness*, which provides both *time stability* and *error gradualness*. The Stair sketch is also generic, in that it can be applied to different types of sketches. We illustrate this by using the Stair sketch with Bloom filters [3], CM sketches [4], and CU sketches [5]. We deploy the Stair sketch in two typical tasks, membership queries, and frequency estimation, and derive theoretical error bounds for both. Experiment results show that our approach outperforms state-of-the-art algorithms by more than $5\times$ in accuracy while providing comparable efficiency.

[2]The curves of the SCM and SCU coincide, and the curves of the ADA-CM and HCM coincide.

References

[1] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296, 2010.

[2] Kleber Vieira, Alexandre Schulter, Carlos Westphall, and Carla Westphall. Intrusion detection for grid and cloud computing. *It Professional*, 12(4):38–43, 2009.

[3] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

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

[5] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 323–336, 2002.

[6] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking*, 8(3):281–293, 2000.

[7] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

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

[9] Sergiy Matusevych, Alexander J Smola, and Amr Ahmed. Hokusai—sketching streams in real time. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 594–603, 2012.

[10] Michael Mitzenmacher. Compressed bloom filters. *IEEE/ACM transactions on networking*, 10(5):604–612, 2002.

[11] Anshumali Shrivastava, Arnd Christian Konig, and Mikhail Bilenko. Time adaptive sketches (ada-sketches) for summarizing data streams. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1417–1432, 2016.

[12] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*, pages 795–810, 2015.

[13] Yanqing Peng, Jinwei Guo, Feifei Li, Weining Qian, and Aoying Zhou. Persistent bloom filter: Membership testing for the entire history. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1037–1052, 2018.

[14] Source code related to the Stair sketch. https://github.com/Stair-Sketches/stair_sketch.

[15] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.

[16] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1449–1463, 2016.

[17] Nan Tang, Qing Chen, and Prasenjit Mitra. Graph stream summarization: From big bang to big crunch. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1481–1496, 2016.

[18] Daniel Ting. Count-min: Optimal estimation and tight error bounds using empirical error distributions. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2319–2328, 2018.

[19] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. Sketching linear classifiers over data streams. In *Proceedings of the 2018 International Conference on Management of Data*, pages 757–772, 2018.

[20] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. Cold filter: A meta-framework for faster and more accurate stream processing. In *Proceedings of the 2018 International Conference on Management of Data*, pages 741–756, 2018.

[21] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A general-purpose counting filter: Making every bit count. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 775–787, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Minmei Wang, Mingxun Zhou, Shouqian Shi, and Chen Qian. Vacuum filters: More space-efficient and faster replacement for bloom and cuckoo filters. *Proc. VLDB Endow.*, 13(2):197–210, October 2019.

[23] Alex D. Breslow and Nuwan S. Jayasena. Morton filters: Faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity. *Proc. VLDB Endow.*, 11(9):1041–1055, May 2018.

[24] Florin Rusu and Alin Dobra. Statistical analysis of sketch estimators. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 187–198, 2007.

[25] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proceedings of the VLDB Endowment*, 5(10), 2012.

[26] Qi Zhao, Mitsunori Ogihara, Haixun Wang, and Jun Xu. Finding global icebergs over distributed data sets. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 298–307, 2006.

[27] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, et al. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1071–1080, 2011.

[28] Biplob Debnath, Sudipta Sengupta, and Jin Li. Skimpystash: Ram space skimpy key-value store on flash-based storage. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 25–36, 2011.

[29] Jose M Faleiro, Alexander Thomson, and Daniel J Abadi. Lazy evaluation of transactions in database systems. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 15–26, 2014.

[30] Mingjie Tang, Yongyang Yu, Qutaibah M Malluhi, Mourad Ouzzani, and Walid G Aref. Locationspark: A distributed in-memory data management system for big spatial data. *Proceedings of the VLDB Endowment*, 9(13):1565–1568, 2016.

[31] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings 2003 VLDB Conference*, pages 464–475. Elsevier, 2003.

[32] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1093–1103, 2012.

[33] Haipeng Dai, Muhammad Shahzad, Alex X Liu, and Yuankun Zhong. Finding persistent items in data streams. *Proceedings of the VLDB Endowment*, 10(4):289–300, 2016.

[34] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid sketch: A sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment*, 10(11):1442–1453, 2017.

[35] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230, 2013.

[36] Kaushik Chakrabarti, Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. An efficient filter for approximate membership checking. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 805–818, 2008.

[37] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large rdf graphs. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 627–640, 2009.

[38] Haiqin Liu, Yan Sun, and Min Sik Kim. Fine-grained ddos detection scheme based on bidirectional count sketch. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2011.

[39] Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX conference on Hot topics in security*, pages 1–8, 2010.

[40] Shahabeddin Geravand and Mahmood Ahmadi. Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks*, 57(18):4047–4064, 2013.

[41] Arwa Alrawais, Abdulrahman Alhothaily, Chunqiang Hu, and Xiuzhen Cheng. Fog computing for the internet of things: Security and privacy issues. *IEEE Internet Computing*, 21(2):34–42, 2017.

[42] Udi Manber and Sun Wu. An algorithm for approximate membership checking with application to password security. *Information Processing Letters*, 50(4):191–197, 1994.

[43] Rui Li, Alex X Liu, Ann L Wang, and Bezawada Bruhadeshwar. Fast range query processing with strong privacy protection for cloud computing. *Proceedings of the VLDB Endowment*, 7(14):1953–1964, 2014.

[44] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *European Symposium on Algorithms*, pages 684–695. Springer, 2006.

[45] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Transactions on Networking*, 14:p.925–937, 2006.

[46] Domenico Ficara, Stefano Giordano, Gregorio Procissi, and Fabio Vitucci. Multilayer compressed counting bloom filters. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 311–315. IEEE, 2008.

[47] Saar Cohen and Yossi Matias. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 241–252, 2003.

[48] Tong Yang, Alex X Liu, Muhammad Shahzad, Yuankun Zhong, Qiaobin Fu, Zi Li, Gaogang Xie, and Xiaoming Li. A shifting bloom filter framework for set queries. *Proceedings of the VLDB Endowment*, 9(5), 2016.

[49] The CAIDA Anonymized Internet Traces. http://www.caida.org/data/overview/.

[50] Real-life transactional dataset. http://fimi.ua.ac.be/data/.

[51] Alex Rousskov and Duane Wessels. High-performance benchmarking with web polygraph. *Software*, 34(2):p.187–211, 2004.

[52] David M. W. Powers. Applications and explanations of zipf's law. *Advances in Neural Information Processing Systems*, 5(4):595–599, 1998.

[53] Robert J. Jenkins Jr. Hash website. http://burtleburtle.net/bob/hash/evahash.html Accessed July 21, 2021.