SteadySketch: A High-Performance Algorithm for Finding Steady Flows in Data Streams

Zhuochen Fan, Xiangyuan Wang, Xiaodong Li, Jiarui Guo, *Graduate Student Member, IEEE*, Wenrui Liu, Haoyu Li, Sheng Long, Zheng Zhong, Tong Yang, *Member, IEEE*, Xuebin Chen, and Bin Cui, *Fellow, IEEE*

***Abstract*— In this paper, we study steady flows in data streams, which refers to the flows whose arrival rate is always non-zero and around a fixed value for several consecutive time windows. To find steady flows in real time, we propose a novel sketch-based algorithm, SteadySketch, aiming to accurately report steady flows with limited memory. To the best of our knowledge, this is the first work to define and find steady flows in data streams. The key novelty of SteadySketch is our proposed reborn technique, which reduces the memory requirement by 75%. Our theoretical proofs show that the negative impact of the reborn technique is small. Experimental results show that, compared with the two comparison schemes, SteadySketch improves the Precision Rate (PR) by around 79.5% and 82.8%, and reduces the Average Relative Error (ARE) by around 905.9× and 657.9×, respectively. Finally, we provide three concrete cases: cache prefetch, Redis and P4 implementation. As we will demonstrate, SteadySketch can effectively improve the cache hit ratio while achieving satisfying performance on both Redis and Tofino switches. All related codes of SteadySketch are available at GitHub.**

***Index Terms*— Data streams, measurement, steady flows, sketch algorithm, cache, Redis, P4, performance.**

I. Introduction

A. Background and Motivation

NOWADAYS, network measurement and monitoring have become a research hotspot in the network field. It provides indispensable information for various network management tasks, such as traffic behavior analysis [2], [3], quality of service/experience [4], [5], performance diagnosis [6], and anomaly detection [7], [8], [9], [10]. In addition to the above tasks, a very important research interest is to define and find new patterns in high-speed data streams, such as burst flows [11], periodic flows [12], and quadratic flows [13], *etc.*

Manuscript received 6 May 2023; revised 21 March 2024 and 15 July 2024; accepted 12 August 2024; approved by IEEE/ACM Transactions on Networking Editor S. M. Kim. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB2901504; in part by China Postdoctoral Science Foundation under Grant 2023TQ0010, Grant GZC20230055, and Grant 2024M750102; and in part by the National Natural Science Foundation of China (NSFC) under Grant U20A20179 and Grant 62372009. The preliminary version of this paper titled "SteadySketch: Finding Steady Flows in Data Streams" [1] was published in the Proceedings of the 2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS), July 27, 2023 [DOI: 10.1109/IWQoS57198.2023.10188743]. *(Corresponding author: Tong Yang.)*

Zhuochen Fan, Xiangyuan Wang, Xiaodong Li, Jiarui Guo, Wenrui Liu, Haoyu Li, Sheng Long, Zheng Zhong, Tong Yang, and Bin Cui are with the National Key Laboratory for Multimedia Information Processing, School of Computer Science, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: fanzc@pku.edu.cn; wangxiangyuan@stu.pku.edu.cn; lxdong0128@gmail.com; ntguojiarui@pku.edu.cn; liuwenrui@pku.edu.cn; lihy@pku.edu.cn; lgenhogns@pku.edu.cn; zheng.zhong@pku.edu.cn; yangtong@pku.edu.cn; bin.cui@pku.edu.cn).

Xuebin Chen is with Heibei Key Laboratory of Data Science and Application, Tangshan Key Laboratory of Data Science, North China University of Technology, Tangshan 063210, China (e-mail: chxb@ncst.edu.cn).

Digital Object Identifier 10.1109/TNET.2024.3444488

This paper defines steady flow, a new pattern in network data streams. In real-world data streaming scenarios, data stream often arrives at high speed, and each flow that composes it may appear multiple times. We divide the data stream into many time windows. Given a time window and a flow e in this window, suppose e appears x times, we define the arrival rate of e as x. For p continuous time windows, if the arrival rate of a flow is non-zero and steady (the variance of arrival rate is less than a given threshold), it should be categorized as a steady flow.

Since steady flow has certain "predictability", it is an important data stream pattern with numerous applications. Here, we show three typical ones as follows. 1) Wireless Sensor Networks. Sink node of WSN is responsible for collecting and processing the data from other sensor nodes [14]. Generally, the sensor node that sends steady flows would have higher data reliability, which is important for data processing of sink nodes. 2) Network Bandwidth Pre-Allocation. In any highly dynamic network, a flow with a steady connection and transmission speed often means that it has higher priority than the transient flows which accounts for the majority of the network [15]. Thus, we can pre-allocate the bandwidth for these flows in advance to improve the quality of network service [16]. 3) Steady Cache Line. In this scenario, a flow refers to a cache line in the cache replacement problem. A steady cache line has a higher probability of recurrence in the next few time windows [17]. We can reduce cache thrash by avoiding steady cache lines being evicted and ultimately improve cache hit ratios. In addition to these more intuitive ones, there is reason to believe that there are many other applications of the steady flow that should not be underestimated, for example, speeding up some machine learning algorithms by reducing running time [18].

To the best of our knowledge, this is the first work to define steady flows, and no existing literature has proposed the same or similar definition. The related problem is finding persistent flows [19], [20], [21], [22] and K-persistent spread estimation [23], [24], [25]. A flow is defined as a persistent flow if the number of time windows where it appears exceeds a given threshold [22]. In other words, the statistical process

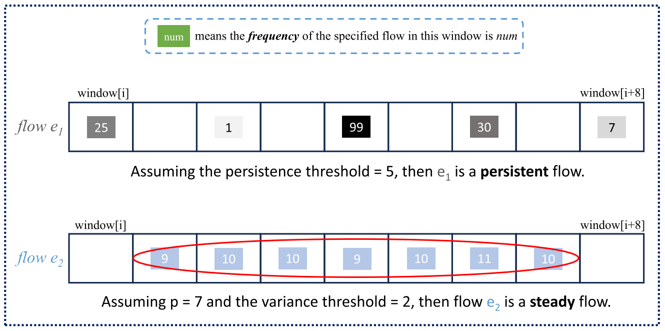

num means the frequency of the specified flow in this window is num
window[i]
window[i+8]
flow e1
25
1
99
30
7
Assuming the persistence threshold = 5, then e1 is a persistent flow.
flow e2
9
10
10
9
10
11
10
Assuming p = 7 and the variance threshold = 2, then flow e2 is a steady flow.

Fig. 1. The persistent flow e_1 and the steady flow e_2 are two different types of flows. e_1 is not concerned with its frequency size and continuity in the occurrence window. As long as the frequency in that window is not 0, whether it is 1 or 99, the effect is the same: persistence is incremented by 1. Conversely, e_2 is concerned with both its frequency size and continuity in the occurrence window.

of a persistent flow is only related to whether the flow has appeared in a window, but not to the size or variance of the arrival rate. For steady flows, instead, we focus on the size of the arrival rate and the variance of the arrival rate across several continuous windows. Figure 1 is shown to more visually distinguish between persistent flows and steady flows, where the arrival rate of steady flow is the frequency (number of occurrences) in the window. In short, there is currently no solution specifically designed for detecting steady flows in real time. In other words, existing solutions cannot ideally detect steady flows even if they can be forcibly implemented.

B. Our Proposed Solution

In this paper, in order to find steady flows, we first propose a strawman solution composed of multiple Count-Min (CM) sketches [26]. However, we find that there are limitations in terms of speed, accuracy, and memory-efficiency. To address these limitations, we propose the first version to optimize speed. Then, we propose the second version to optimize accuracy and the third to further optimize memory usage. Finally, we integrate the three versions and present the final version, namely SteadySketch. Our key novelty lies in the memory optimization: *reborn technique* (see Section III-D). Below we show the rationale of the reborn technique.

The key idea of reborn technique is rebirth with offset variance calculation. In data stream scenarios, flows with large arrival rates are always more important than flows with smaller arrival rate, but we cannot know the size of the arrival rate in advance. Therefore, it seems that we have to use large counters (*e.g.*, 32 bits or even 64 bits) for all flows to record their arrival rate. However, using large counters will make our data structure too large to be held in a small cache. Therefore, we aim to use small counters to record both small arrival rates and large arrival rates. If a small counter overflows, we regard this as a *rebirth*: a finite cyclic group $\mathbb{Z}_{256}$ in theory [27]. Once a rebirth occurs, it will cause the loss of the most significant bit of the frequency, which can further cause the loss of accuracy in the variance calculation. For example, if there are three counters: 253, 254, and 255, suppose that the incoming flow updates the counter of 255 to 0, we call it a rebirth. In this case, the normal variance calculated by the reborn values ($\mathcal{F}_v\{253, 254, 0\}$=21421)[1] in the counters is very large, while actually the real variance ($\mathcal{F}_v\{253, 254, 256\}$=2.33) is very small. In this way, we propose the *offset variance calculation*. The key idea is to offset the values in the counters by a fixed value, and recalculate the variance of the new values to make the calculated variance close to the real value. In the above example, we can offset the values {253, 254, 0} by 128 and get {125, 126, 128} ($-128 = 128$ in $\mathbb{Z}_{256}$). The variance calculated is the same as the real value, *i.e.*, $\mathcal{F}_v\{253, 254, 256\}$=$\mathcal{F}_v\{125, 126, 128\}$. Therefore, if a flow is a steady flow, we can always accurately report it as a steady flow. With a small probability P', there will be a few false-positive reports: some flows that are not steady might be reported as steady flows. According to our theoretical results, P' is very small, and the negative impact of reborn technique on accuracy is very minor (see Section IV-B for details).

Further, our experimental results show that SteadySketch achieves higher accuracy by accommodating much more counters in the same memory space. Compared with the strawman solution, SteadySketch is more memory efficient, more accurate and faster: it achieves over 96% PR with 50KB memory in CAIDA Dataset [28] for finding steady flows, and the throughput has been improved by more than 1.7× on average. Finally, we implement SteadySketch on cache replacement, Redis, and a P4 switch, respectively, and the evaluation results show that SteadySketch can significantly improve the system performance. More details are provided in Section V. We have released our source code at GitHub [29].

Key Contributions:

- We propose and define a new problem: finding steady flows in data streams, which has not been studied before but is important to many applications.
- We propose a novel sketch named SteadySketch to address the above problem with high accuracy and high speed in small memory.
- We provide concise theoretical results by strict derivation. We theoretically prove that the reborn technique has a slight effect on accuracy and then give the error bound of reborn.
- We conduct extensive experiments, and the results show that our solution significantly outperforms the two comparison schemes in finding steady flows. Particularly, SteadySketch improves the PR by about 79.5% and 82.8%, and decreases the ARE by about 905.9× and 657.9×, respectively.

II. Problem Statement & Related Work

In this section, we first define the problem for finding steady flows in data streams in Section II-A, and then introduce the basic sketch algorithms that may be involved in Section II-B. The symbols frequently used in this paper are shown in Table I.

A. Problem Statement

Steady refers to the situation which continues or develops gradually without any interruptions and is not likely to change

[1] $\mathcal{F}_v(.)$ is a function of variance calculation that returns the calculated variance value.

TABLE I
SYMBOLS FREQUENTLY USED IN THIS PAPER

Notation	Meaning
e	a distinct flow in data streams
t	current time window
w	the number of bits or counters in each bucket
γ	the number of buckets in each array in RollingSketch
$g_k(.)$	k^{th} hash function of the SteadyFilter
$f_d(.)$	d^{th} hash function of the RollingSketch
$\mathcal{P}$	the probability of replacing the flow in Stage 2
$\langle e,t\rangle$	the steady flow e reported with the time window of t
$\langle e,t_s,t_e\rangle$	the persistent steady flow reported with the start time of t_s and the end time of t_e

quickly. In data streams, it manifests as the arrival rate of a flow which fluctuates slightly around a fixed value without interruption for a period of time. Therefore, we characterize steady flow from two aspects: continuity and stability.

Temporary steady flow: Given a data stream, we divide it into fixed-width time windows $w_1, w_2, w_3, \ldots$. Given a flow e and a variance threshold H, the arrival rate of e in the time windows are $r_1, r_2, r_3, \ldots$. The function of $\mathcal{F}_v(.)$ is to calculate the variance and return the variance value. If there exist p consecutive time windows $w_{t-p+1}, \ldots, w_{t-1}, w_t$, where

$$\mathcal{F}_v(r_{t-p+1}, \ldots, r_{t-1}, r_t) \le H$$

and

$$r_i > 0,\ \forall i \in \{t, t-1, \ldots, t-p+1\}$$

then e is one steady flow, and we report it as $\langle e,t\rangle$. t is the time window of e becoming a standard steady flow.

Persistent steady flow: The data stream is divided into multiple fixed-width time window. In each time window, there could be multiple temporary steady flows. Given a series of temporary steady flows $\{\langle e_1,t_1\rangle, \langle e_2,t_1\rangle, \langle e_1,t_2\rangle, \ldots\}$, we could try to merge steady flows with the same flow ID. Persistent steady flows $\langle e,t_1,t_2\rangle$ are reported only when steady flows $\langle e,t\rangle\,(t_1 \le t \le t_2)$ are all found. It indicates that the steady process of e lasts from t_1 to t_2. Therefore, persistent steady flows can be simply regarded as a subset of temporary steady flows, and their applications are consistent, as shown in Section I-A.

B. Typical Sketches for Data Streams

1) Membership Query: A Bloom filter [30] is a compact data structure with high spatial efficiency. It uses bit groups to represent a set concisely, and can judge whether a flow belongs to the set. It consists of an array of m bits and is associated with k independent hash functions. Given a flow, it is hashed to k different mapped bits and set to 1. For membership query, the Bloom filter checks whether all k mapped bits are 1. Because it is space-saving and efficient, it is widely used and has produced many variants [31], [32], [33].

2) Frequency Estimation: The Count-Min (CM) sketch [26] consists of d arrays $A_i (1 \le i \le d)$, each consists of w counters, and A_i is associated with a hash function $h_i(.)$. Given an incoming flow e, it increments the d mapped counter

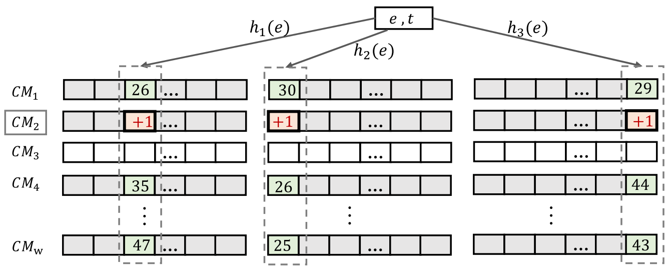


Fig. 2. Examples of insertion in the strawman solution with 3 hash functions and w CM sketches. The counters marked green are frequencies of previous p time windows and the sketch CM_2 is selected to represent the current time window.

$A_i[h_i(e)]$ by 1. To query e, CM sketch only reports the minimum among the d mapped counters. CM sketch has been widely used in many scenarios and has derived many variants, such as Conservative Update (CU) [34], Count [35], and many other typical sketches [11], [36], [37], [38], [39]

3) Finding Frequent Flows: The most typical algorithms are Space-Saving [40] and Unbiased Space-Saving [41]. They both use a data structure called Stream-Summary to keep the top-k frequent/elephant flows. When the Stream-Summary is full and a non-recorded flow arrives, Space-Saving directly replaces the least frequent flow with this new flow, while Unbiased Space-Saving uses probability to replace the least frequent flow to achieve unbiased estimation. State-of-the-art sketch-based solutions for finding top-k elephant flows mostly prefer to accurately separate elephant flows from large-scale mice flows, such as ElasticSketch [42], HeavyKeeper [43], and LadderFilter [44], *etc.*

III. THE STEADYSKETCH

In this section, we introduce different solutions for finding steady flows. First, we propose the strawman solution based on CM sketch [26]. Further, we propose optimization schemes in terms of speed, accuracy and memory utilization, and present the final version in Section III-E. Finally, we discuss the difference of temporary and persistent steady flows, and add a new data structure for finding persistent steady flows.

A. The Strawman Solution

As shown in Figure 2, our strawman solution is based on CM sketch [26] (Section II-B.2). We construct the strawman solution with w CM sketches, in which p sketches to contain the flow information in past p time windows, one CM sketch is used to contain the flow information of current time, and the other is reserved for the next time window. Thus, the w must be set greater than $p+2$.

For each incoming flow e with time window t, the CM sketch representing the current time hashes it to a bucket in each array, and increments the counters by 1. Then, the variance of frequency is calculated by the counters of the mapped buckets in previous p sketches. If the variance is less than the steady threshold H, we report the steady flow $\langle e,t\rangle$.

The strawman solution could indeed be used to report steady flows. However, its low accuracy, low throughput, and large memory consumption restrict its practical applications.

B. Speed Optimization

The strawman solution consists of multiple sketches that require multiple memory accesses to calculate the variance, which greatly reduces throughput. Therefore, we propose a new data structure that merges multiple sketches into one, reducing the time complexity of the query to $O(1)$. Inspired by the principle of locality, we set the counters in the same location of different sketches into one bucket. In this way, we construct a sketch with only d bucket arrays, and each bucket consists of multiple counters to maintain the frequencies of different time windows. When querying frequencies to calculate the variance, only one memory access is required per array. Through the above optimization, memory accesses are reduced by w times, which greatly improves the throughput.

C. Accuracy Optimization

The key idea of accuracy optimization lies in continuity checking. It refers to checking if the flow appears consecutively in previous p time windows. Due to limited memory and hash collisions, it is hard to determine whether it is interrupted for some small flows. Especially in real-world data streams, most flows are small/mouse flows, checking the continuity of flows with counters of sketch is hard and space consuming.

Based on the above challenges, we propose SteadyFilter to filter out those flows that are uninterrupted in previous p time windows (continuity checking). Inspired by the Bloom filter [30] (Section II-B.1), we utilize the bits group to maintain the arrival information for the flows. In SteadyFilter, each bucket consists of multiple bits to record the appearance of a flow in different windows. In this way, the accuracy is greatly improved at the cost of slight extra memory usage.

D. Memory Optimization

The key novelty of this paper lies in this optimization. The main innovation is the proposal of the reborn technique. In SteadySketch, we optimize the memory efficiency and reduce the counters to 8 bits, which is sufficient for most flows. However, this poses challenges for accurate frequency statistics of elephant flows due to their risk of overflow. Thus, we propose the reborn technique, the key of which is rebirth and offset variance calculation.

Rebirth: In abstract algebra, a **cyclic group** is a group that can be generated by one element $G = \{e, a, a^{-1}, a^2, a^{-2}, \cdots\} = \langle a \rangle$. It is already well known that every finite cyclic group with n elements is isomorphic to a group of integers modulo n, which is $\mathbb{Z}_n = \{0, 1, \cdots, n-1\}$. Inspired by the concept of group, the value of an 8-bit counter can be regarded as a finite cyclic group $G = \mathbb{Z}_{256} = \{0, 1, \cdots, 255\}$. Once the frequency is greater than 255, the rebirth is triggered and the value increments from 0 again.

Offset Variance Calculation: The rebirth being triggered indicates that the frequency has not been well recorded. Thus, we propose the *offset variance calculation* to ensure the accuracy of variance calculation. Initially, we calculate the normal variance using the raw data as usual. Next, we calculate the offset variance. The innovation of offset variance calculation is that we offset the frequencies by a fixed value: we increment the values by 128. In this way, the overflowed data can be contiguous. Then, we calculate the variance again with the new offset values.

Example: A steady flow appears 248, 258, 260 times in three time windows, recorded as 248, 2, 4 in the respective counters. In the normal variance calculation, the variance result calculated by the original data in the counters is 13339, while the real frequency variance is 27. In this case, the normal variance is significantly larger than the real frequency variance. In the offset variance calculation, the frequency is recalculated as 120, 130, 132, and the variance value is the same as the real variance.

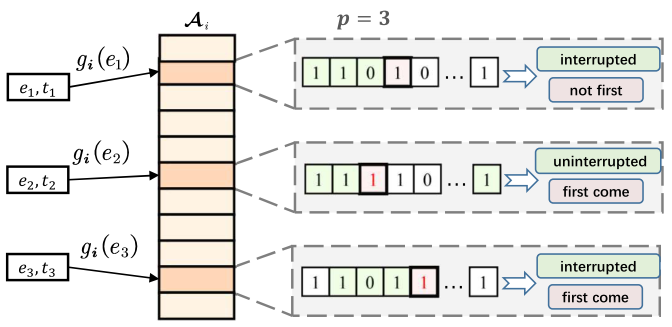


Fig. 3. Examples of insertion in SteadyFilter with one hash function. All the buckets are initialized to (1, 1, 0, 1, 0, $\cdots$, 1). The bits marked red represent the current time window, while the green ones are previous p time windows.

E. Our Final Version

Integrating the above three techniques, SteadySketch includes two parts: a SteadyFilter and a RollingSketch. Next, we introduce the two parts in details as follows.

1) SteadyFilter:

Similar to the typical Bloom filter [30], SteadyFilter consists of k bucket arrays $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k$, and k hash functions $g_1(.)$, $g_2(.)$, $\cdots$, $g_k(.)$. Each bucket consists of w bits, representing the appearance of a flow in w time windows. When querying, if the i^{th} bit in the bucket is set, it indicates that the flow has reached in the i^{th} time window.

Figure 3 shows the data structure of SteadyFilter of the version with one hash function. In this figure, e_1, e_2 and e_3 are the flows inserted into SteadyFilter at time of t_1, t_2 and t_3 respectively. The bits marked red represent the time windows of t_1, t_2 and t_3 in the mapped buckets, while the green ones are the previous p time windows. If a flow is recognized as uninterrupted and first comes in the current time window, it would be marked with T_{pf}, like e_2 in Figure 3.

Insertion (Algorithm 1): Given an incoming flow e and timestamp t, we hash it to k mapped buckets of SteadyFilter $\mathcal{A}_1[g_1(e)]$, $\mathcal{A}_2[g_2(e)]$, $\cdots$, $\mathcal{A}_k[g_k(e)]$ using the hash function $g_1(.)$, $g_2(.)$, $\cdots$, $g_k(.)$. Next, we first select the bucket $\mathcal{A}_i[g_i(e)]$ and check the bit of $\mathcal{A}_i[g_i(e)][t\%w]$ to determine whether it comes the first time. Then, we select the bit of $(t-j)\%w$ $(1 \le j \le p)$ in the bucket separately to check the continuity of flow e. Once the flow is identified as first-arriving and contiguous, RollingSketch (explain later) manipulates this flow by setting the temporary variable T_{pf}. Finally, we set the

Algorithm 1 Insertion Procedure for SteadyFilter

```
Input: input a coming flow $e$ with the timestamp $t$
Output: output the flag $T_{pf}$
1  $T_{pf} = 0$; $T_f = 0$; $T_p = 0$;
2  for $i \leftarrow 1$ to $k$ do
3      $T_f$ += $\mathcal{A}_i[g_i(e)][t \% w]$
4      for $j \leftarrow 1$ to $p$ do
5          $T_p$ += $\mathcal{A}_i[g_i(e)][(t-j) \% w]$;
6      $\mathcal{A}[g_i(e)][t \% w] = 1$;
7  if $T_p == k \times p$ && $T_f == 0$ then
8      $T_{pf} = 1$
```

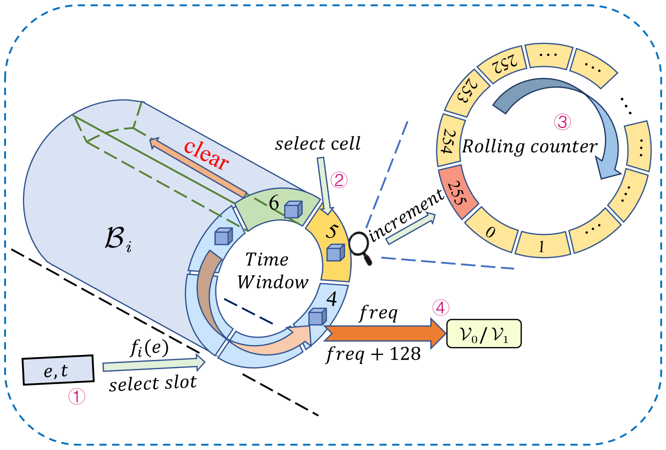


Fig. 4. Examples of insertion in RollingSketch, assuming one hash function is used and p is set to 4. In addition, ① ② ③ ④ in the figure represent the sequence of insertion.

bit of $\mathcal{A}_i[g_i(e)][t\%w]$ to 1, indicating that the flow has come to the current time window.

2) RollingSketch:

RollingSketch consists of d arrays, each of which being associated with a hash function. Figure 4 shows the data structure of RollingSketch with one array, which looks like a cylinder. In this paper, the cylinder is set to be cut into γ slots, each slot consists of w cells. Each cell records the flow frequency in a time window. Thus, a slot can record the flow frequencies of w time windows.

Insertion (Algorithm 2): We first select the slots $\mathcal{B}_1[f_1(e)]$, $\mathcal{B}_2[f_2(e)]$, $\cdots$, $\mathcal{B}_d[f_d(e)]$ in each array with hash functions $f_1(.)$, $f_2(.)$, $\cdots$, $f_d(.)$. We use the $\mathcal{S}_i(1 \le i \le d)$ to denote the slot selected in each array. Then, we select the cell $\mathcal{S}_i[t\%w]$ and increment it by 1. The insertion process is shown in Figure 4 and marked as ① ② and ③. It can be summarized as selecting the slot representing the flow in each array, then choosing the cell representing the current time window in the mapped slot, and turning the counter clockwise one grid.

Report (Algorithm 2): First, we select the values in the cells of $\mathcal{S}_i[(t-j)\%w](1 \le j \le p)$ to calculate the normal variance $\mathcal{V}_0$. Next, we perform the *offset variance calculation* (Section III-D): increment the cells by 128 and recalculate the variance to calculate the variance $\mathcal{V}_0$ and $\mathcal{V}_1$. Once one of the two variances is smaller than the threshold H, we report

Algorithm 2 Operations Procedure for RollingSketch

```
Input: input a coming flow $e$ with the timestamp $t$
Output: output the steady flow $(e, t)$
1  for $i \leftarrow 1$ to $d$ do
2      $B[f_i(e)][t \% w]$ ++;
3      if $T_{pf} == 1$ then
4          for $j \leftarrow 1$ to $p$ do
5              $t_l = (t - j) \%w$;
6              $G[0][j] \leftarrow B[f_i(e)][t_l]$;
7              $G[1][j] \leftarrow (G[0][j] + 128)\%256$;
8  $\mathcal{V}_0 = \mathcal{F}_v(G[0])$; // $\mathcal{F}_v$ is the function of
       variance calculation
9  $\mathcal{V}_1 = \mathcal{F}_v(G[1])$;
10 if $\mathcal{V}_0 < H$ || $\mathcal{V}_1 < H$ then
11     report steady flow $\langle e, t \rangle$;
```

the flow e as a steady flow $\langle e, t\rangle$, where t represents the time when the flow e becomes a steady flow. In Figure 4, the report process is marked as ④, the blue cells representing the previous frequencies are read out in a cache line, and used to calculate the variance $\mathcal{V}_0$ and the offset variance $\mathcal{V}_1$. If the variances meet the condition, flow e will be reported as one steady flow $\langle e, t\rangle$.

Algorithm 3 Clear Procedure

```
Input: current time window $t$ and the time window $t_p$
       of last flow
1  if $t \neq t_p$ then
2      for each $i \in [1, d]$ do
3          $t_n$ = $(t + 1) \% w$;
4          for each $j \in [0, len(A_i)]$ do
5              $\mathcal{A}_i[j][t_n]$ = 0;
6          for each $j \in [0, len(B_i)]$ do
7              $\mathcal{B}_i[j][t_n]$ = 0;
```

Clear Strategy (Algorithm 3): As the time window increases, there are not enough bits and cells in each bucket to accommodate all time windows. If we select the cells representing the current time window to clear, we have to put the incoming packets into the buffer queue first, which is not practical in high-speed data streams. Therefore, we select the bits and cells of the next time window in the SteadyFilter and RollingSketch, and set them to 0 when the time window switches. In this way, the clear operation can be implemented in parallel without much impact on throughput.

3) Summary: In SteadySketch, we separately design different processing schemes for various types of flows in data streams as follows:

Case 1: Elephant flows: such flows may lead to counter overflows. We propose the reborn technique to make amendments for the accuracy loss caused by overflow.

Case 2: Medium-sized flows: the frequencies of such flows can be well recorded and calculated by the counters normally.

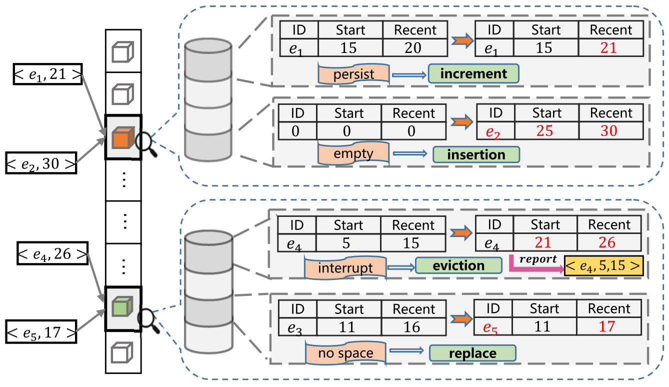


Fig. 5. Insertion examples in stage 2 and the parameter p is set as 5.

Case 3: Small flows: the majority of flows in a data stream. It is hard to identify the continuity of a flow in limited memory. In this case, we use SteadyFilter to specifically identify the continuity of flows.

F. Finding Temporary and Persistent Steady Flows

As for finding temporary steady flows, the above algorithm is fully capable of reporting the temporary steady flows in every time window. However, it cannot effectively report persistent steady flows. In order to find persistent steady flows, we additionally add Stage 2 to cooperate with RollingSketch, where Stage 2 is named to distinguish it from the part of finding temporary steady flows as Stage 1 (*i.e.*, SteadySketch itself or the strawman solution). Stage 2 is designed to record the steady duration of the steady flows. Figure 5 shows the data structure and the insertion process of Stage 2. Below we show the details.

Data Structure: Stage 2 consists of an array of l buckets, and each bucket has four cells. Let G_i be the i^{th} bucket, and $G_i[j]$ be the j^{th} cell in bucket G_i. For each cell, it is designed with the following three fields: 1) A `flow_ID` field $G_i[j].ID$ records the ID of the steady flows, and we call the flow in the cell as the *residing flow*. 2) A `start time` field $G_i[j].t_s$ records the start time of the steady flow as a residing flow. 3) A `recent time` field $G_i[j].t_e$ records the recent time of the steady flow as a residing flow. The duration of a steady flow is represented by the interval ΔT, *i.e.*, $\Delta T = G_i[j].t_e - G_i[j].t_s$. All fields in the data structure are initialized to 0 or $Null$.

Insertion: When inserting flow e into Stage 2, we first map the flow to a bucket G_v by computing the hash function $h(e)$. Next, we try to insert it. There are four cases as follows:

Case 1: **Insertion**. If e is not in the bucket and there is still at least one empty cell. We select an empty cell, insert $\langle e, t\rangle$ into the bucket, and set the ID of e to $G_v[j].ID$, the timestamp t_s, t_e to $t-p$ and t, respectively.

Case 2: **Eviction**. If the recent time t_e of the residing flow e_p in the cell is not the last time $t - 1$, it means that e_p is no longer steady. Thus, we report the flow e_p as a persistent steady flow $\langle e_p, G_v[j].t_s, G_v[j].t_e\rangle$. Then, the fields in cell are replaced by $[e,\ t-p,\ t]$.

Case 3: **Increment.** If e is in the bucket, and $G_v[j].t_e$ equals to $t - 1$. It shows that e is still steady. We increment $G_v[j].t_e$ by 1.

Case 4: **Replacement**. If the bucket cannot satisfy the above three cases, it means that all flows in the bucket are still steady. In that case, we use ΔT to find the shortest persistent steady flow, and replace it with a certain probability. Then, we kick the residing flow with $\mathcal{P}$, and replace $G_v[j].ID$ and $G_v[j].t_e$ with $\langle e, t\rangle$, respectively, without $G_v[j].t_s$ replacement. The definition of $\mathcal{P}$ is shown in Equation (1):

$$\mathcal{P} = \frac{1}{t_e - \ t_s - p} \tag{1}$$

Examples: In Figure 5, we give examples of four insertion cases. For the temporary steady flow $\langle e_1, 21\rangle$ and $\langle e_2, 30\rangle$, both of them map into the same bucket. In the bucket, there is exactly one residing flow is e_1, and its steady process is on-ging. Thus, we directly increase the recent time of e_1. As for e_2, there is still one empty cell in the bucket, so we insert the information of flow e_2 into the empty cell directly. At the time of 26 and 17, RollingSketch report two temporary steady flows e_4 and e_5 respectively, which are mapped into the same bucket. We select the cell of e_4 and try to insert the temporary steady flow $\langle e_4, 26\rangle$, but the recent time of residing flow shows that the steady process of e_4 has been interrupted. Thus, we evict and report is as one persistent steady flow $\langle e_4, 5, 15\rangle$, and insert e_4 as a new one. For flow e_5, because there is no empty cell, we choose the flow with the shortest duration and perform probability replacement.

Report: There are two ways to report the persistent steady flows. The main way is to directly traverse the buckets of Stage 2 and return the flows, *i.e.*, ($\langle e, t_s, t_e\rangle$ pairs). There are also a few persistent steady flows reported by being evicted during flows insertion, such as the e_4 in Figure 5.

IV. Mathematical Analysis

In this section, we compare our algorithm with strawman solution and provide theoretical bounds to illustrate the superiority of our solution.

A. Role of SteadyFilter

Theorem 1: For any flow e that fails to appear consecutively from the $(t-p+1)$-th window to the t-th window, let q denote the probability of SteadySketch wrongly reporting $\langle e, t\rangle$ as a steady flow, $\hat{q}$ denote the probability of RollingSketch without SteadyFilter wrongly reporting $\langle e, t\rangle$ as a steady flow, and ε denote the false positive rate (FPR) of SteadyFilter, then

$$q \le \varepsilon \hat{q}. \tag{2}$$

Proof: Assume that e does not arrive in window $t_1, \cdots, t_k$, where $t - p + 1 \le t_1 < \cdots < t_k \le t$. Since our SteadySketch is composed of a SteadyFilter and a RollingSketch, a flow e is wrongly reported as a steady flow only if e cheats both data structures. Let A_i denote the event of SteadyFilter reports false positive in t_i-th window, B denote the event of RollingSketch wrongly reports e. From conditional probability we know that

$$\begin{aligned} q &= P(\text{SteadySketch reports } e) \\ &= P(A_1 \cdots A_k B) \\ &= P(B) \cdot P(A_1 \cdots A_k | B) \end{aligned}$$

$$= \hat{q}P(A_1 \cdots A_k|B). \tag{3}$$

Assume that whether SteadyFilter or RollingSketch makes mistakes are independent and whether the SteadyFilter makes mistakes on each window are independent, we get

$$\begin{aligned} q &= \hat{q} \cdot P(A_1 \cdots A_k) \\ &= \hat{q} \cdot P(A_1) \cdots P(A_k) \\ &\le \hat{q}\varepsilon^k \le \varepsilon\hat{q}. \end{aligned} \tag{4}$$

since $k \ge 1$. Hence (2) holds. Here, it is assumed that SteadyFilter and RollingSketch are independent of each other by using independent hash functions. □

Experimental analysis: We conduct experiments to validate Theorem 1. The main goal of this experiment is to test the effectiveness of SteadyFilter in filtering out unqualified flows, and the experimental design and results are as follows (k is set to 3 and CAIDA Dataset [28] is used): 1) Keep SteadyFilter to count the correct reporting of steady flows (Figure (6a)); 2) Remove SteadyFilter and count the correct reporting of steady flows (Figure (6b)); 3) Count the average proportion of 1 in SteadyFilter and FPR (Figure (6c)). The results show that the PR after removing SteadyFilter drops sharply, verifying that its filtering effect is significant. In other words, after removing SteadyFilter, it is no longer possible to detect flows with a frequency of 0 in a certain window.

Corollary 1: Assume that the N' flows that fail to arrive for p windows in a row are falsely reported by the RollingSketch to be steady flows. Then after adding the SteadyFilter, the number of flows N which will still be reported to be steady among these flows satisfies

$$EN \le \varepsilon N', \tag{5}$$

where EN is the expectation number of N.

B. Error Bound of RollingSketch

In this part, we assume that our RollingSketch uses 8-bit counters, and strawman solution uses 32-bit counters. Both algorithm use the same number of counters and use only one hash function (*i.e.* $w = 1$). We first analyze the error bound of the strawman solution, then we provide an error bound for RollingSketch. We first analyze the error bound of variance for strawman solution through Theorem 2. Then, we apply Lemma 3 and 4 to show that, under certain conditions, strawman solution and RollingSketch report the same variance. Finally, we come to the conclusion: the probability that the difference between the new and old variance is larger than an error bound is bounded. In other words, we save 75% memory without sacrificing the performance of RollingSketch too much.

Theorem 2: For an arbitrary flow e and p consecutive windows $t-p+1, \cdots, t$, let $Z_1, \cdots, Z_p$ denote its real frequency in each window, and $X_1, \cdots, X_p$ denote its frequency reported by strawman solution. Let $v, \hat{v}$ denote the real variance and the variance reported by strawman solution respectively. For an arbitrary small number ε, strawman solution guarantees

$$|\hat{v} - v| \le 2\varepsilon\bar{Z} + \varepsilon^2 \tag{6}$$

with probability at least $1 - \frac{Np}{\varepsilon\gamma}$, where N denotes the number of items in each window, γ denotes the number of counters in CM sketch in strawman solution, and $\bar{Z} = \frac{1}{p}(Z_1 + \cdots + Z_p)$.

Proof: Note that

$$\begin{aligned} \hat{v} - v = &\frac{1}{p}\sum_{i=1}^{p} X_i^2 - \frac{1}{p^2}\left(\sum_{i=1}^{p} X_i\right)^2 \\ &- \frac{1}{p}\sum_{i=1}^{p} Z_i^2 + \frac{1}{p^2}\left(\sum_{i=1}^{p} Z_i\right)^2 \\ = &\frac{1}{p}\sum_{i=1}^{p}(X_i + Z_i)(X_i - Z_i) \\ &- \frac{1}{p^2}\sum_{i=1}^{p}(X_i + Z_i)\sum_{i=1}^{n}(X_i - Z_i). \end{aligned}$$

Since strawman solution uses CM sketches, we have $Z_i \le X_i \le Z_i + \varepsilon$ with probability at least $1 - \frac{N}{\varepsilon\gamma}$ [26]. Applying union bound, strawman solution guarantees $Z_i \le X_i \le Z_i + \varepsilon, \forall 1 \le i \le p$ with probability at least $1 - \frac{Np}{\varepsilon\gamma}$. In this situation, we get

$$\begin{aligned} \hat{v} - v &\le \frac{1}{p}\sum_{i=1}^{p}(X_i + Z_i)(X_i - Z_i) \\ &\le \frac{1}{p}\sum_{i=1}^{p}(2Z_i + \varepsilon) \cdot \varepsilon = 2\varepsilon\bar{Z} + \varepsilon^2, \end{aligned}$$

and

$$\begin{aligned} \hat{v} - v &\ge -\frac{1}{p^2}\sum_{i=1}^{p}(X_i + Z_i)\sum_{i=1}^{p}(X_i - Z_i) \\ &\ge -\frac{1}{p^2} \cdot (2p\bar{Z} + p\varepsilon) \cdot p\varepsilon = -2\varepsilon\bar{Z} - \varepsilon^2. \end{aligned}$$

So Equation 6 holds. □

Lemma 3: For a flow e and an arbitrary timestamp t, let g_0, g_1 be its frequency reported by RollingSketch, f be its frequency reported by strawman solution, then $(f - g_0)\%256 = 0, |g_1 - g_0| = 128$.

Proof: This lemma is a simple property of rebirth. Note that rebirth happens only when $g_0 = 256$, and its value will change to 0 immediately, hence the two equality holds. □

Lemma 4: For a flow e and p consecutive windows $t-p+1, \cdots, t$, let $X_1, \cdots, X_p$ denotes the frequency of e in each window reported by strawman solution and

$$M = \max\{X_i : 1 \le i \le p\}, m = \min\{X_i : 1 \le i \le p\}.$$

If $M - m < 128$, then RollingSketch will report the same variance as strawman solution.

Proof: We resort $X_1, \cdots, X_p$ as $x_1, \cdots, x_p$, s.t. $x_1 \le \cdots \le x_p$. Since $M - m = x_p - x_1 < 128$, there exists some integer l, s.t. $0 \le x_1 - 128l \le x_2 - 128l \le \cdots \le x_k - 128l \le 127 < 128 \le x_{k+1} - 128l \le \cdots \le x_p - 128l \le 255$. During the calculation process, the sequence of numbers is kept unchanged, guaranteeing that the variance of the statistics are reflected correctly. Next we show that this value is actually the minimum among our calculation of variance. Note that for

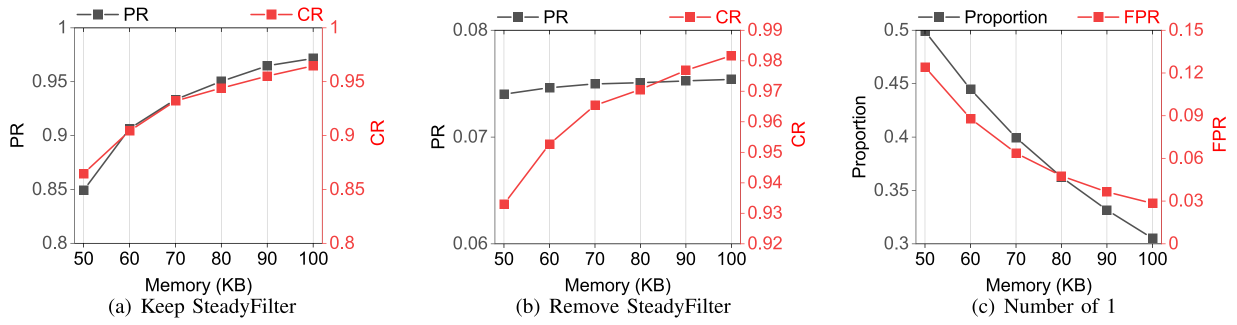


Fig. 6. Experimental analysis of steadyfilter for theorem 1.

two arbitrary integers x and l, $(x+256l)\%256 = x\%256$, hence the only way for a smaller variance is

$$\begin{cases} y_1 \triangleq (x_1+128)\%256 = x_1+128, \\ \cdots \\ y_k \triangleq (x_k+128)\%256 = x_k+128, \\ y_{k+1} \triangleq (x_{k+1}+128)\%256 = x_{k+1}-128, \\ \cdots \\ y_p \triangleq (x_p+128)\%256 = x_p-128. \end{cases}$$

If $k=0$ or $k=p$, then all x_i will increment or decrement 128, which makes the variance unchanged; otherwise, define $\bar{x} = \frac{1}{p}(x_1+\cdots+x_p)$, $u_i = x_i - \bar{x}$, and $s^2, \hat{s}^2$ be the variance of $x_1,\cdots,x_p; y_1,\cdots,y_p$. Then $u_1+\cdots+u_p=0$, and

$$s^2 = \frac{1}{p}\sum_{i=1}^{p}(x_i-\bar{x}) = \frac{1}{p}\sum_{i=1}^{p}u_i^2, \tag{7}$$

$$\begin{aligned} \hat{s}^2 &= \frac{1}{p}\sum_{i=1}^{p}(y_i-\bar{y})^2 \\ &= \frac{1}{p}\left(\sum_{i=1}^{k}(u_i+128)^2 + \sum_{j=k+1}^{p}(u_j-128)^2\right) \\ &\quad - \frac{1}{p^2}\left[128(2k-p)\right]^2 \\ &= \frac{1}{p}\left[\sum_{i=1}^{p}u_i^2 + 256\left(\sum_{i=1}^{k}u_i - \sum_{j=k+1}^{p}u_j\right) + 128^2 p \right. \\ &\quad \left. -128^2\frac{(2k-p)^2}{p}\right] \\ &= \frac{1}{p}\left[\sum_{i=1}^{p}u_i^2 + 512(u_1+\cdots+u_k) + \frac{128^2(4pk-4k^2)}{p}\right]. \end{aligned} \tag{8}$$

Hence

$$\begin{aligned} \hat{s}^2 - s^2 &= \frac{512}{p^2}\left[p(u_1+\cdots+u_k) + 128(kp-k^2)\right] \\ &= \frac{512}{p^2}\left(kp\bar{u}^- + 128kp - 128k^2\right), \end{aligned} \tag{9}$$

where $\bar{u}^-$ is defined as $\bar{u}^- = \frac{1}{k}(u_1+\cdots+u_k)$. Since $x_p - x_1 < 128$, then $u_p - u_1 < 128$. Hence for $i=k+1,\cdots,p$, $u_i \le u_p < 128 + u_1 \le 128 + \bar{u}^-$. Finally, we get

$$\begin{aligned} &\begin{cases} u_1+\cdots+u_k = k\bar{u}^-, \\ u_{k+1}+\cdots+u_p \le (p-k)(128+\bar{u}^-) \end{cases} \\ &\Rightarrow 0 = \sum_{i=0}^{p}u_i \le k\bar{u}^- + (p-k)(128+\bar{u}^-) \\ &\Rightarrow \bar{u}^- \ge \frac{128(k-p)}{p}. \end{aligned} \tag{10}$$

Applying this inequality, we get

$$\hat{s}^2 - s^2 \ge \frac{512}{p^2}\left[128k(k-p) + 128kp - 128k^2\right] = 0, \tag{11}$$

which shows that the variance reported by the RollingSketch is accurate. □

Corollary 2: For a flow e, let $\tilde{v}$ denote the variance reported by strawman solution. If $M - m \le 128$, then RollingSketch guarantees

$$|\tilde{v} - v| \le 2\varepsilon\bar{Z} + \varepsilon^2 \tag{12}$$

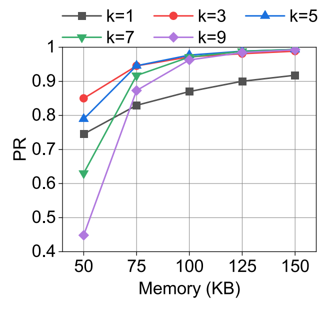
with probability at least $1 - \frac{Np}{\varepsilon\gamma}$.

Corollary 3: For a flow e, assume its frequency in each window is reported to be $X_1, X_2, \cdots$ by strawman solution. If RollingSketch wrongly reports $\langle e,t\rangle$ as steady flow but strawman solution does not, then $\exists t-p+1 \le i \ne j \le t$, s.t. $X_i - X_j \ge 128$.

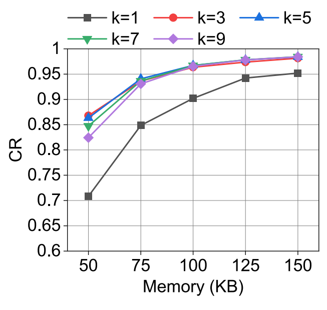
Since steady flows should occur a similar number of times in each window, we conclude that these flows cannot be steady flows.

V. Experimental Results

In this section, we present comprehensive experimental results on SteadySketch. First, we describe the experimental setup in Section V-A, including datasets and metrics, *etc.* Second, we explain how parameter settings affect SteadySketch's performance in Section V-B. Third, we evaluate SteadySketch's performance on finding transient and persistent steady flows on different datasets in Sections V-C and V-D, respectively, and compare it with the strawman solution. Finally, we provide three concrete applications of PISketch: cache prefetch, Redis and P4 implementation, in Sections V-E, V-F, and V-G, respectively.

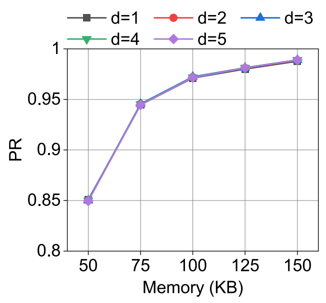


(a) PR of k

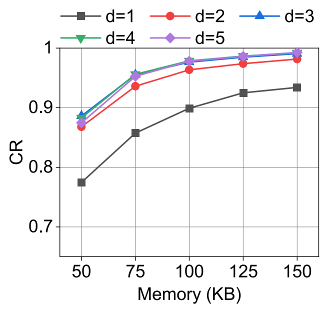


(b) CR of k

Fig. 7. Effect of the parameter: k.



(a) PR of d



(b) CR of d

Fig. 8. Effect of the parameter: d.

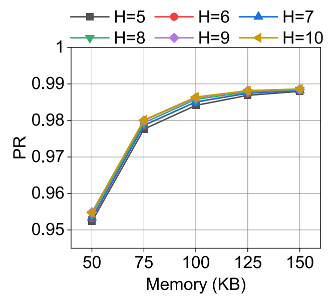
A. Experimental Setup

Datasets: We use a total of three real-world datasets and one synthetic dataset as follows.
1) CAIDA Dataset: This dataset is streams of anonymized IP traces collected by CAIDA [28]. In CAIDA Dataset, there are around 30M flows and 900K distinct flows.
2) Campus Dataset: This dataset is comprised of IP packets captured from the network of our campus. In Campus Dataset, there are 10M flows in total, belonging to 1M distinct flows.
3) MAWI Dataset: This dataset of real traffic traces is provided by the MAWI Working Group [45]. In MAWI Dataset, there are around 13M flows.
4) Synthetic Dataset: We generate a synthetic dataset that follows the Zipf [46] distribution using Web Polygraph [47], an opensource performance testing tool. In Synthetic Datasets, there are around 32M flows, and the skewness is 1.5.

We divide each of the datasets above into multiple time windows sized at around 10K flows.
Competitor algorithms: In fact, as long as each window size is set to 1, any data structure used to find persistent flows can also be used to replace the CM sketch for frequency estimation. Therefore, in addition to **strawman solution**, SteadySketch's competitors may also consist of schemes for finding persistent flows (On-off sketch [22], PIE [20], *etc.*). Here, we design an algorithm composed of several On-off sketches to find steady flows, called **Steady-On-Off**, in which the operations that it replaces the CM sketches are designed as follows. Its data structure is very similar to CM sketch, consisting of d arrays, each array has w buckets, but each bucket has two fields, one is a counter and the other is an On-off flag. For insertion operations, when one bucket in each array is hashed: if there is On in the bucket, the counter is incremented by 1 and the flag is switched to Off; if the bucket is Off, nothing is done. For query operations, output the smallest value among all d counters that have been hashed. When each window ends, switch all flags to On. The rest is consistent with strawman solution.
Implementation: We implement SteadySketch and the strawman solution in C++. The hash functions are implemented using the 32-bit Murmur Hash (obtained from the open-source website [48]) with different initial seeds.
Computation Platform: We conduct all the experiments on a machine with one 8-core processor (8 threads, Intel(R) Core(TM) i7-9700U CPU @ 3.00GHz) and 16 GB DRAM memory. The processor has 512KB L1 cache, 2MB L2 cache for each core, and 12MB L3 cache shared by all cores.
Metrics:
1) Precision Rate (PR): PR is the ratio of the number of correctly steady flows to the number of steady flows reported.
2) Recall Rate (CR): CR is the ratio of the number of correctly reported steady flows to the number of correctly steady flows.
3) Mean Squared Error (MSE): We define the MSE as $\frac{1}{n}\sum_{i=1}^{n}\left(\mathcal{V}_i-\hat{\mathcal{V}}_i\right)^2$, where $\mathcal{V}_i$ is the real variance of steady flow e_i, $\hat{\mathcal{V}}_i$ is the estimated variance of steady flow, and the n is the correct number of steady flows reported.
4) Average Relative Error (ARE): We define the ARE as $\frac{1}{|\Psi|}\sum_{e_i\in\Psi}\frac{|f_i-\hat{f}_i|}{f_i}$, where f_i is the real duration of persistent steady flow e_i, $\hat{f}_i$ is its estimated duration of persistent steady flow, and Ψ is the query set.
5) Throughput: We use Million of operations (insertions) per second (Mops) to measure the throughput. Experiments are repeated 10 times and the average throughput is reported.

B. Experiments on Parameter Settings

In this section, we measure the effects of some key parameters of SteadySketch, namely, the number of hash functions k in SteadyFilter, the number of hash functions d in RollingSketch, the the variance threshold H for the steady flows, the threshold p for time window period, and the ratio r of the memory size of SteadyFilter to the memory size of the whole SteadySketch. We use CAIDA Dataset in these experiments, and PR and CR to evaluate the effects.
1) Effect of k (Figure 7(a) - 7(b)): *The experimental results show that the best value for k is 3.* In this experiment, we fix the d to 2. When a flow is being inserted, more hash functions mean higher accuracy, but the number of counters per array will be reduced in fixed memory, which leads to accuracy reduction. Under the same memory, as k becomes larger, PR first increases and then decreases, while CR first increases and then reach a stable value, after which it does not change significantly. Considering the influence of space and hash number on accuracy, we set the parameter k to 3.
2) Effect of d (Figure 8(a) - 8(b)): *The experimental results show that the optimal value for d is 2.* In this experiment, we fix the k to 3. Under the same memory, as d becomes larger, CR first increases and then decreases, while PR does


(a) PR of H

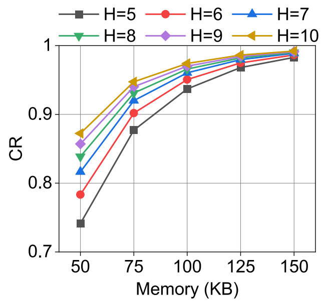
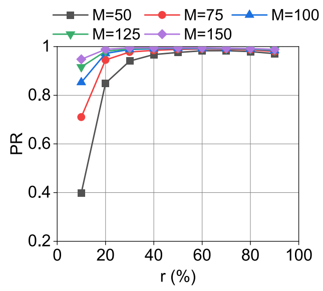

(b) CR of H

Fig. 9. Effect of the parameter: H.


(a) PR of r

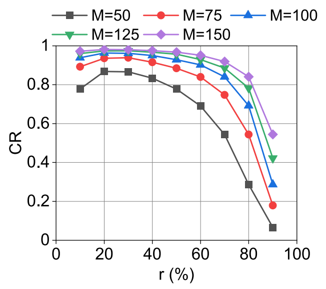
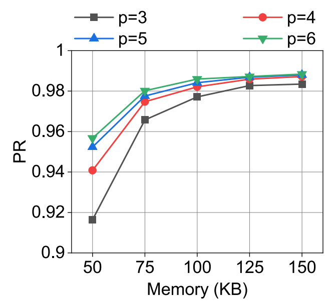

(b) CR of r

Fig. 11. Effect of the parameter: r.


(a) PR of p

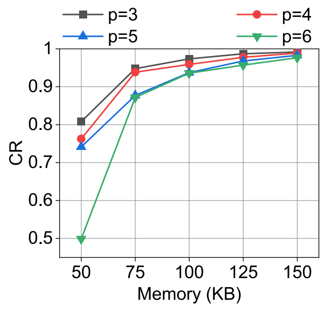

(b) CR of p

Fig. 10. Effect of the parameter: p.

not change significantly. However, more hash functions mean more memory access and thus the throughput is reduced. To better balance the CR and throughput, we set $d = 2$.

3) Effect of H (Figure 9(a) - 9(b)): *Our experimental results show that the PR and CR do not vary significantly under different H values.* In this experiment, we fix the k to 3 and d to 2. As H becomes larger, CR gradually increases but gradually approaches each other as the memory becomes larger, while PR does not change significantly. In the definition, smaller H represents higher requirements for stability and higher requirements for the accuracy of frequency estimation. Thus, the experimental results show that the larger H is, the higher accuracy we can achieve in a small memory. Since this value is set as small as possible, we finally choose $H = 5$.

4) Effect of p (Figure 10(a) - 10(b)): *The experimental results show that the optimal value for p is 5.* In this experiment, we fix k to 3, d to 2 and H to 5. As p increases, PR gradually increases and gradually approaches each other as the memory becomes larger, while CR gradually decreases and gradually approaches each other as the memory becomes larger. When the value of p is larger, the more counters in each slot of RollingSketch are needed. In limited memory, for each time window, there are less counters for frequency estimation when the p is larger, which may lead to the low accuracy. Therefore, in this paper, we set $p = 5$. In practical applications, it can be given any value that users find desirable.

5) Effect of r (Figure 11(a) - 11(b)): *The experimental results show that the optimal value for r is 20%.* In this experiment, we fix k to 3, d to 2, H to 5 and p to 5. M refers to the total memory of SteadySketch, which consists of the memory of SteadyFilter and RollingSketch. As r becomes larger, PR gradually increases, while CR first increases and then decreases. Therefore, we choose $r = 20\%$ because it can trade off PR and CR well for different values of M.

Analysis: The results show that SteadyFilter has a more significant impact on PR compared to that to CR, while the RollingSketch is opposite. When the memory of SteadyFilter is relatively small, it could not fully exert its filtering ability, which leads to a large number of interrupted flows misjudged, resulting in low precision rate. However, this situation doesn't affect CR much, since the steady flows are not identified as interrupted. RollingSketch, on the other hand, is used for frequency estimation, and it is overestimated. When the space occupied by RollingSketch is too small, there are only a few counters for frequency estimation, which leads to serious hash collisions. In this case, most steady flows will be identified as non-steady flows, which results in the low CR.

C. Finding Temporary Steady Flows

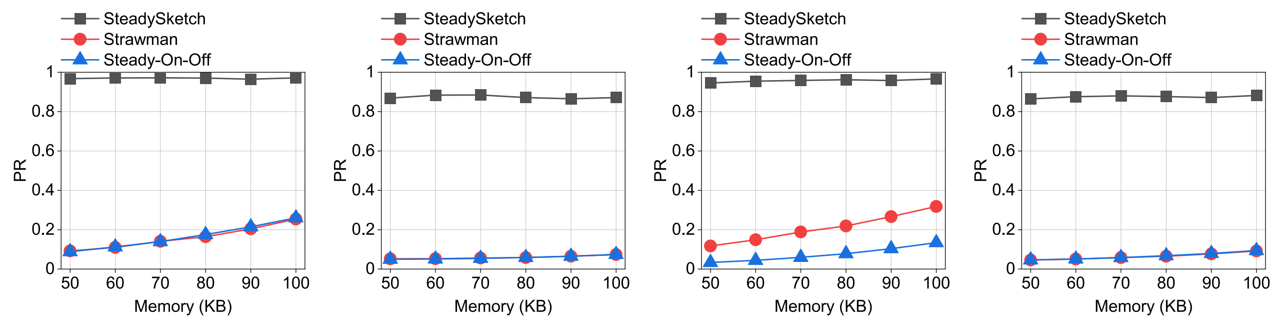
In this section, we compare the performance of SteadySketch in finding temporary steady flows with that of the strawman solution and Steady-On-Off in the metrics below.

1) PR (Figure 12(a) - 12(d)): *This experiment shows that the PR of SteadySketch is significantly better than the one of the strawman solution and Steady-On-Off.* On the four datasets, the PR of SteadySketch is around 79.5% and 82.8% higher than the one of the strawman solution and Steady-On-Off on average, respectively.

2) CR (Figure 13(a) - 13(d)): *This experiment shows that the CR of SteadySketch is much higher than the one of the strawman solution and Steady-On-Off.* On the four datasets, the CR of SteadySketch is around 26.1% and 21.5% higher than the one of the strawman solution and Steady-On-Off on average, respectively.

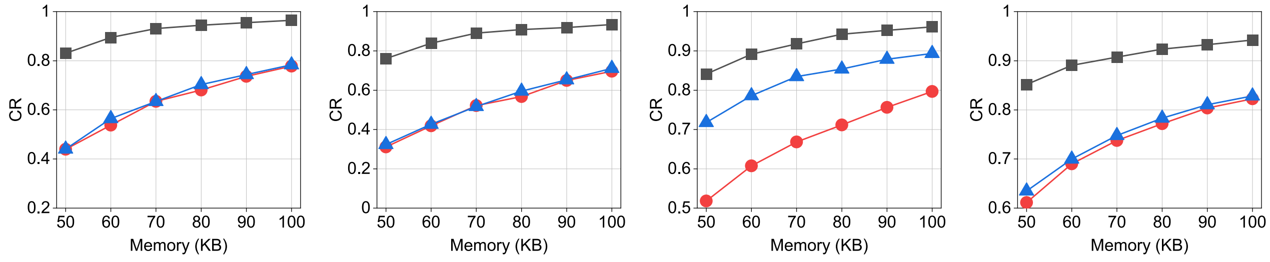
3) MSE (Figure 14(a) -14(d)): *This experiment shows that the MSE of SteadySketch is obviously lower than the one of the strawman solution and Steady-On-Off.* On the four datasets, the MSE of SteadySketch is around 4.3× and 4.1× lower than the one of the strawman solution and Steady-On-Off on average, respectively.

4) Throughput (Figure 15(a) - 15(d)): *This experiment shows that the throughput of SteadySketch is significantly higher than the one of the strawman solution and Steady-On-Off.* On the four datasets, the throughput of SteadySketch is around 1.6× and 14.7× higher than the one of the strawman solution and Steady-On-Off on average, respectively.



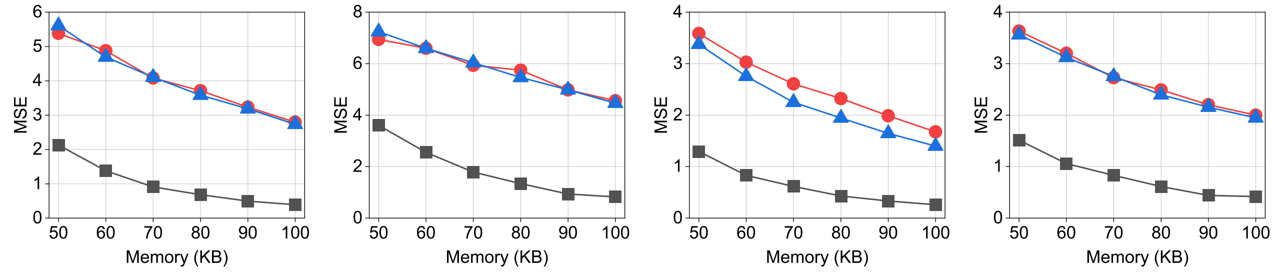
(a) CAIDA (b) Campus (c) MAWI (d) Synthetic

Fig. 12. Temporary steady flows: precision rate (PR) vs. memory.



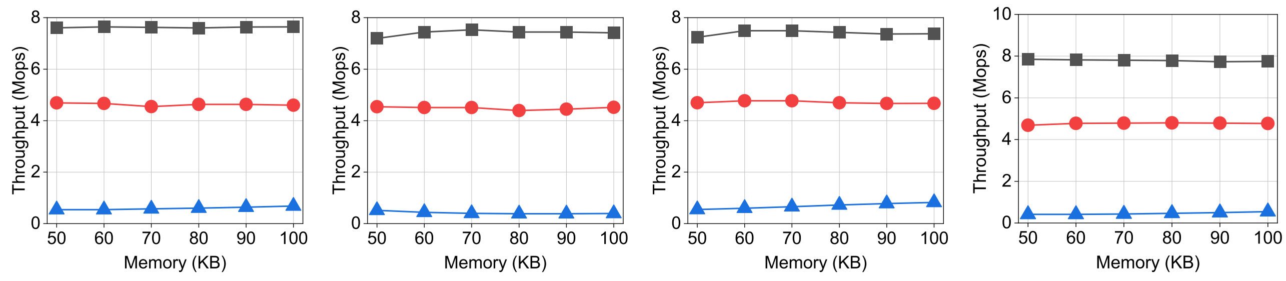
(a) CAIDA (b) Campus (c) MAWI (d) Synthetic

Fig. 13. Temporary steady flows: recall rate (CR) vs. memory.



(a) CAIDA (b) Campus (c) MAWI (d) Synthetic

Fig. 14. Temporary steady flows: mean squared error (MSE) vs. memory.



(a) CAIDA (b) Campus (c) MAWI (d) Synthetic

Fig. 15. Temporary steady flows: throughput vs. memory.

Analysis: Our experimental results show that SteadySketch achieves clearly higher accuracy and throughput. Compared with the strawman solution, SteadySketch reduce the size of the counter to one quarter, thereby increasing the number of counters by 4×. The reborn technique avoids the accuracy loss of variance estimation caused by the frequent flows. Thus, using the same memory space, the increased number of counters does not lead to loss in accuracy, which significantly reduces the MSE variance. In addition, compared with its great superiority over the strawman solution in terms of precision, the advantage of our solution in recall is less significant. By comparing the results reported by the strawman solution and SteadySketch, it is found that there are dozens of times more steady flows reported by the strawman

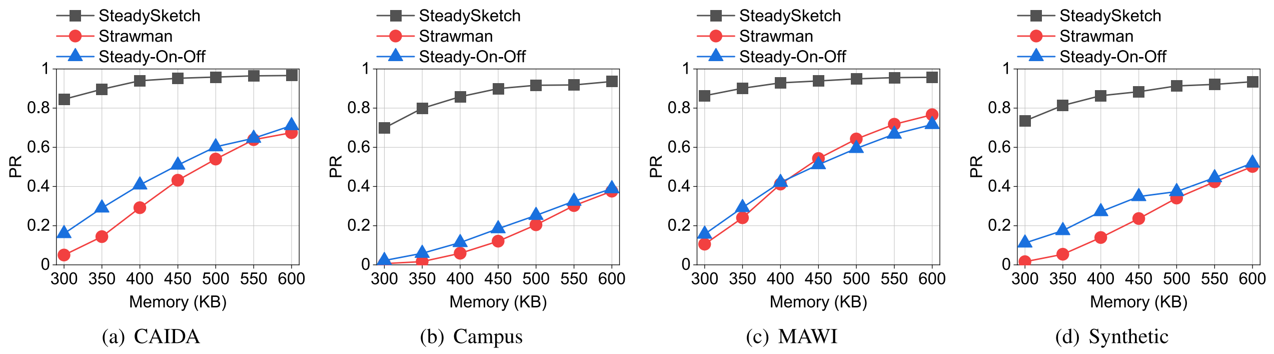


Fig. 16. Persistent steady flows: precision rate (PR) vs. memory.

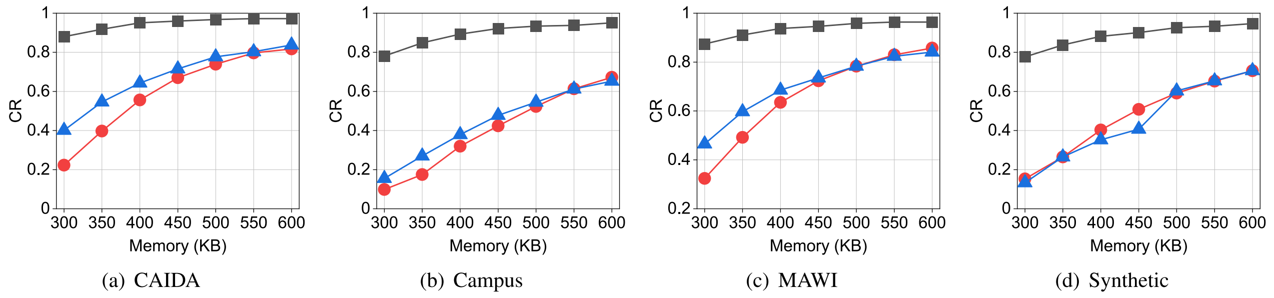


Fig. 17. Persistent steady flows: recall rate (PR) vs. memory.

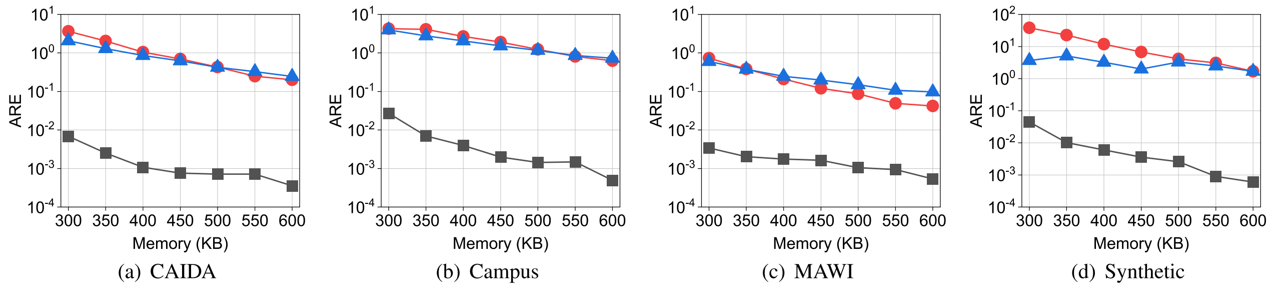


Fig. 18. Persistent steady flows: average relative error (ARE) vs. memory.

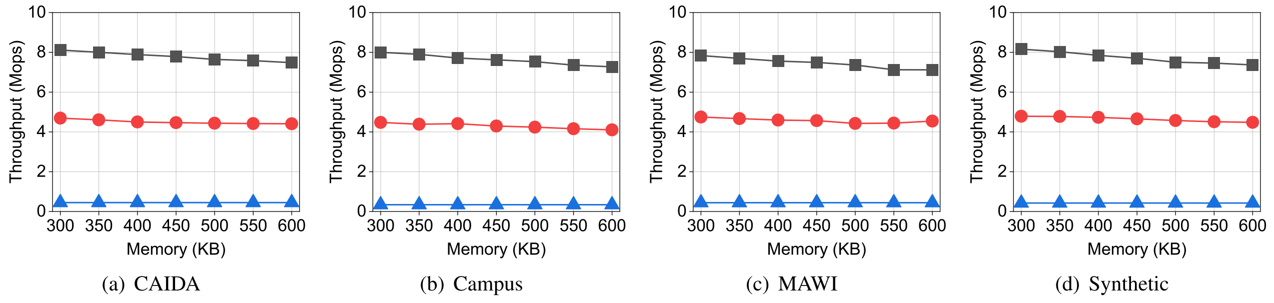


Fig. 19. Persistent steady flows: throughput vs. memory.

solution than the ground-truth that contains most of the steady flows. The reason for this is that there are too many small flows hash collision. As for Steady-On-Off, it has a extremely low throughput due to the fact that it treats each flow as a window for frequency estimation. Also, since its structure is similar to the CM sketch, its precision, recall, and MSE are also similar to the strawman solution.

D. Finding Persistent Steady Flows

In this section, we compare the performance of SteadySketch in finding persistent steady flows with that of the strawman solutions and Steady-On-Off in the metrics below. The total memory varies from 300KB to 600KB, including a fixed 150KB of Stage 2 memory.

5) PR (Figure 16(a)-16(d)): *This experiment shows that SteadySketch substantially outperforms the strawman solution and Steady-On-Off in PR.* On the four datasets, the PR of SteadySketch is around 57.6% and 53.0% higher than the one of the strawman solution and Steady-On-Off on average, respectively.

6) CR (Figure 17(a)-16(d)): *This experiment shows that SteadySketch significantly outperforms the strawman solution and Steady-On-Off in CR.* On the four datasets, the CR of SteadySketch is around 38.2% and 34.9% higher than the one of the strawman solution and Steady-On-Off on average, respectively.

7) ARE (Figure 18(a)-18(d)): *This experiment shows that the ARE of SteadySketch is obviously lower than the one of the strawman solution and Steady-On-Off.* On the four datasets, the ARE of SteadySketch is around 905.9× and 657.9× lower than the one of the strawman solution and Steady-On-Off on average, respectively.

8) Throughput (Figure 19(a)-19(d)): *This experiment shows that the throughput of SteadySketch is significantly higher than the one of the strawman solution and Steady-On-Off.* On the four datasets, the throughput of SteadySketch is around 1.7× and 18.7× higher than the one of the strawman solution and Steady-On-Off on average, respectively.

Analysis: The results show that our solution achieves better performance in accuracy, ARE, and throughput compared to the strawman solution and Steady-On-Off. Owing to the superior performance of Stage 1, the flows inserted into Stage 2 are in high accuracy. Compared with SteadySketch, both the strawman solution and Steady-On-Off report a large number of false positive flows in Stage 1, resulting in lower throughput and accuracy.

E. Cache Replacement Optimization

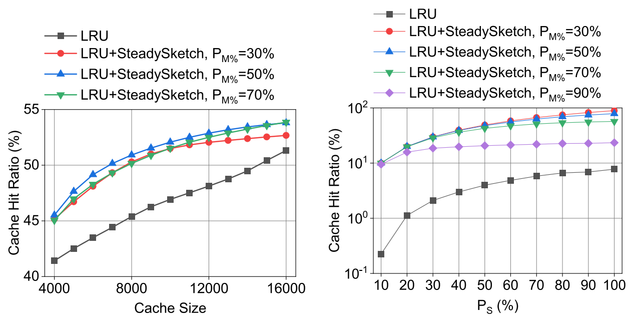
To the best of our knowledge, modern caches often adopt Least Recently Used (LRU) eviction strategy. In this section, however, we creatively introduce **SteadySketch** to ***predict*** the coming cache line and thus improve the cache hit ratio.[2] Our success depends on a reasonable hypothesis: once we consider a cache line is steady, it is probably fetched in the near future.

Experimental Setup: We use LRU as a comparison scheme of the cache replacement strategy, and conduct C++ simulation experiments. In our algorithm, we divide the cache into 3 parts: A SteadySketch (100KB, $k = 3$, $d = 2$, $H = 5$ and $p = 5$), a fully-associative ***steady part*** and a fully-associative ***general part***. Both the steady part and the general part can be seen as a small LRU cache. When we fetch a new cache line, we first insert its address into the SteadySketch to check if it is a steady cache line: A steady cache line will be fetched into the steady part and will not be evicted by unsteady ones; While an unsteady cache line will be fetched into the general part and will not be evicted by steady ones.

Experiments on Real-World Dataset: We conduct experiments on Campus dataset, and use $P_{M\%}$ to denote the case when the general part takes up $M\%$ of the memory of the cache. The experimental results are shown in Figure 20(a). We find that under a limited cache memory, using SteadySketch can significantly improve the cache hit ratio by up to 13.02% in a wide range of cache size. Therefore, our SteadySketch provides a new way to optimize cache replacement problem.

Experiments on Steady-Synthetic Dataset: Our Steady-Synthetic Dataset is a mixture of 10^7 steady and non-steady flows in total. The $P_S(\%)$ is the portion of steady flows in the dataset, while the other flows are random flows, most of

[2]The cache hit rate can be calculated as (the number of cache hits)/(the number of memory accesses).



(a) Campus (b) Steady-Synthetic

Fig. 20. The cache hit ratio comparison.

them only appear for a short time. In the Steady-Synthetic Dataset experiment, we fix the cache size as 12K and increase the ratio of steady flows in the dataset to observe the cache hit ratio of different algorithms. The experimental results are shown in Figure 20(b). SteadySketch can significantly improve the cache hit ratio compared with typical LRU, especially when the $P_{M\%}$ is relatively small. It means that the larger the proportion of SteadySketch is, the more significant the effect will be, when the P_S is set to 30%, the accuracy increased by an average of 44×.

F. Redis Implementation

Redis is an open-source, in-memory data structure store that has been widely used as a database, cache, message broker, and streaming engine. Therefore, if SteadySketch can be implemented on Redis, it can not only be used to find steady flows, but also lay the foundation for a potential application: persistence of Redis. This application means that the Redis Database files need to save the entire dataset to disk within a fixed time interval. In case of Redis stopping working without a correct shutdown, we will lose a lot of data in the last few minutes. Generally, the data with steady access is much more important. Thus, as a future work, we can consider preferentially storing these flows at smaller intervals to avoid loss of important data. Next, let's give our attention to SteadySketch's Redis implementation.

Challenges: Although Redis provides the API that allows us to customize commands, it still brings us to the following challenges. 1) The API provided by Redis has rich functions, but it is very miscellaneous and user-unfriendly. 2) Redis is written with ANSI C, so many data structures and functions in C++ cannot be realized. 3) Memory allocation must be completed through Redis Module API, so it is not recommended to dynamically allocate memory with new or malloc function. Therefore, many functions in STL cannot be used directly, such as set, map and string.

Implement on Redis: In this paper, we implement our SteadySketch and strawman solution on Redis system. Each algorithm aims at two data characteristics: temporary and persistent steady flows. Thus, we have added four data structures to Redis, and each data structure has created many commands such as create, insert, clear, info, *etc.*

We conduct the experiment with the metric of throughput on the datasets of CAIDA and Campus. The experimental results are shown in Figure 21, the T-SteadySketch is for

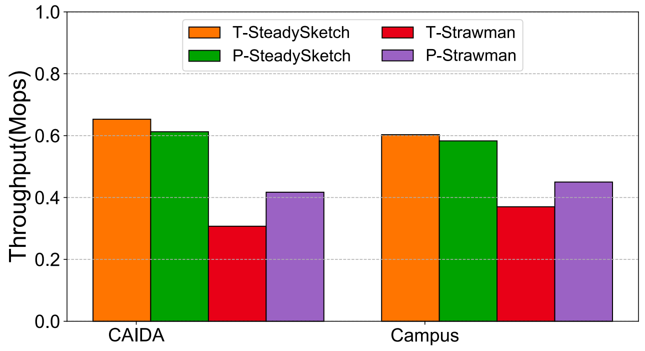


Fig. 21. The throughput comparison on redis.

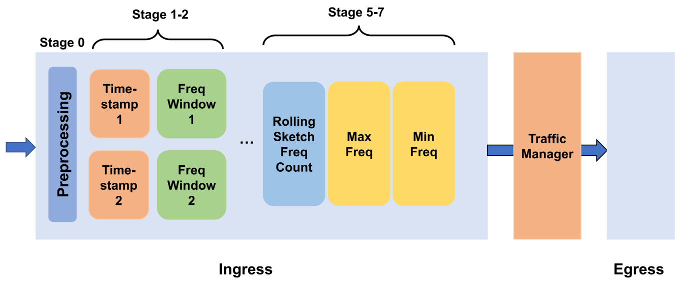


Fig. 22. The logic of P4 implementation.

finding temporary steady flows, while the P-SteadySketch is for persistent steady flows. The same is true for T-Strawman and P-Strawman.

Analysis: In Figure 21, SteadySketch shows superior performance over the strawman solution, especially in finding temporary steady flows. For SteadySketch, in the two tasks, the throughput of finding persistent steady flows is slightly lower than that of temporary steady flows, which is caused by the insertion of Stage 2. On the contrary, the throughput of finding persistent steady flows with strawman solution is much higher. This is because the task of finding temporary steady flows could report a lot of flows in each time window, which results in frequent calls of Redis output.

G. P4 Implementation

We have fully built a P4 prototype of the SteadySketch on the Tofino switch [49] using P4 language [50]. In the P4 version implementation of SteadySketch, we combine a two-layer SteadyFilter with an one-layer RollingSketch. Under the hardware restriction, we use the difference between the maximum and the minimum instead of the variance.

Challenge: In Tofino architecture, packets go through the ingress pipeline, the traffic manager and the egress pipeline in turn. The ingress and egress pipeline each contains 12 separate stages, each capable of handling some simple logic operations and having independent memory. Due to the speed requirements of the switch, packets can only visit each stage once and spend ultrashort time, which results in the simple operational logic in each stage, such as addition, subtraction and shift operation. How to measure the stability of the flow becomes the biggest challenge in the P4 implementation. To address the above problem, we use an additional stage to record the maximum frequency and the minimum frequency of a flow, then calculate the difference between them, which also plays a role in measuring the flow stability.

TABLE II
H/W Resources Used by SteadySketch

Resource	Usage	Percentage
Hash Bits	233	5.09%
SRAM	41	4.66%
Map RAM	41	7.77%
TCAM	0	0%
Stateful ALU	0	0%
VLIW Instr	19	5.40%
Match Xbar	44	3.12%

Design: As shown in Figure 22, we use a total of 8 stages in Ingress to implement the P4 version of SteadySketch under the Tofino model. Specifically, Stage 1 and Stage 2 are SteadyFilter, Stage 5 to 7 are RollingSketch and some simple calculation units scatter in other stages. In SteadyFilter part, we build two parallel filters, each containing a bucket array of timestamp and a bucket array of frequency window. As a packet passes through, the index of the bucket array is calculated with the built-in hash function. The packet compares its timestamp with the corresponding timestamp bucket, then adds the frequency bucket or sets the frequency value to one, which is same as the SteadyFilter algorithm we mentioned earlier. After passing the filter part, the packet entered RollingSketch part, which consists of a bucket array of the frequency counter, a bucket of array of the max frequency and a bucket array of the min frequency. The index of the buckets can also be calculated with the built-in hash function. The frequency counter records the total number of packets passed within a timestamp interval. If the first packet within an interval passes through, it sets the frequency bucket corresponding to the hash index to one, and brings the old frequency value to update the maximum bucket and the minimum bucket in next two stages. Otherwise, the value of the frequency bucket is simply incremented by one. The packet reads the maximum value and the minimum value when visits the two bucket. If the frequency window value is bigger than the threshold and the difference between the minimum frequency and the maximum frequency is bigger than the threshold, just like the algorithm we mentioned, the packet is from a steady flow and we set the flag bit to one in the header.

Evaluation results: We list the utilization of various hardware resources on the switch in Table II. We find the simplified logic uses few resources of the Tofino switch. Thanks to this, the usage rate of all hardware resources is lower than 10%. For example, Map RAM and VLIW Instr, which consume the most resources, account for 7.77% and 5.40% of the total quota, respectively. The evaluation results show that the P4 version of SteadySketch is lightweight and easy to deploy.

VI. Conclusion

In many applications, it is important to find steady flows in high speed data streams in real time. In this paper, we propose a novel algorithm called SteadySketch for real-time steady flow detection, which is fast, memory-efficient and accurate. Experimental results show that SteadySketch can achieve about 79.5% and 82.8% higher PR, 1.7× and 18.7× higher throughput, and 905.9× and 657.9× lower ARE than the two comparison schemes, respectively. Finally, we implement SteadySketch on three concrete cases: cache replacement, Redis, and P4 implementation. Cache replacement experiments show that SteadySketch can significantly improve the cache hit ratio. In Redis experiments, we add new functions and new data structures to Redis, supporting steady flow detection and querying in data streams. Additionally, we verify that the P4 version of SteadySketch is lightweight and easy to deploy.

REFERENCES

[1] X. Li et al., "SteadySketch: Finding steady flows in data streams," in *Proc. IWQoS*, 2023, pp. 1–9.

[2] L. Li et al., "A measurement study on multi-path TCP with multiple cellular carriers on high speed rails," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 161–175.

[3] L. Li, K. Xu, D. Wang, C. Peng, Q. Xiao, and R. Mijumbi, "A measurement study on TCP behaviors in HSPA+ networks on high-speed rails," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 2731–2739.

[4] J. Fan, X. Ming-wei, C. Yong, and X. Ke, "Real-time measurement of local QoS states," in *Proc. TENCON*, vol. 100, 2004, pp. 48–51.

[5] M. Shen, J. Zhang, K. Xu, L. Zhu, J. Liu, and X. Du, "DeepQoE: Real-time measurement of video QoE from encrypted traffic with deep learning," in *Proc. IWQoS*, 2020, pp. 1–10.

[6] Y. Lei, L. Yu, V. Liu, and M. Xu, "PrintQueue: Performance diagnosis via queue measurement in the data plane," in *Proc. ACM SIGCOMM Conf.*, Aug. 2022, pp. 516–529.

[7] M. Wyss, G. Giuliari, M. Legner, and A. Perrig, "Secure and scalable QoS for critical applications," in *Proc. IEEE/ACM 29th Int. Symp. Quality Service (IWQOS)*, Jun. 2021, pp. 1–10.

[8] M. Chen, M. Xu, Q. Li, and Y. Yang, "Measurement of large-scale BGP events: Definition, detection, and analysis," *Comput. Netw.*, vol. 110, pp. 31–45, Dec. 2016.

[9] M. Chen, M. Xu, Y. Yang, and Q. Li, "A measurement study on the distribution disparity of BGP instabilities," in *Proc. IEEE 41st Conf. Local Comput. Netw. (LCN)*, Nov. 2016, pp. 19–27.

[10] Q. Huang and P. P. C. Lee, "LD-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 1420–1428.

[11] Z. Zhong, S. Yan, Z. Li, D. Tan, T. Yang, and B. Cui, "BurstSketch: Finding bursts in data streams," in *Proc. Int. Conf. Manage. Data*, Jun. 2021, pp. 2375–2383.

[12] Z. Fan et al., "PeriodicSketch: Finding periodic items in data streams," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, May 2022, pp. 96–109.

[13] J. Liu et al., "DUET: A generic framework for finding special quadratic elements in data streams," in *Proc. WWW*, 2022, pp. 2989–2997.

[14] C. Hua and T.-S.-P. Yum, "Optimal routing and data aggregation for maximizing lifetime of wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 892–903, Aug. 2008.

[15] D. Ayyagari and A. Ephremides, "Admission control with priorities: Approaches for multi-rate wireless systems," *Mobile Netw. Appl.*, vol. 4, pp. 209–218, Sep. 1999.

[16] N. Afrin, J. Brown, and J. Y. Khan, "Design of a buffer and channel adaptive LTE semi-persistent scheduler for M2M communications," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5821–5826.

[17] H. Gomaa, G. G. Messier, C. Williamson, and R. Davies, "Estimating instantaneous cache hit ratio using Markov chain analysis," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1472–1483, Oct. 2013.

[18] Z. Fan et al., "Finding simplex items in data streams," in *Proc. IEEE 39th Int. Conf. Data Eng. (ICDE)*, Apr. 2023, pp. 1953–1966.

[19] B. Lahiri, S. Tirthapura, and J. Chandrashekar, "Space-efficient tracking of persistent items in a massive data stream," *ASA Data Sci. J.*, vol. 7, no. 1, pp. 70–92, 2014.

[20] H. Dai, M. Shahzad, A. X. Liu, M. Li, Y. Zhong, and G. Chen, "Identifying and estimating persistent items in data streams," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2429–2442, Dec. 2018.

[21] H. Dai, M. Li, A. X. Liu, J. Zheng, and G. Chen, "Finding persistent items in distributed datasets," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 1–14, Feb. 2020.

[22] Y. Zhang et al., "On-off sketch: A fast and accurate sketch on persistence," *Proc. VLDB Endowment*, vol. 14, no. 2, pp. 128–140, Oct. 2020.

[23] H. Huang et al., "You can drop but you can't hide: K-persistent spread estimation in high-speed networks," in *Proc. INFOCOM*, 2018, pp. 1889–1897.

[24] H. Huang et al., "An efficient K-persistent spread estimator for traffic measurement in high-speed networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1463–1476, Aug. 2020.

[25] Y.-E. Sun, H. Huang, S. Chen, Y. Zhou, K. Han, and W. Yang, "Privacy-preserving estimation of k-persistent traffic in vehicular cyber-physical systems," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8296–8309, Oct. 2019.

[26] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[27] S. Shahriar, *Algebra in Action: A Course in Groups, Rings, and Fields*, vol. 27. Providence, RI, USA: American Mathematical Society, 2017.

[28] (2018). *The CAIDA Anonymized Internet Traces*. [Online]. Available: http://www.caida.org/data/overview/

[29] (2024). *The Source Code of SteadySketch*. [Online]. Available: https://github.com/pkufzc/SteadySketch_ToN

[30] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[31] O. Rottenstreich, Y. Kanizo, and I. Keslassy, "The variable-increment counting Bloom filter," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1092–1105, Aug. 2014.

[32] J. Lu et al., "Low computational cost Bloom filters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2254–2267, Oct. 2018.

[33] Y. Wu et al., "Elastic Bloom filter: Deletable and expandable filter using elastic fingerprints," *IEEE Trans. Comput.*, vol. 71, no. 4, pp. 984–991, Apr. 2022.

[34] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, Aug. 2003.

[35] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theor. Comput. Sci.*, vol. 312, no. 1, pp. 3–15, 2004.

[36] T. Yang et al., "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 561–575.

[37] Y. Zhang et al., "CocoSketch: High-performance sketch-based measurement over arbitrary partial key query," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 207–222.

[38] Q. Huang et al., "SketchVisor: Robust network measurement for software packet processing," in *Proc. SIGCOMM*, 2017, pp. 113–126.

[39] L. Tang, Q. Huang, and P. P. C. Lee, "A fast and compact invertible sketch for network-wide heavy flow detection," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2350–2363, Oct. 2020.

[40] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proc. ICDT*, 2005, pp. 398–412.

[41] D. Ting, "Data sketches for disaggregated subset sum and frequent item estimation," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 1129–1140.

[42] T. Yang et al., "Adaptive measurements using one elastic sketch," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2236–2251, Dec. 2019.

[43] T. Yang et al., "HeavyKeeper: An accurate algorithm for finding top-k elephant flows," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 1845–1858, Oct. 2019.

[44] Y. Li et al., "LadderFilter: Filtering infrequent items with small memory and time overhead," in *Proc. SIGMOD*, 2023, pp. 1–21.

[45] (2010). *MAWI Working Group Traffic Archive*. [Online]. Available: http://mawi.wide.ad.jp/mawi/

[46] D. M. Powers, "Applications and explanations of Zipf's law," in *Proc. NeMLaP3/CoNLL*, 1998, pp. 151–160.

[47] A. Rousskov and D. Wessels, "High-performance benchmarking with web polygraph," *Softw., Pract. Exper.*, vol. 34, no. 2, pp. 187–211, Feb. 2004.

[48] (2016). *The Source Code of Murmur Hash*. [Online]. Available: https://github.com/aappleby/smhasher

[49] (2022). *Barefoot Tofino: World's Fastest P4-Programmable Ethernet Switch Asics*. [Online]. Available: https://barefootnetworks.com/products/brief-tofino/

[50] (2020). *P4-16 Language Specification*. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.1.html#sec-checksums

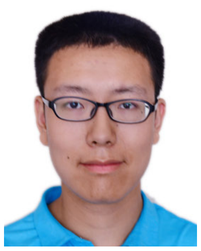
Zhuochen Fan received the Ph.D. degree in computer science from Peking University in 2023, advised by Tong Yang. He is currently a Boya Post-Doctoral Fellow with the School of Computer Science, Peking University. He had articles published by IEEE/ACM TRANSACTIONS ON NETWORKING, *VLDB Journal*, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, ICDE, RTSS, ICPP, and ICNP, etc. His research interests include approximation algorithms in computer networks and databases.

Xiangyuan Wang is currently pursuing the bachelor's degree with Peking University, majoring in information and computing sciences, advised by Tong Yang. His research include data structures and algorithms in network measurements. He is also interested in machine learning.

Xiaodong Li received the B.E. degree in IoT engineering from the University of Science and Technology Beijing in 2019 and the M.S. degree from Peking University in 2023. His research interests mainly focus on data stream processing and programmable switches.

Jiarui Guo (Graduate Student Member, IEEE) received the B.S. degree in computer science from Peking University in 2023, where he is currently pursuing the Ph.D. degree in computer science, advised by Tong Yang. His research interests include approximation algorithms in data streams and computer network systems.

Wenrui Liu received the B.S. degree in computer science from Peking University in 2023, where he is currently pursuing the M.S. degree in computer science, advised by Tong Yang. His research interests include network measurements, programmable switches, and network systems.



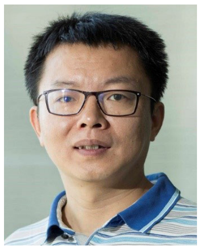
Haoyu Li received the B.S. degree in computer science from Peking University in 2023, advised by Tong Yang. He is currently pursuing the Ph.D. degree in computer science with The University of Texas at Austin. His research interests include computer networking, systems, and databases.

Sheng Long received the B.S. degree from the Department of Electrical Engineering and Computer Science, Peking University, in 2022, advised by Tong Yang. His research interests include network measurements, sketches, and KV stores.

Zheng Zhong received the B.S. degree from the Department of Electrical Engineering and Computer Science, Peking University, in 2023, advised by Tong Yang. He is currently pursuing the M.S. degree with Guanghua School of Management, Peking University. He has published articles in SIGMOD and SIGKDD. His research interests include streaming processing, bloom filters, and data structures.

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

Xuebin Chen received the Ph.D. degree from the Hebei University of Technology. He is currently a professor of North China University of Science and Technology, the Director of Tangshan Key Laboratory of Data Science, and the Director of Hebei Key Laboratory of Data Science and Application. He is a Council Member of China Computer Federation (CCF), Secretary General CCF Computer Application Technical Committee, Member of CCF High Performance Computing Technical Committee, and Member of CCF Big Data Expert Committee. His research interests include big data, network security, and data security. He presided and participated more than 30 horizontal projects of national, provincial and municipal levels. He published more than 50 academic papers, registered more than 40 software copyrights, and gained 3 Science and Technology Progress Awards.



Bin Cui (Fellow, IEEE) received the Ph.D. degree from the National University of Singapore in 2004. He is currently a Professor and the Vice Dean of the School of CS, Peking University. His research interests include database system architectures, query and index techniques, big data management, and mining. He was awarded the Microsoft Young Professorship Award (MSRA 2008), the CCF Young Scientist Award (2009), and the Second Prize of Natural Science Award of MOE China (2014), and was appointed as a Cheung Kong Distinguished Professor by MOE in 2016. He is serving as the Vice Chair for Technical Committee on Database China Computer Federation (CCF) and a Trustee Board Member for VLDB Endowment. He is also on the Editorial Board of *Distributed and Parallel Databases*, *Journal of Computer Science and Technology*, and *Science China Information Sciences*, and was an Associate Editor of IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING and *The VLDB Journal*.