On-Off Sketch: A Fast and Accurate Sketch on Persistence

Yinda Zhang[*]
Peking University
hgdkgszyd@gmail.com

Jinyang Li[*]
Peking University
lijinyang@pku.edu.cn

Yutian Lei[†]
Xiangtan University
leiyutianchina@hotmail.com

Tong Yang[*‡]
Peking University
yangtongemail@gmail.com

Zhetao Li[†]
Xiangtan University
liztchina@hotmail.com

Gong Zhang[§]
Huawei
nicholas.zhang@huawei.com

Bin Cui[*¶]
Peking University
bin.cui@pku.edu.cn

ABSTRACT

Approximate stream processing has attracted much attention recently. Prior art mostly focuses on characteristics like frequency, cardinality, and quantile. Persistence, as a new characteristic, is getting increasing attention. Unlike frequency, persistence highlights behaviors where an item appears recurrently in many time windows of a data stream. There are two typical problems with persistence – persistence estimation and finding persistent items. In this paper, we propose the On-Off sketch to address both problems. For persistence estimation, using the characteristic that the persistence of an item is increased periodically, we compress increments when multiple items are mapped to the same counter, which significantly reduces the error. Compared with the Count-Min sketch, 1) in theory, we prove that the error of the On-Off sketch is always smaller; 2) in experiments, the On-Off sketch achieves around 6.17 times smaller error and 2.2 times higher throughput. For finding persistent items, we propose a technique to separate persistent and non-persistent items, further improving the accuracy. We show that the space complexity of our On-Off sketch is much better than the state-of-the-art (PIE), and it reduces the error up to 4 orders of magnitude and achieves 2.84 times higher throughput than prior algorithms in experiments.

PVLDB Reference Format:
Yinda Zhang, Jinyang Li, Yutian Lei, Tong Yang, Zhetao Li, Gong Zhang, and Bin Cui. On-Off Sketch: A Fast and Accurate Sketch on Persistence. PVLDB, 14(2): 128-140, 2021.
doi:10.14778/3425879.3425884

1 INTRODUCTION

Nowadays, with the increasing speed of data streams, it is more and more challenging to answer queries accurately on data streams using a limited amount of memory. Therefore, lots of interests have been given to approximate stream processing algorithms [1–7], which can process the data in one pass and give an approximate answer immediately. Prior algorithms mostly focus on data characteristics such as frequency [8–16], cardinality [17–19], and quantile [20–22]. Apart from the above characteristics, another important characteristic – ***persistence***, has received growing attention. Given an item e and a data stream with T non-overlapping and contiguous time windows, the persistence of e is defined as the number of time windows where e appears. For example, the number of days a person visits a website in one year is a kind of persistence, which indicates the person's preference for the website. Some prior algorithms [23] require the sizes of time windows to be equal. In this paper, to make our definition more general, we do not make any assumptions about the sizes of time windows.

Persistence is often related to different data stream applications, especially in network quality inspection [24] and anomalies detection. Persistence can be used to detect potentially malicious behaviors [25] about network security [26–30] or click fraud detection [31]. Frequency, which is used as an indicator by many detection systems, is more suitable for detecting anomalies behaving as heavy hitters. However, as shown in [25], some threats are trying to hide by spreading their communications over many time windows. For example, instead of communicating 2400 times in an hour, which can be easily detected by finding frequent items, some threats communicate only once every hour for 100 days, which cannot be detected by finding frequent items. To detect such threats, we should use persistence instead of frequency as an indicator.

In this paper, we address two problems with persistence. The first one is ***persistence estimation***. This problem requires the algorithm to report an approximate estimation of persistence for every item in the data stream. The second problem is ***finding persistent items***. Persistent items refer to items whose persistence is larger than a given threshold. This problem focuses on items with high persistence.

To the best of our knowledge, no prior work is designed for persistence estimation. Though there are many algorithms for frequency

[*] Department of Computer Science and Technology, Peking University, China
[†] The Key Laboratory of Hunan Province for Internet of Things and Information Security and College of Computer Science, Xiangtan University, Xiangtan, China
[‡] PCL Research Center of Networks and Communications, Pengcheng Laboratory
[§] Huawei Theory Lab, China
[¶] National Engineering Laboratory for Big Data Analysis Technology and Application (PKU), China
Corresponding author: Tong Yang {yangtongemail@gmail.com}.

Proceedings of the VLDB Endowment, Vol. 14, No. 2 ISSN 2150-8097.
doi:10.14778/3425879.3425884

estimation [9–11], they cannot be directly used for persistence estimation. Because of the different definitions of persistence and frequency, for a frequent item, its frequency could be much larger than its persistence. For example, for an item e_i appearing 100 times in only 1 time window, its frequency is 100, but its persistence is 1.

For finding persistent items, some prior algorithms [23] cannot work when the sizes of time windows are unequal. Two well-known algorithms, Small-Space [32] and PIE [33], can work without any assumptions about the sizes of time windows. Small-Space selects items by sampling and stores the sampled information in a hash table. The key advantage of Small-Space is that it can effectively reduce memory usage by sampling. However, it has three limitations. 1) It needs large memory usage because the hash table is a typical data structure that uses a large memory, and it has to keep all items sampled regardless of their persistence. 2) Its sampling rate needs to be low to keep small memory usage, which magnifies its error. 3) Hash collisions decrease its throughput. PIE finds persistent items by encoding and decoding item IDs. It reduces memory usage by storing the Raptor code [34] instead of ID. The key advantage of PIE is that the larger persistence an item has, the more Raptor codes PIE stores, and the larger probability the item is decoded successfully. However, it has the following two limitations. 1) Though it can reduce memory usage by encoding, it needs to store the Raptor code of every distinct item in every time window. Because most items in data streams are non-persistent items, PIE wastes much space to record the information of non-persistent items. 2) The information stored in PIE needs to be encoded through matrix multiplication, which is slow compared to the fast speed of data streams.

In this paper, we propose a new algorithm, On-Off sketch, to address problems of persistence estimation and finding persistent items. Our key idea is: utilizing the characteristic that the persistence of an item is increased periodically, we compress increments when multiple items are mapped to the same counter, which significantly reduces the error.

To better understand our key idea, we first show a strawman solution: using prior algorithms for frequency estimation (*e.g.*, Count-Min sketch [9]) to estimate persistence, with a Bloom filter [35] to remove duplicates. However, for this solution, in addition to the error incurred by the Bloom filter, there are still errors caused by hash collisions in the same time window. Because the Count-Min sketch uses hash functions to map items into a counter for each array, there could be many distinct items mapped to the same counter due to the space limitation. For example, if there are 5 distinct items mapped into the same counter and all occurred in the time window W_{cur}, the mapped counter is increased by 5, but the true persistence of each item is only increased by 1 in a time window. As a result, the persistence is highly overestimated because of hash collisions.

From the above example, we can find that regardless of the number of items mapped to the counter in a time window, we should only increment the counter by one due to the characteristic of the persistence. Therefore, for persistence estimation, instead of using the Bloom filter, we add a state field for each counter to show whether this counter has been incremented in the current time window. At the beginning of each time window, the state of every counter in the sketch is set to ***On***. Every time an item is mapped to a counter, only if the state of the counter is ***On***, we can increment the counter, and then turn the state to ***Off***. In this way, the On-Off sketch guarantees every counter is incremented at most once in a time window, eliminating the error from repeated items and significantly reducing the error from hash collisions. Further, the On-Off sketch does not need to query the Bloom filter to do additional memory accesses, which also improves its throughput.

For finding persistent items, based on the solution to persistence estimation, our key idea is to separate persistent and non-persistent items. We store the information on persistent items and protect them from hash collisions with other items. In this way, we further improve the accuracy of persistent items. In addition, the data structure used by our On-Off sketch is more time- and memory-efficient than prior algorithms. To achieve time-efficiency, the On-Off sketch guarantees that every item only needs to access one counter and one bucket. We also accelerate the On-Off sketch by enabling it to make use of the data parallelism of SIMD [36] instructions. To achieve memory-efficiency, the On-Off sketch does not use pointers and spends most space storing persistent items, which are often a small part of items in data streams.

Theoretical Result Advancement: For persistence estimation, our theoretical proofs show that the error of our On-Off sketch is always smaller than that of the Count-Min sketch with the same parameters. For finding persistent items, our theoretical proofs show that the space complexity of our On-Off sketch is much better than the state-of-the-art (PIE).

Experimental Result Advancement: For persistence estimation, our experimental results show that the On-Off sketch achieves around 6.17 times smaller error and 2.2 times higher throughput. For finding persistent items, our experimental results show that the On-Off sketch reduces the error up to 4 orders of magnitude and achieves 2.84 times higher throughput than prior algorithms. The source codes of the On-Off sketch and all other related algorithms are available at GitHub [37].

2 RELATED WORK

To the best of our knowledge, no prior work is designed for persistence estimation. Though there are prior works called persistent sketches [38] and persistent bloom filters [39], the meaning of persistence in these works is different. Therefore, we divide existing works into two categories: solutions for frequency estimation (Section 2.1), and solutions for finding persistent items (Section 2.2).

2.1 Frequency Estimation

Many solutions for frequency estimation use sketches. Typical works include Count-Min (CM) sketch [9] and Count sketch [11].

CM sketch: The CM sketch consists of d arrays, each consisting of l counters. The d arrays are associated with d pairwise independent hash functions. There are also two operations for this data structure: insertion and query. To insert item e_i, the CM sketch first calculates d hash functions and maps e_i to d counters, one counter in each array. We call these counters the d *mapped counters* for convenience. Then, the CM sketch increments each counter by 1. To query the frequency of item e_i, CM sketch maps it to d counters with the same hash functions and reports the *minimum mapped counter*, the minimum one of d counters, as the estimated frequency.

These solutions cannot be directly used on persistence estimation because they cannot avoid errors caused by repeated occurrences of an item within a time window. One way to deal with this problem is to use a Bloom filter [35]. In each time window, we build a new Bloom filter to remove duplicates and guarantee that every distinct item is inserted to the sketch at most once. However, there are still two limitations to this solution: 1) The Bloom filter brings additional error. 2) Errors caused by hash collisions in the same time window still exist.

2.2 Finding Persistent Items

Prior works on persistence often focus on finding persistent items. These works can report persistent items, but cannot estimate the persistence of all items. The two recent works which can work without any requirements for the sizes of time windows are Small-Space [32] and PIE [33].

Small-Space: The key idea of Small-Space is "sample and count". It maintains a hash table to track occurrences of some items. When an item arrives, if it is already tracked, Small-Space modifies the estimated persistence according to the time window field. Otherwise, Small-Space samples it according to its ID and the current time window. Sampling approaches used by Small-Space can effectively reduce time and space overhead. The shortcoming of Small-Space is the lack of space efficiency. The root reason is that it must keep all items sampled. In practice, most items are non-persistent items, and many non-persistent items will be sampled into Small-Space. As a result, it wastes much space to store many non-persistent items.

PIE: PIE proposes the Space-Time Bloom filter (STBF) and uses Raptor codes[34] to find persistent items. In each time window, PIE creates one STBF, which consists of several cells. For an item, PIE maps it to multiple cells in the STBF by hash functions, and in each of these mapped cells, PIE inserts the item by computing its fingerprint and Raptor code. To find persistent items, PIE searches all cells in all STBFs and decodes an ID if enough raptor codes are found. Items with larger persistence occur in more STBFs and have a larger probability of being decoded successfully. However, as the number of time windows increases, the memory needed by PIE is quite large. If there is not enough memory in STBFs, hash collisions may frequently occur, resulting in most cells being useless and the error increasing.

There are also some other algorithms for finding persistent items. In [25, 30], they use a hash table or a bitmap to record every item in the data stream to find persistent items. However, as pointed out by [32], these methods are memory consuming and impractical due to the large volume of data. Besides, LTC [23], which is proposed to find items that are both frequent and persistent, can also be used to find persistent items by adjusting the weight of frequency and persistence. However, due to the CLOCK technique it uses, LTC can only work when the sizes of time windows are equal, limiting its flexibility.

3 ALGORITHM DESIGN

In this section, before presenting details of our On-Off sketch, we first show a strawman solution and discuss its limitations (Section 3.1). Then we show the data structure and operations of our On-Off sketch for persistence estimation (Section 3.2) and finding persistent items (Section 3.3). Finally, we show the optimization of our On-Off sketch on finding persistent items (Section 3.4). For convenience, we list symbols frequently used in this paper and their meanings in Table 1.

Table 1: Symbols frequently used in this paper.

Notation	Meaning
S	a data stream
T	the number of time windows in S
N	the number of distinct items in S
e_i	the i_{th} distinct item in S
p_i	the persistence of item e_i
$\hat{p_i}$	the estimated persistence of item e_i
l	the number of counters/buckets
d	the number of arrays
w	the number of key-value pairs in a bucket
$h_i(.)$	the i_{th} hash function from item to $\{1, \ldots, l\}$
$C_i[j]$	the j_{th} counter in the i_{th} array
$B[i]$	the i_{th} bucket
$B[i]^{min}$	the smallest counter in $B[i]$

3.1 A Strawman Solution

Recall that the persistence of item e is the number of time windows where e appears. A strawman solution for persistence estimation is to use an existing sketch for frequency estimation to estimate persistence, and use an additional data structure to remove duplicates. We use the strawman solution proposed by PIE [33]: a Count-Min (CM) sketch [9] on frequency estimation and a Bloom filter [35] on duplicates removing. We have presented the details of the CM sketch in Section 2.1, and we only show the details of the Bloom filter here.

Bloom filter: A Bloom filter is used for the set membership query. It is an array of m bits, all of which are initialized to 0. The array is associated with z independent hash functions, each of which maps the item uniformly into one bit of the array. To insert an item e, the Bloom filter uses z hash functions to map e into z bits in the array and sets all these z bits to 1. To query an item e, the Bloom filter maps e to z bits and reports true only if all the z bits are 1.

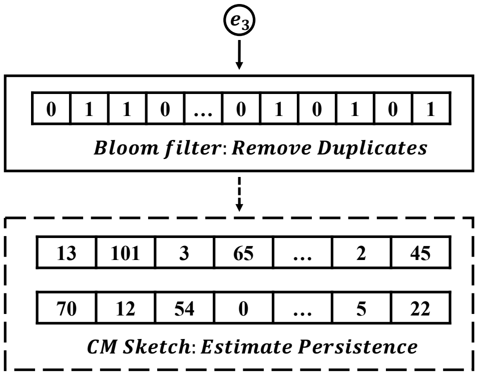


Figure 1: Insertion of the strawman solution.

Here we show how the strawman solution works, shown in Figure 1. To insert item e_i, we first query the Bloom filter. If the Bloom filter reports true, indicating e_i has occurred in this time window, we do nothing. Otherwise, we insert e_i into both the Bloom filter and CM sketch. At the end of each time window, we clear the Bloom filter by setting all bits to 0. To estimate persistence, we query CM sketch and report the minimum mapped counter as the estimated persistence.

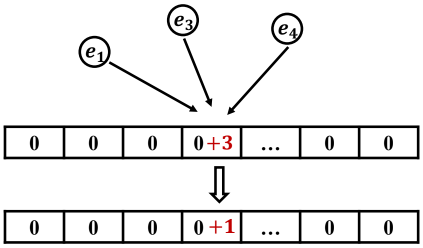


Figure 2: Limitations of the strawman solution.

Limitations: There are two main limitations to this strawman solution. First, the Bloom filter will incur additional errors. If an incoming item has not appeared, the Bloom filter may also report true, which is called *false positive*. Second, hash collisions in the CM sketch make the persistence significantly overestimated. As shown in Figure 2, if three different items are mapped to the same counter at the same time window, the mapped counter will increase by 3. However, we should note that the persistence of each item only increases by one at this time window. To address these limitations, we aim to avoid using the Bloom filter and guarantee that a counter in the sketch is incremented at most once in each time window.

3.2 Persistence Estimation

To address the limitations of the strawman solution, our On-Off sketch adds a state field for each counter to indicate whether it has been incremented or not in the current time window. Next, we illustrate the On-Off sketch for persistence estimation in detail.

Data Structure (Figure 3): The On-Off sketch consists of d arrays, each of which consists of l counters. Let $C_i[j]$ be the j^{th} counter in the i^{th} array. Each counter has a state field with two states: *On* and *Off*. Initially, all state fields are *On*, and all counters are zero. The d arrays are associated with d pairwise independent hash functions $h_1(.)...h_d(.)$, respectively. For $1 \leqslant i \leqslant d$, $h_i : \{1, \ldots, N\} \rightarrow \{1, \ldots, l\}$.

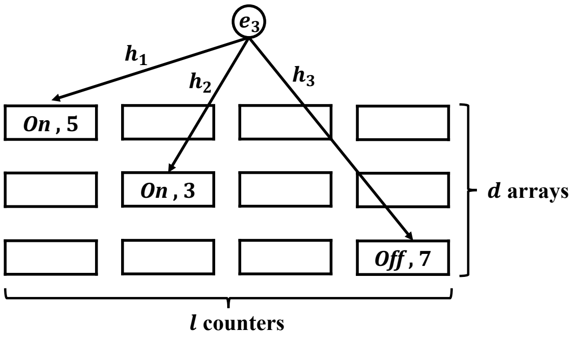


Figure 3: Data Structure of persistence estimation.

Insertion: To insert item e_i, the On-Off sketch calculates d hash functions and maps e_i to d counters $C_j[h_j(e_i)]$, one counter in each array. For each of these d mapped counters, there are two cases:

Case 1: The counter's state is *On*, indicating that this counter has not been accessed in the current time window yet. The On-Off sketch increments the counter by one and turns the state to *Off*.

Case 2: The counter's state is *Off*, indicating that this counter has been accessed in the current time window. The On-Off sketch does not change the counter.

Examples of Insertion (Figure 4): Let $d = 3$. To insert item e_3, the On-Off sketch calculates 3 hash functions and maps e_3 to one counter in each array. In the mapped counter of the first array, the state field is *On*, so this is **Case 1**. In this case, the On-Off sketch will increment the counter by 1, *i.e.*, from 5 to 6, and turns the state to *Off*. In the mapped counter of the second array, the state field is *On*, so the On-Off sketch increments the counter by 1, *i.e.*, from 3 to 4, and turns the state to *Off*. In the mapped counter of the third array, the state field is *Off*, which is **Case 2**. In this case, the On-Off sketch does nothing.

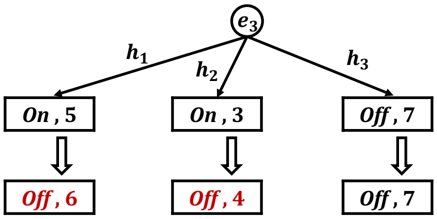


Figure 4: Insertion examples of persistence estimation.

Query: To query the persistence of item e_i, the On-Off sketch first calculates the d hash functions and gets d mapped counters. Then it reports the minimum mapped counter among these d counters as e_i's persistence. In other words, let $\hat{p_i}$ be the estimated persistence, and we have $\hat{p_i} = min_{1 \leqslant j \leqslant d}(C_j[h_j(e_i)])$.

Periodical Emptying: At the end of each time window, the On-Off sketch should set all state fields to *On*.

Analysis: By adding a state field for each counter, we can guarantee every counter will be incremented at most once in each time window, regardless of the number of items mapped to it. Our On-Off sketch reduces the overestimation error by eliminating the error incurred by repeated occurrences of the same item in one time window and significantly reducing the error from hash collisions. In addition, the On-Off sketch only has one-sided errors, *i.e.*, it only overestimates the persistence, since the state field does not incur false positives like the Bloom filter.

3.3 Finding Persistent Items

On persistence estimation, we can find that the On-Off sketch does not store the IDs of items, indicating the data structure of the On-Off sketch in Section 3.2 cannot be directly used to find persistent items. Therefore, we change the data structure of our On-Off sketch to find persistent items. The key technique of our On-Off sketch in finding persistent items is that we distinguish items by persistence and only store the IDs of persistent items.

Data Structure (Figure 5): There are two parts in the On-Off sketch for finding persistent items. The first part is an array similar to the data structure shown in Section 3.2, which is used to record the estimated persistence of non-persistent items. Specifically, it has l counters, where the i^{th} counter is denoted as $C_1[i]$. This

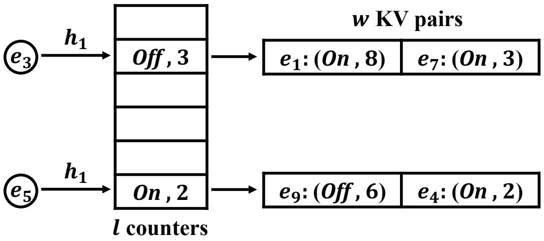


Figure 5: Data Structure of finding persistent items.

array is associated with a hash function $h_1(.)$, mapping each item into one counter in the array. The second part is used to record persistent items. It consists of an array of l buckets, where the i^{th} bucket is denoted as $B[i]$. Each bucket corresponds to a counter, *i.e.*, $B[i]$ corresponds to $C_1[i]$. There are w key-value (KV) pairs in each bucket, where the key is the ID of the item, and the value is the corresponding counter. We use $B[i][e_j]$ to represent the corresponding counter of e_j if e_j is stored in $B[i]$. There is also a state field for each counter with two states: *On* and *Off*. Initially, all state fields are *On*, and all counters are zero.

Increment of counters: No matter it is the counter in the first or second part, we increment the counters in the same way as shown in Section 3.2: if the counter's state is *On*, the On-Off sketch increments the counter by one and turns the state to *Off*. Otherwise, it does nothing.

Insertion of the On-Off sketch: To insert e_i, the On-Off sketch first checks whether e_i has been recorded in bucket $B[h_1(e_i)]$. If e_i has been recorded in $B[h_1(e_i)]$, the On-Off sketch increments the corresponding counter in $B[h_1(e_i)][e_i]$. Otherwise, it increments counter $C_1[h_1(e_i)]$. After insertion, the On-Off sketch compares $C_1[h_1(e_i)]$ with the smallest counter in $B[h_1(e_i)]$ ($B[h_1(e_i)]^{min}$) in order to determine whether e_i is persistent enough to store in the bucket $B[h_1(e_i)]$. If $C_1[h_1(e_i)] > B[h_1(e_i)]^{min}$, indicating that the estimated persistence of e_i is larger, e_i should be stored in bucket $B[h_1(e_i)]$, while $C_1[h_1(e_i)]$ should be replaced with the value of the smallest counter in $B[h_1(e_i)]$. Therefore, the On-Off sketch sets the key of $B[h_1(e_i)]^{min}$ to e_i, swaps $C_1[h_1(e_i)]$ and $B[h_1(e_i)]^{min}$, and swaps their states. If $C_1[h_1(e_i)] \leqslant B[h_1(e_i)]^{min}$, indicating e_i is not persistent enough, the On-Off sketch does nothing.

Examples of Insertion (Figure 6): Let $w = 2$. To insert item e_3, the On-Off sketch first calculates one hash function and maps e_3 to a counter and the corresponding bucket. Because e_3 is not stored in the bucket, the On-Off sketch inserts e_3 into the counter. The state field of the counter is *Off*, so nothing is changed. Because 3 is not larger than the smallest counter (3) in the bucket, the On-Off sketch does not swap. After that, to insert item e_1, we find e_1 is stored in the bucket, so the On-Off sketch inserts e_1 into the corresponding counter, and the counter is updated to (*Off*, 9). In another example, we insert item e_5. We find that e_5 is not stored in the bucket, so the On-Off sketch inserts e_5 into the counter, and the counter is updated to (*Off*, 3). Because 3 is larger than the smallest counter (2) in the bucket, the On-Off sketch sets the key to e_5, and swaps (*Off*, 3) and (*On*, 2). After that, to insert item e_2, because e_2 is not stored in the bucket, the On-Off sketch inserts e_2 into the counter, and the counter is updated to (*Off*, 3). Because 3 is not larger than the smallest counter (3), the On-Off sketch does not swap.

Query: To report the persistent items whose persistence is above a predefined threshold, we traverse all buckets and report the IDs of items e_i whose corresponding counter $B[h_1(e_i)][e_i]$ is larger than the given threshold.

Periodical Emptying: At the end of each time window, the On-Off sketch should set all state fields to *On*.

Analysis: Compared with the On-Off sketch for persistence estimation, the On-Off sketch for finding persistent items only uses one array of counters and uses an additional bucket for each counter. By reducing the number of arrays, we can improve the throughput. By using an additional bucket for each counter, we can separate persistent items from non-persistent items and only record the IDs of persistent items. If an item is persistent enough, it will not be the smallest in the bucket, so it will be protected from replacements and hash collisions.

3.4 Optimization

In this section, we optimize the On-Off sketch for finding persistent items by exploiting SIMD (Single Instruction Multiple Data) instructions [36]. SIMD instructions achieve data parallelism by vectorization, which can effectively accelerate sequential access operations [40]. Prior works on finding persistent items cannot take advantage of SIMD instructions, because there are no sequential access operations in their algorithms. For example, Small-Space [32] and PIE [33] need several memory accesses for each insertion, and for each memory access, they only read or write one item.

Then we show how SIMD instructions can accelerate the On-Off sketch. As shown in Section 3.3, to insert an item e_i, the On-Off sketch first checks whether it is stored in $B[h_1(e_i)]$. Specifically, we sequentially compare the ID of e_i with IDs stored in $B[h_1(e_i)]$. With SIMD instructions, our On-Off sketch can match IDs of items in parallel. If e_i is not stored in $B[h_1(e_i)]$, the On-Off sketch has to find the smallest counter and get its position. Such an operation cannot directly take advantage of the acceleration of SIMD instructions. To address this issue, we use an operation equivalent to finding the smallest value to make our algorithm more SIMD-friendly. We can find that, for an incoming item e_i, if the replacement happens, the state of $C_1[h_1(e_i)]$ is *On* and $C_1[h_1(e_i)] = B[h_1(e_i)]^{min}$, which is proved in Theorem 4.4. As a result, the On-Off sketch can decide whether it should replace by checking if $C_1[h_1(e_i)]$'s state is *On* and if there is a counter equal to $C_1[h_1(e_i)]$ in bucket $B[h_1(e_i)]$. In our implementation, we sequentially compare $C_1[h_1(e_i)]$ with counters stored in $B[h_1(e_i)]$, which can also be accelerated by SIMD instructions. Suppose the length of ID is 4 bytes, and each bucket stores 8 items. The pseudo-code of the insertion implemented by AVX2 SIMD instructions is provided in Algorithm 1.

From line 1 to line 3, the On-Off sketch searches whether e_i has been stored in buckets. The TZCNT [41] in line 5 is an instruction that counts the number of trailing least significant zero bits. It can return the position of the matching item according to the result. If e_i has been in the bucket, as shown in line 6, the On-Off sketch inserts it into the corresponding counter. Otherwise, if $C_1[h_1(e_i)]$'s state is *On*, as shown from line 9 to line 11, the On-Off sketch searches whether there is a counter equal to $C_1[h_1(e_i)]$. If there is, as shown from line 13 to line 15, the On-Off sketch modifies the corresponding KV pair in $B[h_1(e_i)]$. Otherwise, as shown in line 17, it modifies $C_1[h_1(e_i)]$.

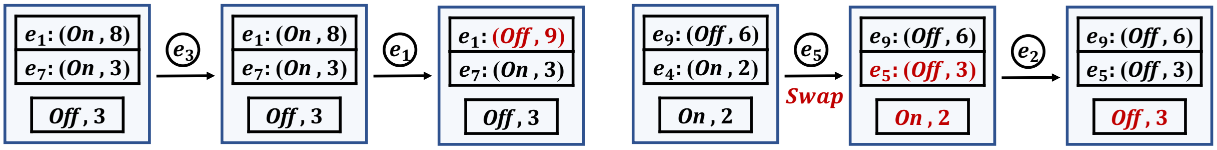


Figure 6: Insertion examples of finding persistent items.

Algorithm 1: Insertion implemented by SIMD instructions.

```
Input: An item $e_i$, the array of IDs $keys$, the array of
       counters $values$
__m256i item = _mm256_set1_epi32($e_i$);
__m256i cmp = _mm256_cmpeq_epi32(item, $keys$);
int match = _mm256_movemask_ps(cmp);
if $match \neq 0$ then
    $values$[TZCNT(match)].Insert();
    return;
if the state of $C_1[h_1(e_i)]$ is On then
    item = _mm256_set1_epi32($C_1[h_1(e_i)]$);
    cmp = _mm256_cmpeq_epi32(item, $values$);
    match = _mm256_movemask_ps(cmp);
    if $match \neq 0$ then
        int pos = TZCNT(match);
        $keys$[pos] = $e_i$;
        $values$[pos].Insert();
    else
        $C_1[h_1(e_i)]$.Insert();
    end
return;
```

3.5 Sliding Window

In many applications about data analysis, people are often more concerned about recent items. To adapt our On-Off sketch to the sliding window model, we can use the *Sliding Sketch* [42], a framework that can be applied to most of the existing sketches. As shown in the paper, Sliding Sketch [42] has been applied to the CM sketch and HeavyKeeper [43] (an algorithm for finding heavy hitters) and achieves good performance. We can also apply the Sliding Sketch to the On-Off sketch due to the high similarity between our data structure and that of the CM sketch and HeavyKeeper. The key idea of Sliding Sketch is dividing the sketch into many "time zones" and deleting out-dated information through scanning. The error bound and one-sided error of our On-Off sketch still hold after using the framework. Since there is a suitable framework available, we do not design new methods for sliding windows.

4 MATHEMATICAL ANALYSIS

In this section, we provide a performance analysis for our On-Off sketch on persistence estimation (Section 4.1) and finding persistent items (Section 4.2).

4.1 Persistence Estimation

We first show the error bound of our On-Off sketch (Section 4.1.1). Then we compare the On-Off sketch with prior works on frequency estimation (Section 4.1.2). Let T be the number of time windows in the data stream, p_i be the persistence of item e_i, $\|p\|_1 = \sum_{i=1}^{N} p_i$, and $\hat{p}_i$ be the estimated persistence reported by our On-Off sketch.

4.1.1 *Error Bound.*

Theorem 4.1.

$$p_i \leqslant \hat{p}_i \leqslant T \tag{1}$$

Proof. According to the algorithm, every time e_i arrives in time window W_{cur}, the mapped counter will be incremented by one in W_{cur}. Therefore, $\forall j, C_j[h_j(e_i)] \geqslant p_i$. Thus we have

$$\hat{p}_i = min_{1 \leqslant j \leqslant d}(C_j[h_j(e_i)]) \geqslant p_i$$

Further, $C_j[h_j(e_i)]$ will be incremented at most once in a time window. Therefore, $\forall j, C_j[h_j(e_i)] \leqslant T$. Thus we have

$$\hat{p}_i = min_{1 \leqslant j \leqslant d}(C_j[h_j(e_i)]) \leqslant T.$$

The estimated persistence $\hat{p}_i$ of item e_i has an upper bound T and a lower bound p_i. □

Theorem 4.2. *Let $l = e/\epsilon$ and $d = \ln(1/\delta)$. We have*

$$\mathbb{P}\left(\hat{p}_i \leqslant p_i + \epsilon\|p\|_1\right) \geqslant 1 - \delta \tag{2}$$

Proof. Let $\Delta_j p_i = C_j[h_j(e_i)] - p_i$, E_t be the set of all distinct items appearing in time window t, P_i be the set of time windows where item e_i occurs, $\overline{P_i} = \{1, ..., T\} - P_i$, and $I_{i,j,t}$ be 1 if $\exists e_k \in E_t, i \neq k \wedge h_j(e_i) = h_j(e_k)$ and 0 otherwise. We have

$$\mathbb{E}\left[\Delta_j p_i\right] = \sum_{t \in \overline{P_i}} \mathbb{E}\left[I_{i,t}\right] = \sum_{t \in \overline{P_i}} \left[1 - \left(1 - \frac{1}{l}\right)^{|E_t|}\right]$$

Because each distinct item in a time window makes contribution to its persistence, we have

$$\|p\|_1 = \sum_{i=1}^{N} p_i = \sum_{t=1}^{T} |E_t| = \|E\|_1$$

Therefore, when $l = e/\epsilon$, we have

$$\mathbb{E}\left[\Delta_j p_i\right] \leqslant \frac{\sum_{t \in \overline{P_i}} |E_t|}{l} \leqslant \frac{\|p\|_1}{l} = \frac{\epsilon\,\|p\|_1}{e}$$

By the Markov inequality,

$$\begin{aligned} \mathbb{P}\left(\hat{p}_i \leqslant p_i + \epsilon\,\|p\|_1\right) &\geqslant 1 - \mathbb{P}\left(\forall_j \Delta_j p_i > e\mathbb{E}\left[\Delta_j p_i\right]\right) \\ &\geqslant 1 - e^{-d} \\ &= 1 - \delta \end{aligned}$$

Therefore, the On-Off sketch can address the persistence estimation problem with $O((1/\epsilon)\ln(1/\delta))$ space complexity and $O(\ln(1/\delta))$ time complexity. □

4.1.2 *Comparison with Related Work.*

As shown in Section 2, there is no solution that can be directly used on persistence estimation. Therefore, we compare our On-Off sketch with the strawman solution, *i.e.*, a CM sketch with a Bloom filter. According to Theorem 4.1, we can find that our On-Off sketch has a much better upper bound than that of the strawman solution. The upper bound of the estimated persistence in CM sketch is $\|p\|_1$, while it is T in the On-Off sketch, which is much lower. Then we show that, ignoring the error brought by the Bloom filter, the error of our On-Off sketch is always smaller than that of the strawman solution with same parameters.

Theorem 4.3. *Let $\hat{p_i}^{CM}$ be the estimated persistence of the strawman solution. With same l and d and ignoring the error brought by the Bloom filter,*

$$\hat{p_i} - p_i \leqslant \hat{p_i}^{CM} - p_i \tag{3}$$

Proof. Let $P_{i,j,k}$ be the set of time windows where item e_k occurs if $i \neq k \wedge h_j(e_i) = h_j(e_k)$ and null set otherwise. For the On-Off sketch, the error of estimated persistence can be only caused by hash collisions that happen in time window where e_i does not occur. Therefore, $\Delta_j p_i = \left|\bigcup\limits_k P_{i,j,k} - P_i\right|$. For the strawman solution, ignoring the error brought by the Bloom filter, $\Delta_j^{CM} p_i = \sum\limits_k \left|P_{i,j,k}\right|$. Because $\left|\bigcup\limits_k P_{i,j,k} - P_i\right| \leqslant \sum\limits_k \left|P_{i,j,k}\right|$, we have

$$\begin{aligned}\hat{p_i} - p_i &= min_{1\leqslant j\leqslant d}\left(\Delta_j p_i\right)\\ &\leqslant min_{1\leqslant j\leqslant d}\left(\Delta_j^{CM} p_i\right)\\ &= \hat{p_i}^{CM} - p_i\end{aligned}$$

□

4.2 Finding Persistent Items

On finding persistent items, we let $\hat{p_i} = B[h_1(e_i)][e_i]$ for item e_i which is stored in buckets, and $\hat{p_i} = C_1[h(e_i)]$ for other items. To prove the error bound of the On-Off sketch, we first show the necessary and sufficient condition for replacements.

Theorem 4.4. *When an item e_i comes, the replacement happens if and only if $e_i \notin B[h_1(e_i)] \wedge$ the state of $C_1[h_1(e_i)]$ is On $\wedge$ $C_1[h_1(e_i)] = B[h_1(e_i)]^{min}$.*

Proof. As shown in the algorithm, a necessary condition for the replacement is $e_i \notin B[h_1(e_i)]$. Because $C_1[h_1(e_i)]$ is incremented at most one in a time window, $C_1[h_1(e_i)]$ must be equal to $B[h_1(e_i)]^{min}$ before $C_1[h_1(e_i)] > B[h_1(e_i)]^{min}$. Only when the state of $C_1[h_1(e_i)]$ is *On*, the On-Off sketch can increment $C_1[h_1(e_i)]$ by 1. In addition, the replacement must happen, if $e_i \notin B[h_1(e_i)] \wedge$ the state of $C_1[h_1(e_i)]$ is *On* $\wedge$ $C_1[h_1(e_i)] = B[h_1(e_i)]^{min}$, according to the algorithm. Therefore, it is the necessary and sufficient condition for replacements. □

Then we show the error bound (Section 4.2.1) and recall rate (Section 4.2.2) of our On-Off sketch. Finally we compare the On-Off sketch with the state-of-the-art – PIE [33] (Section 4.2.3).

4.2.1 *Error Bound.*

Theorem 4.5.

$$\hat{p_i} - C_1[h_1(e_i)] \leqslant p_i \leqslant \hat{p_i} \leqslant T \tag{4}$$

Proof. According to Theorem 4.1, the counter is incremented at most once in a time window, so $\hat{p_i} \leqslant T$. In the initial state, the equation is satisfied. Next, we need to prove that the equation is satisfied at every insertion. When an item e_i comes, there are 3 cases.

Case 1: e_i is in $B[h_1(e_i)]$. This insertion does not affect other items. For e_i, if it has occurred in T_{cur}, p_i and $\hat{p_i}$ do not change, the equation still holds. Otherwise,

$$\hat{p_i} + 1 - C_1[h_1(e_i)] \leqslant p_i + 1 \leqslant \hat{p_i} + 1.$$

Case 2: e_i is not in $B[h_1(e_i)]$ and the replacement does not happen. This insertion only affects the items e_j which are not in buckets and $h_1(e_j) = h_1(e_i)$. If the state of $C_1[h_1(e_i)]$ is *Off*, $C_1[h_1(e_i)]$ does not change, so the equation holds. Otherwise, for e_i,

$$0 \leqslant p_i + 1 \leqslant \hat{p_i} + 1,$$

while for other affected items e_j

$$0 \leqslant p_i \leqslant \hat{p_i} + 1.$$

The equation still holds.

Case 3: e_i is not in $B[h_1(e_i)]$ and the replacement happens. According to Theorem 4.4, $C_1[h_1(e_i)] = B[h_1(e_i)]^{min}$ at that time, so the estimated persistence of other items does not change. For e_i,

$$\hat{p_i} + 1 - C_1[h_1(e_i)] \leqslant p_i + 1 \leqslant \hat{p_i} + 1,$$

Finally, we can find that the equation holds for all cases. □

Theorem 4.6. *Let $l = 2/\epsilon$. We have*

$$\mathbb{P}\left(\hat{p_i} \leqslant p_i + \frac{\epsilon\|p\|_1}{w+1}\right) \geqslant \frac{1}{2} \tag{5}$$

Proof. According to Theorem 4.5, the error of the estimated persistence $\hat{p_i}$ only comes from insertions to $C_1[h_1(e_i)]$ when e_i is not stored in buckets. Replacements and insertions to buckets do not bring any error. Let X_i be the sum of the persistence of all items e_j, whose $h_1(e_i) = h_1(e_j)$ and $i \neq j$, *i.e.*,

$$X_i = \sum_{h_1(e_i)=h_1(e_j)\wedge j\neq i} p_j$$

Because all counters in $B[h_1(e_i)]$ are not smaller than $C_1[h_1(e_i)]$, we have

$$\hat{p_i} - p_i \leqslant \frac{X_i}{w+1}$$

Therefore,

$$\begin{aligned}\mathbb{P}\left(\hat{p_i} \leqslant p_i + \frac{\epsilon\|p\|_1}{w+1}\right) &\geqslant \mathbb{P}\left(\frac{X_i}{w+1} \leqslant \frac{2(\|p\|_1 - p_i)}{(w+1)\cdot l}\right)\\ &= 1 - \mathbb{P}\left(\frac{X_i}{w+1} > 2\cdot\mathbb{E}\left[\frac{X_i}{w+1}\right]\right)\\ &\geqslant \frac{1}{2}\end{aligned}$$

□

4.2.2 ***Recall Rate.***

THEOREM 4.7. *If $p_i > C_1[h_1(e_i)]$, e_i is guaranteed to be stored in buckets.*

PROOF. According to Theorem 4.5, if p_i is not stored in buckets,

$$0 \leqslant p_i \leqslant \hat{p_i} = C_1[h_1(e_i)]$$

Therefore, if $p_i > C_1[h_1(e_i)]$, e_i must be stored in buckets. □

THEOREM 4.8. *Let $\mathbb{P}_i$ be the probability that e_i is stored in buckets.*

$$\mathbb{P}_i \geqslant 1 - \frac{||p||_1 - p_i}{w \cdot l \cdot p_i}. \tag{6}$$

PROOF. Because all counters in $B[h_1(e_i)]$ are not smaller than $C_1[h_1(e_i)]$, we have

$$C_1[h_1(e_i)] \leqslant \frac{X_i + p_i}{w+1}$$

Because $p_i > C_1[h_1(e_i)]$ is a sufficient condition of that e_i is stored in buckets,

$$\begin{aligned} \mathbb{P}_i &\geqslant \mathbb{P}\left(p_i > C_1[h_1(e_i)]\right) \\ &\geqslant 1 - \mathbb{P}\left(X_i + p_i \geqslant (w+1) \cdot p_i\right) \\ &\geqslant 1 - \frac{\mathbb{E}[X_i]}{w \cdot p_i} \\ &= 1 - \frac{||p||_1 - p_i}{w \cdot l \cdot p_i} \end{aligned}$$

□

4.2.3 ***Comparison with Related Work.***

In this section, we compare our On-Off sketch with the state-of-the-art – PIE [33]. Because PIE does not show how much space it needs in its paper, we first calculate its space complexity. In each time window, PIE builds a Space-Time Bloom Filter (STBF). Let E_t be the set of all distinct items appearing in time window t. The space needed by STBF in time window t is $O(E_t)$. Therefore, the space complexity of PIE is

$$O\left(\sum_{t=1}^{T} |E_t|\right) = O(||p||_1)$$

According to Theorem 4.6 and Theorem 4.8, if $l = 2/\epsilon$ and $w = 1$,

$$\mathbb{P}\left(\hat{p_i} \leqslant p_i + \frac{\epsilon||p||_1}{2}\right) \geqslant \frac{1}{2}$$

and for item e_i whose persistence is larger than $\epsilon||p||_1$,

$$\mathbb{P}_i \geqslant \frac{1+\epsilon}{2}$$

Therefore, the space complexity of our On-Off sketch is $O(1/\epsilon)$, which is much smaller than that of the PIE.

Parameter Estimation: We note that for all algorithms [32, 33] on persistence, their error bounds or space complexities are related to $||p||_1$, so it is important to get the estimation of $||p||_1$ in the data stream. Let $\widetilde{||p||_1}$ be the estimation of $||p||_1$. We provide three methods to estimate $||p||_1$. First, we can use the number of items in the data stream to be $\widetilde{||p||_1}$. In this way, the error bound still holds because $\widetilde{||p||_1} \geqslant ||p||_1$. The estimation of the number of items is also easy to get, but such approximation will make the error bound loose. Second, we can get an approximate estimation of $||p||_1$ according to the number of distinct items in past time windows. $||p||_1 = \sum_{t=1}^{T} |E_t|$, where $|E_t|$ is the number of distinct items in time window t. After getting the estimation of the number of distinct items in a time window, we can multiply it by T to be $\widetilde{||p||_1}$. Third, if we can go through the data stream, we can use sketches on cardinality estimation, such as FM sketch [19] and Hyperloglog [17], to estimate the number of distinct items in each time window with small memory. By summing up the result in each time window, we can get $\widetilde{||p||_1}$. In this way, the error of $\widetilde{||p||_1}$ is related to the sketch we use. If we use the Hyperloglog with m estimator, the standard error is around $1.04/\sqrt{m}$.

5 EXPERIMENTAL RESULTS

In this section, we provide experimental results of our On-Off sketch. First, we describe the experimental setup (Section 5.1). Then we show the performance of our On-Off sketch on persistence estimation (Section 5.2) and finding persistent items (Section 5.3).

5.1 Experimental Setup

Datasets: For each dataset, we divide it into 1600 time windows, *i.e.*, $T = 1600$.

1) Synthetic Datasets: We generate a synthetic dataset that follows the Zipf [44] distribution using Web Polygraph [45], an open-source performance testing tool. The length of each item ID is 4 bytes, and the skewness is 1.5.

2) Data Center Dataset: The Data center dataset [46] contains traces collected from the data centers studied in [47]. Each item (4 bytes) represents the ID of the trace.

3) Network Dataset: The network dataset contains users' posting history on the stack exchange website [48]. Each item (4 bytes) represents the ID of each user.

4) IP Trace Dataset: The IP Trace Dataset contains streams of anonymized IP traces collected in 2016 by CAIDA [49]. Each item contains a source IP address (4 bytes) and a destination IP address (4 bytes), 8 bytes in total.

Implementation: We have implemented our On-Off sketch in C++. The hash functions are implemented using 32-bit Bob Hash (obtained from the open-source website [50]) with different initial seeds. All algorithms we implemented are single-thread.

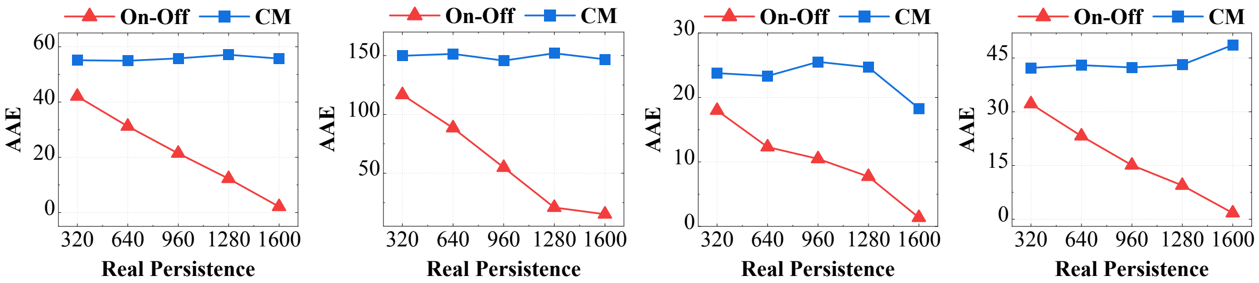
Computation Platform: We conducted all experiments on a machine with one 6-core processors (6 threads, Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz) and 16 GB DRAM memory. Each processor has three levels of cache memory: one 32KB L1 data cache and one 32KB L1 instruction cache for each core, one 256KB L2 cache for each core, and one 9MB L3 cache shared by all cores.

Metrics:

1) Average Absolute Error (AAE): $\frac{1}{|\Psi|}\sum_{e_i \in \Psi} |p_i - \hat{p_i}|$, where p_i is the real persistence of item e_i, $\hat{p_i}$ is its estimated persistence, and Ψ is the query set. For persistence estimation, the query set is all distinct items in the data stream. For finding persistent items, the query set is the reported items whose persistence is higher than a predefined threshold in the data stream.

2) False Positive Rate (FPR): Ratio of the number of non-persistent items reported to the number of non-persistent items.

3) False Negative Rate (FNR): Ratio of the number of persistent items that are not reported to the number of persistent items.



(a) Synthetic (b) Data center (c) Network (d) IP trace

Figure 7: AAE on persistence estimation.

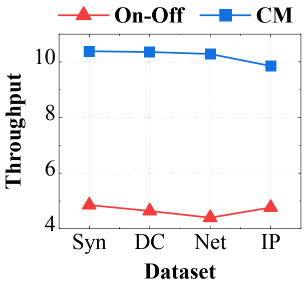


Figure 8: Throughput on persistence estimation.

4) F1 Score: Let CR be the recall rate, the ratio of the number of persistent items that are reported to the number of persistent items, and PR be the precision rate, the ratio of the number of persistent items that are reported to the number of reported items. F1 Score is $(2 \cdot CR \cdot PR)/(CR + PR)$.

5) Throughput: Million operations per second (Mops). All the experiments about throughput are repeated 5 times, and the median throughput is reported.

5.2 Persistence Estimation

5.2.1 *Comparison with Related Work.*

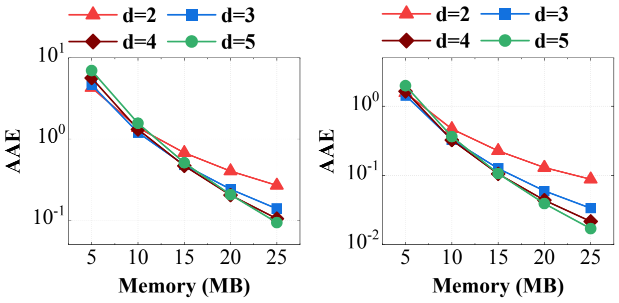
For persistence estimation, we compare our On-Off sketch with the strawman solution, *i.e.*, a CM sketch [9] with a Bloom filter [35], which is proposed by PIE [33]. For the On-Off sketch, we set $d = 2$. For CM sketch, we also set $d = 2$, while for the Bloom filter, we set $z = 4$ to balance the accuracy and throughput. We fix the memory size to make comparisons.

AAE (Figure 7(a)-7(d)): As proved in Theorem 4.3, the error of our On-Off sketch is always smaller than that of the strawman solution. Our results show that, if the real persistence is lower than 320, the AAE of our On-Off sketch is around 1.31 times lower than that of the strawman solution. If the real persistence is larger than 1280, the AAE of our On-Off sketch is around 19.3 times lower than that of the strawman solution.

Throughput (Figure 8): Our results show that, on average, the throughput of our On-Off sketch is around 2.20 times higher than that of the strawman solution. It is because that the strawman solution has to access memory more times.

5.2.2 *Parameter Setting.*

According to Section 3.2, we have two parameters in persistence estimation: l and d. To evaluate the influence of parameter setting, we fix the memory usage of the On-Off sketch and vary d.



(a) Synthetic (b) Network

Figure 9: AAE of the On-Off sketch.

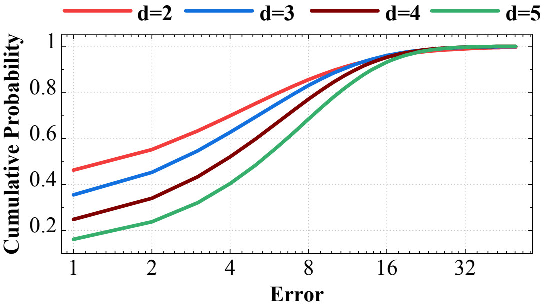
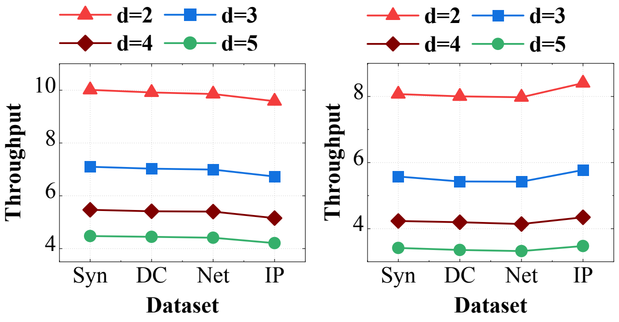


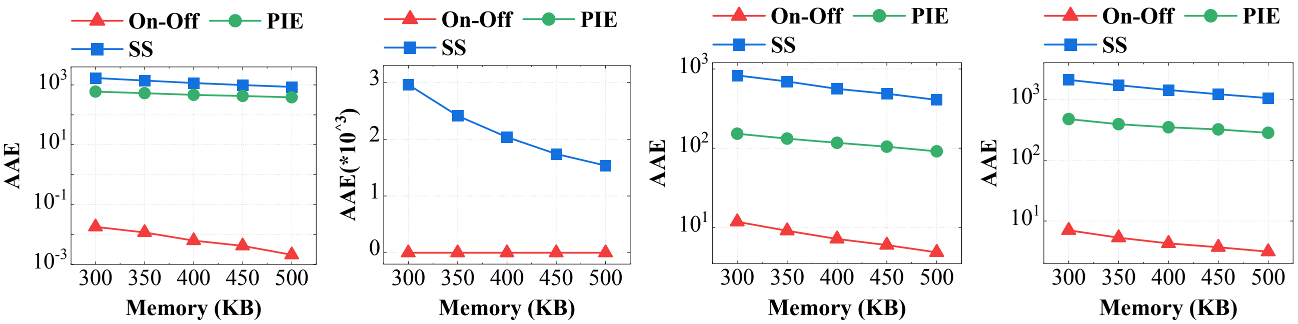
Figure 10: CDF of the error on persistence estimation.



(a) 5MB (b) 25MB

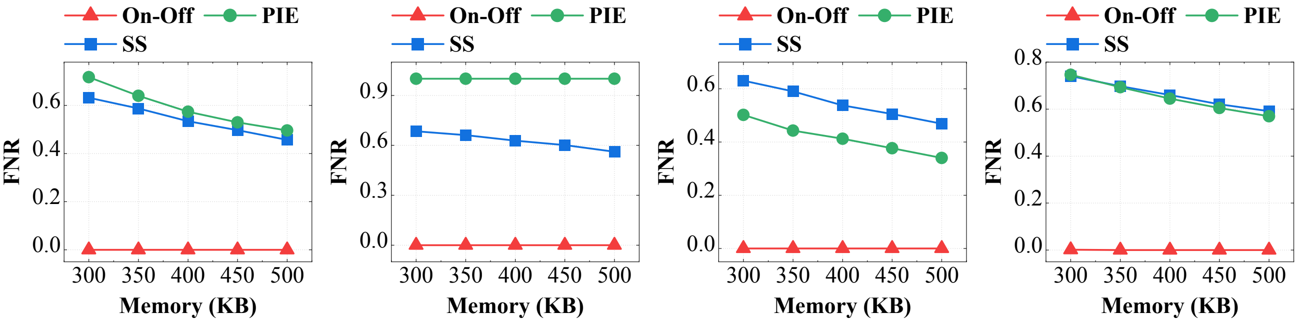
Figure 11: Throughput of the On-Off sketch.

AAE (Figure 9(a)-9(b)): In the synthetic dataset, when the memory is 5MB, the optimal d is 2, whose AAE is around 8%, 31%, and 62% lower than that of the $d = 3$, $d = 4$, and $d = 5$, respectively. When



(a) Synthetic (b) Data center (c) Network (d) IP trace

Figure 12: AAE on finding persistent items.



(a) Synthetic (b) Data center (c) Network (d) IP trace

Figure 13: FNR on finding persistent items.

the memory is 25MB, the optimal d is 5, whose AAE is around 187%, 48%, and 11% lower than that of the $d = 2$, $d = 3$, and $d = 4$, respectively. In the network dataset, when the memory is 5MB, the optimal d is 3, whose AAE is around 7%, 24%, and 38% lower than that of the $d = 2$, $d = 4$, and $d = 5$, respectively. When the memory is 25MB, the optimal d is 5, whose AAE is around 415%, 97%, and 27% lower than that of the $d = 2$, $d = 3$, and $d = 4$, respectively.

CDF of the error (Figure 10): We use the synthetic dataset, set the memory to 5MB, and repeat experiments 10 times to show the CDF of error for each item. We can find that, with a lower d, there is a higher probability that the errors of items are low. For example, for $d = 2$, the probability that the error is less than 2 is about 46.1%, while for $d = 5$, the probability is about 16.1%. In addition, with a lower d, the error of the worst case is also higher. The probability that the error is higher than 50 for $d = 2$ is 0.41%, while for $d = 5$, the probability is about 0.03%.

Throughput (Figure 11(a)-11(b)): Our results show that, the insertion throughput when the memory is 5MB is about 26% higher than the insertion throughput when the memory is 25MB. In addition, the insertion throughput of $d = 2$ is around 44%, 88%, and 132% higher than that of the $d = 3$, $d = 4$, and $d = 5$, respectively.

Analysis: 1) For AAE, the optimal d increases as the memory increases. With larger d, the AAE may be larger when the memory is small, but the AAE also drops faster as the memory increases. 2) The insertion throughput decreases as the memory or d increases, and d has a greater impact on the throughput because larger d indicates more memory accesses.

5.3 Finding Persistent items

5.3.1 *Comparison with Related Work.*

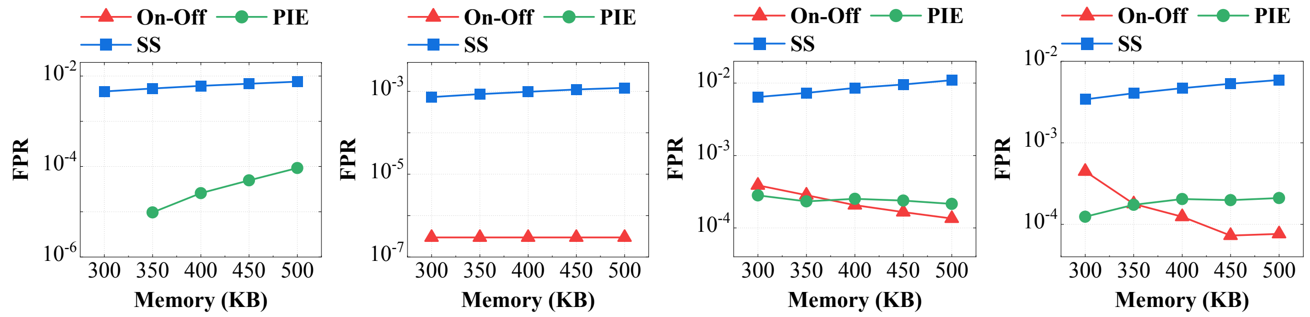
For finding persistent items, we first compare our On-Off sketch with 2 other algorithms: PIE [33] and Small-Space (SS) [32], which can work without any assumptions on the sizes of time windows. For the On-Off sketch, we set $w = 8$, which means there are 8 KV pairs in each bucket. We use AVX2 SIMD instructions to implement our On-Off sketch. For other sketches, parameters are set according to their authors' recommendations. In experiments, the memory size ranges from 300KB to 500KB to expose differences among algorithms. Because PIE cannot work with small memory, it will use 50 times more memory as SS and On-Off sketch.

AAE (Figure 12(a)-12(d)): Our results show that, the AAE of our On-Off sketch is 70048 times and 29342 times lower than that of SS and PIE. There is no data of PIE in Figure 12(b), because PIE cannot work in the data center dataset, *i.e.*, it cannot report any items.

FNR (Figure 13(a)-13(d)): Our results show that, the FNR of our On-Off sketch is often 0, while the FNR of other algorithms is often larger than 0.5.

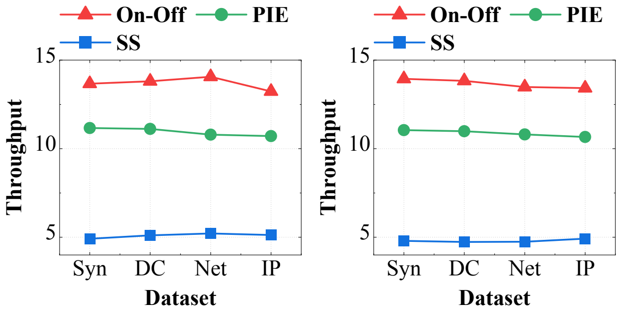
FPR (Figure 14(a)-14(d)): In Figure 14(a), there is no data of our On-Off sketch because the FPR is 0 in the synthetic dataset. In Figure 14(b), there is no data of PIE, because PIE cannot work in the data center dataset. In the network dataset and IP trace dataset, the FPR of our On-Off sketch is around 44.1 times and 1.42 times lower than that of SS and PIE.

Throughput (Figure 15(a)-15(b)): Our results show that, for 300KB memory, the insertion throughput of our On-Off sketch is around 2.68 times and 1.25 times higher than that of SS and PIE. For 500KB



(a) Synthetic (b) Data center (c) Network (d) IP trace

Figure 14: FPR on finding persistent items.



(a) 300KB (b) 500KB

Figure 15: Throughput on finding persistent items.

memory, the insertion throughput of our On-Off sketch is around 2.84 times and 1.26 times higher than that of SS and PIE.

Analysis: Before showing the efficiency of our On-Off sketch, we first analyze the shortcomings of prior algorithms.

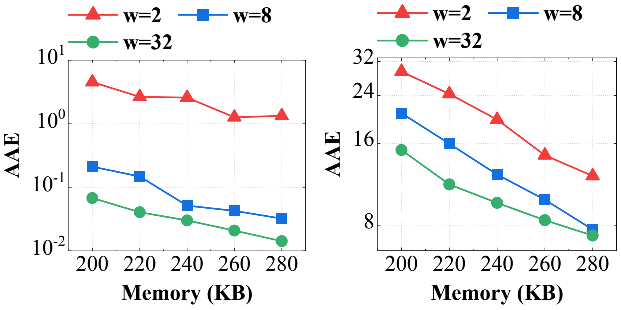
1) For SS, the root cause of its inaccuracy is its space-inefficiency. First, there are many pointers in its data structure, which is space-consuming. Second, it has to keep all items sampled, so there are many non-persistent items stored, which also take up much space. To keep a small memory, the sample rate of SS has to be lower, which increases its error. We can find that FPR of SS decreases with memory increases. It is because that if the sample rate is too low, more space only leads to more non-persistent items mistakenly recognized as persistent items. The throughput of the SS is the lowest because hash collisions in the hash table will increase the number of memory accesses.

2) For PIE, although it uses 50 times memory as the On-Off sketch, its accuracy is still low. As shown in Section 4.2.3, PIE's space complexity is $O(\|p\|_1)$, which is much larger than that of our On-Off sketch, because it has to record much information of non-persistent items. There will be many hash collisions in PIE with small memory. Because PIE will ignore all cells where collisions happen, it tends to underestimate the persistence when its memory is small. Therefore, in the network and IP trace dataset, the FPR of PIE is comparable to our On-Off sketch, while its FNR is much larger than our On-Off sketch. As shown in Figure 14(a) and Figure 14(d), the FPR of PIE may increase with memory increasing. It is because its degree of underestimation decreases when it has more memory. Though PIE's data structure is more memory-efficient, it has to encode every ID of incoming items, which lowers its throughput.

3) Compared with SS and PIE, our On-Off sketch is much more space-efficiency. Our data structure neither applies pointers nor has many empty cells. In addition, non-persistent items will be replaced quickly and will not take up much space. Therefore, the accuracy of our On-Off sketch is much better. We can find that the FNR of our On-Off sketch is often 0. It is because our On-Off sketch will only overestimate the persistence. Due to fewer memory accesses and the acceleration of SIMD instruction, the throughput of our On-Off sketch is the best.

5.3.2 *Parameter Setting.*

As shown in Section 3.3, we have two parameters in finding persistent items: l and w. To evaluate the influence of parameter setting, we fix the memory usage of the On-Off sketch and vary the value of w. In experiments, the memory size ranges from 200KB to 280KB.



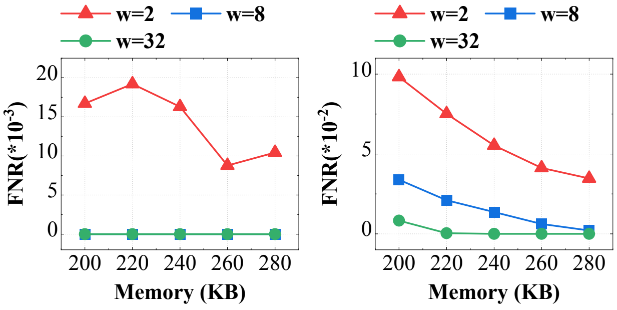
(a) Synthetic (b) IP trace

Figure 16: AAE of the On-Off sketch.

AAE (Figure 16(a)-16(b)): Our results show that, in the synthetic dataset, the AAE of $w = 32$ is around 74.4 times and 2.56 times lower than that of $w = 2$ and $w = 8$. However, in the IP trace dataset, the gap among AAEs under different parameters is smaller.

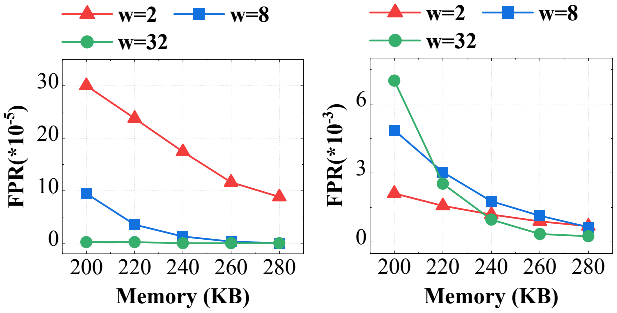
FNR (Figure 17(a)-17(b)): Our results show that, in the synthetic dataset, the FNR of $w = 32$ and $w = 8$ is often 0. In the IP trace dataset, when the memory is only 200KB, the FNR of $w = 32$ is around 11.9 times and 4.1 times lower than that of $w = 2$ and $w = 8$.

FPR (Figure 18(a)-18(b)): Our results show that, in the synthetic dataset, when the memory is only 200KB, the FPR of $w = 32$ is around 140 times and 44 times lower than that of $w = 2$ and $w = 8$. In the IP trace dataset, when the memory is only 200KB, the FPR of $w = 32$ is around 3.3 times and 1.4 times higher than $w = 2$ and



(a) Synthetic (b) IP trace

Figure 17: FNR of the On-Off sketch.



(a) Synthetic (b) IP trace

Figure 18: FPR of the On-Off sketch.

$w = 8$, while the memory is 280KB, the FPR of $w = 32$ is around 2.7 times and 2.6 times lower than $w = 2$ and $w = 8$.

Throughput (Figure 19): In this figure, “SIMD-8” means that we implement our On-Off sketch with AVX2 SIMD instructions and $w = 8$, while “w=8” means we set w to be 8 but the On-Off sketch does not use SIMD instructions. Our results show that the insertion throughput of $w = 2$ is around 13% and 40% higher than that of $w = 8$ and $w = 32$. In the IP trace dataset, the effect of SIMD instructions is relatively small. In the other three datasets, using SIMD instructions can increase the insertion throughput by 10%.

Analysis: 1) With enough memory, when w increases, the AAE and FNR often decrease. However, we can find that the FPR of $w = 32$ is the worst in the IP trace dataset when the memory is 200KB. The reason is that 200KB is too small for our On-Off sketch to find persistent items in the IP trace dataset. Because the length of ID is 8 byte in the IP trace dataset, the number of items we can store is less with the same space. As a result, the degree of overestimation is too high in 200KB, and many non-persistent items will be mistakenly recognized as persistent. With smaller w, the l increases, and the randomness of also increases. When the memory is too small, such randomness can reduce FPR because there are often some buckets whose degree of overestimation is small, so the AAE and FNR decrease. When there is enough memory, randomness increases FPR because there are more buckets whose degree of overestimation is too high, so the AAE and FNR increase.

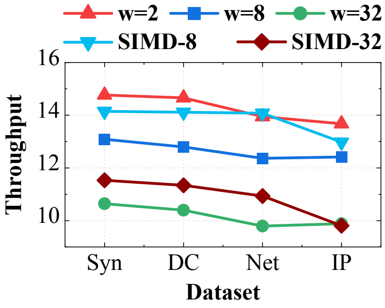


Figure 19: Throughput of the On-Off sketch.

2) The insertion throughput decreases as w increases. Because the number of memory accesses is small, the insertion throughput is still high when $w = 32$. In the IP trace dataset, the length of ID is 8 bytes, so we can only search 4 items at a time with AVX2 SIMD instructions. Therefore, SIMD does not perform well under the IP trace dataset. In the other three datasets, the length of ID is 4 bytes, so that we can search 8 items at a time, and we can find that SIMD instructions can obviously improve the throughput.

3) When there is enough space, choosing an appropriate w is a trade-off between accuracy and throughput. The larger w is, the higher the accuracy is, while the lower the throughput is. If the application demands high throughput, we should decrease the w (*e.g.*, $w = 2$). If the application demands high accuracy, we should increase w (*e.g.*, $w = 32$). Because the memory is often fixed, we can set l according to d and memory size.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose the On-Off sketch, which can address two typical problems about persistence – persistence estimation and finding persistent items. We derive the error bound of our On-Off sketch and prove that it does not underestimate persistence. In addition, we conduct extensive experiments on three real world datasets and a synthetic dataset. For persistence estimation, we theoretically prove that our On-Off sketch's error is always smaller than that of the CM sketch. In experiments, the On-Off sketch can achieve around 6.17 times smaller error and 2.2 times higher throughput. For finding persistent items, the space complexity of our On-Off sketch is much better than the state-of-the-art (PIE), and it can reduce the error up to 4 orders of magnitude and achieves 2.84 times higher throughput than prior algorithms in experiments.

We thank the anonymous reviewers for their valuable suggestions. In the future, we hope to further explore how to enable persistence queries at different sizes of time windows efficiently. We should note that it is still an open problem and the prior algorithm [25] which supports it just simply holds one data structure for each size of the time window.

ACKNOWLEDGMENTS

This work is supported by National Key RD Program of China (No. 2018YFB1004403), National Natural Science Foundation of China (NSFC) (No. 61832001, 61702016, 61672061, 62032020), Hunan Science and Technology Planning Project under Grant No.2019RS3019, the Hunan Provincial Natural Science Foundation of China for Distinguished Young Scholars under Grant 2018JJ1025, and the project of "FANet: PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications" (No. LZC0019).

REFERENCES

[1] 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 *SIGMOD Conference*, 2018.

[2] Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? - A survey on approximate query processing. *Data Sci. Eng.*, 3(4):379–397, 2018.

[3] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. Heavyguardian: Separate and guard hot items in data streams. In *SIGKDD*, 2018.

[4] T. Nan, C. Qing, and M. Prasenjit. Graph stream summarization: From big bang to big crunch. In *Proc. ACM SIGMOD*, 2016.

[5] Tongya Zheng, Gang Chen, Xinyu Wang, Chun Chen, Xingen Wang, and Sihui Luo. Real-time intelligent big data processing: technology, platform, and applications. *Sci. China Inf. Sci.*, 62(8):82101:1–82101:12, 2019.

[6] Z. Haida, H. Zengfeng, W. Zhewei, Z. Wenjie, and L. Xuemin. Tracking matrix approximation over distributed sliding windows. In *Proc. ICDE*, 2017.

[7] Zhengxin Wang, Jingbo Fan, Guoping Jiang, Jinde Cao, Min Xiao, and Ahmed Alsaedi. Consensus in nonlinear multi-agent systems with nonidentical nodes and sampled-data control. *Sci. China Inf. Sci.*, 61(12):122203:1–122203:14, 2018.

[8] Nacéra Bennacer Seghouani, Francesca Bugiotti, Moditha Hewasinghage, Suela Isaj, and Gianluca Quercini. A frequent named entities-based approach for interpreting reputation in twitter. *Data Sci. Eng.*, 3(2):86–100, 2018.

[9] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *Latin American Symposium on Theoretical Informatics*, 2004.

[10] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. *Acm Transactions on Computer Systems*, 21(3):270–313, 2003.

[11] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages & Programming*, 2002.

[12] 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*, SIGMOD '16, pages 1449–1463, New York, NY, USA, 2016. ACM.

[13] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, 2005.

[14] Daniel Ting. Data sketches for disaggregated subset sum and frequent item estimation. In *SIGMOD Conference*, 2018.

[15] Jiecao Chen and Qin Zhang. Bias-aware sketches. *Proceedings of the VLDB Endowment*, 10(9):961–972, 2017.

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

[17] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *IN AOFA '07: PROCEEDINGS OF THE 2007 INTERNATIONAL CONFERENCE ON ANALYSIS OF ALGORITHMS*, 2007.

[18] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *Acm Sigcomm Conference on Internet Measurement*, 2003.

[19] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer & System Sciences*, 31(2):182–209, 1985.

[20] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *Foundations of Computer Science*, 2016.

[21] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. Quantiles over data streams: An experimental study. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 737–748, New York, NY, USA, 2013. ACM.

[22] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Acm Sigmod International Conference on Management of Data*, 2001.

[23] Tong Yang, Haowei Zhang, Dongsheng Yang, Yucheng Huang, and Xiaoming Li. Finding significant items in data streams. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 1394–1405. IEEE, 2019.

[24] Saikat Guha, Jaideep Chandrashekar, Nina Taft, and Konstantina Papagiannaki. How healthy are today's enterprise networks? In *Acm Sigcomm Conference on Internet Measurement*, 2008.

[25] Frederic Giroire, Jaideep Chandrashekar, Nina Taft, Eve Schooler, and Dina Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *International Symposium on Recent Advances in Intrusion Detection*, 2010.

[26] Advanced persistent threat. http://www.usenix.org/event/lisa09/tech/slides/daly.pdf.

[27] Ibrahim Ghafir and Vaclav Prenosil. Advanced persistent threat attack detection: an overview. *Int J Adv Comput Netw Secur*, 4(4):5054, 2014.

[28] Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[29] Stuart Staniford, James A. Hoagland, and Joseph M. Mcalerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1-2):105–136, 2002.

[30] Qingjun Xiao, Yan Qiao, Mo Zhen, and Shigang Chen. Estimating the persistent spreads in high-speed networks. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 131–142. IEEE, 2014.

[31] Nicole Immorlica, Kamal Jain, Mohammad Mahdian, and Kunal Talwar. Click fraud resistant methods for learning click-through rates. In *Proceedings of the First International Conference on Internet and Network Economics*, WINE'05, pages 34–45, Berlin, Heidelberg, 2005. Springer-Verlag.

[32] Bibudh Lahiri, Jaideep Chandrashekar, and Srikanta Tirthapura. Space-efficient tracking of persistent items in a massive data stream. In *Acm International Conference on Distributed Event-based Systems*, 2011.

[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] Mohammad Amin Shokrollahi and Michael Luby. Raptor codes. *IEEE Transactions on Information Theory*, 6(6):213–322, 2009.

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

[36] Intel sse2 documentation. https://software.intel.com/en-us/node/683883.

[37] Source code related to On-Off sketch. https://github.com/Sketch-Data-Stream/On-Off-Sketch.

[38] 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*, SIGMOD '15, page 795–810. ACM, 2015.

[39] 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*, SIGMOD '18, page 1037–1052. ACM, 2018.

[40] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner. Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units. *Proc. VLDB Endow.*, 2(1):385–394, 2009.

[41] Intel advanced vector extensions programming reference. https://software.intel.com/file/36945.

[42] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1015–1025. ACM, 2020.

[43] Junzhi Gong, Tong Yang, Haowei Zhang, Hao Li, Steve Uhlig, Shigang Chen, Lorna Uden, and Xiaoming Li. Heavykeeper: An accurate algorithm for finding top-k elephant flows. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, pages 909–921. USENIX Association, 2018.

[44] David MW Powers. Applications and explanations of Zipf's law. In *Proc. EMNLP-CoNLL*. Association for Computational Linguistics, 1998.

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

[46] The data center dataset. http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html.

[47] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In Mark Allman, editor, *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, pages 267–280. ACM, 2010.

[48] The Network dataset Internet Traces. http://snap.stanford.edu/data/.

[49] The caida anonymized 2016 internet traces. http://www.caida.org/data/overview/.

[50] Hash website. http://burtleburtle.net/bob/hash/evahash.html.