PBSketch: Finding Periodic Burst Items in Data Streams

Zhuochen Fan[*]
Pengcheng Laboratory
Shenzhen, China
fanzc@pku.edu.cn

Zhongxian Liang[*]
Harbin Institute of Technology
Shenzhen, China
24b951067@stu.hit.edu.cn

Zirui Liu[*]
Peking University
Beijing, China
zirui.liu@pku.edu.cn

Dayu Wang
Nanyang Technological University
Singapore, Singapore
dayu001@e.ntu.edu.sg

Dong Wen
National University of Defense Technology, Changsha, China
wendong19@nudt.edu.cn

Wenjun Li[†]
Pengcheng Laboratory
Shenzhen, China
wenjunli@pku.org.cn

Tong Yang
Peking University
Beijing, China
yangtong@pku.edu.cn

Yuzhou Liu
Jilin University
Changchun, China
liuyuzhou@jlu.edu.cn

Weizhe Zhang
Harbin Institute of Technology
Shenzhen, China
wzzhang@hit.edu.cn

Abstract

Detecting periodic burst (PB) items in data streams is crucial for applications like rate limiting but remains unexplored. While combining existing sketch algorithms offers a baseline, it suffers from significant inaccuracy and inefficiency. In this paper, we propose PBSketch, the first dedicated sketch algorithm designed for detecting PB items in real time. Its key techniques mainly include: 1) a two-stage hierarchical structure that efficiently maintains potential burst items and discards those without potential; 2) a fine-grained PB selection mechanism during window processing, coupled with the Window Smoothing Processing optimization to amortize performance overhead and eliminate processing spikes. We provide its error bounds through rigorous theoretical analysis. Our extensive experiments show that PBSketch outperforms the baseline solution in accuracy and speed. By deploying it on an FPGA platform, the throughput is further significantly improved. Moreover, it effectively optimizes a practical application of rate limiting, clearly improving performance with almost negligible overhead.

CCS Concepts

• **Information systems → Data stream mining**; • **Networks → Network measurement**.

Keywords

Data Streams; Sketch; Data Structure; Algorithm; Rate Limiting

ACM Reference Format:
Zhuochen Fan, Zhongxian Liang, Zirui Liu, Dayu Wang, Dong Wen, Wenjun Li, Tong Yang, Yuzhou Liu, and Weizhe Zhang. 2026. PBSketch: Finding Periodic Burst Items in Data Streams. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '26), August 09–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3770854.3780188

[*]Co-first authors. Zhongxian Liang is also with Pengcheng Laboratory.
[†]Corresponding author: Wenjun Li (Institutional email: liwj@pcl.ac.cn).

Resource Availability:
The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.17730917.

1 Introduction

1.1 Background and Motivations

Most data exist widely in the form of data streams. It has always been a challenge to accurately extract the required information from massive data streams. As probabilistic algorithms, sketches [1–4] are fast and memory-saving. They always obtain target information in real time with only a small sacrifice of accuracy and have been widely recognized by the research community in addressing the challenge. Among the diverse tasks of data stream processing using sketches, targets include not only the traditionally well-studied frequent items (including heavy hitters) [5–12] and persistent items [13–17], but also burst items [18–20], periodic items [21] and batches [22], PI items [23, 24], simplex items [25], steady items [26, 27], and quadratic items [28], *etc.* that play crucial roles in different application scenarios but remain understudied.

In this paper, we introduce and study periodic burst (PB) items, a novel concept in the realm of data streams. PB items represent a unique combination of periodic and burst items, where burst items occur at fixed intervals. Despite their significance, PB items have not been explored in prior research. However, they hold great promise in many applications. For instance, in rate limiting, the predictability of PB items enables proactive resource allocation to improve QoS [29]; In LLM training, detecting periodic synchronization bursts [30, 31] helps identify stragglers and prevent costly halts. PB items also have important potential roles in intrusion detection [32–35], IoT data management [36], and network failure localization [37–40].

A baseline solution for finding PB items is to combine existing algorithms such as BurstSketch [20] and PeriodicSketch [21], which are state-of-the-art (SOTA) sketches for detecting burst items and periodic items, respectively. However, such a baseline solution is far from optimal in both accuracy and efficiency due to poor cooperation and redundant components. This highlights the need for a purpose-built solution to accurately and efficiently detect PB items.

1.2 Our Proposed Solution

Towards the design goal, we propose a novel sketch algorithm, called **PBSketch**, to find PB items in data streams in real time for the first time. PBSketch is compact, requiring only 20KB of memory overhead when processing 35M items; PBSketch is accurate, with its F1 Score, Average Absolute Error (AAE), and Average Relative Error (ARE) improved by up to 60.0%, 22.9×, 30.1× compared to the baseline solution in finding PB items, respectively; PBSketch is fast, with its throughput 1.43× faster than that of the baseline solution.

Compared with the four components of the above-mentioned baseline solution, PBSketch consists of only two interconnected/-collaborative components: Part 1 dynamically maintains potential burst items with high frequency, tracks those compliant burst items and calculates their periodicity; Part 2 maintains those periodic items with more periodic occurrences as PB items based on the burst item information reported by Part 1. It is easy to see that the key innovations of PBSketch are reflected mainly in the design of Part 1, whose design philosophy is shown below.

The basic data structure of Part 1 is a customized hot-cold sketch with two arrays, which is also one of the key techniques of PBSketch. Each array has the same number of hot-cold buckets, and the structure of each bucket is designed to have a unique layout of several variable-length cells: Only one large-size hot cell is used to record the complete information required for potential burst items, while other cold cells of gradually smaller sizes only record basic information such as the frequency of non-burst active items. Since items with high frequency have more potential to become burst items, the cells in the bucket are sorted dynamically in real time so that the item most likely to become a burst item is always in the hot cell. Of course, if the potential burst item in the current hot cell is in its burst active period, it will be properly protected and not participate in the sorting: As long as it becomes an eligible burst item during the retention period, we calculate the burst period and pass its related information to Part 2. Once there are two potential burst items in a bucket, the new one will avoid conflict by being rehashed into the bucket of another array and eliminating the least active item at the bottom. In summary, the above technique exploits the skewness of data streams to select eligible active burst items in a memory-efficient manner. While centrally processing the potential burst item information before each cross-time window (to select the eligible ones) may cause unstable performance. Thus, we propose Window Smoothing Processing (WSP) optimization for PBSketch to smooth throughput. Each bucket only needs a minimal additional overhead to avoid repeated access, so that the burst compliance is judged immediately after the item insertion process of the current window is completed. Thanks to the techniques from Part 1, Part 2 only needs to use a probability equation to dynamically select the ones with more pronounced periodicity as PB items. See § 3 for more details.

Further, we provide error bounds and time complexity of PBSketch through rigorous mathematical analysis in § 4. Finally, we conduct extensive experiments in § 5, as follows. 1) We compare PBSketch with the baseline solution in terms of accuracy and speed on two real-world datasets in § 5.3 and § 5.4, respectively, and the results fully demonstrate its clear advantages in both aspects. 2) We implement PBSketch on an FPGA platform with a throughput of 223.2 Mops in § 5.5. 3) We also implement the rate limiting optimization mentioned in § 1.1, *i.e.*, we apply PBSketch to optimize two typical rate limiting algorithms: counter-based and leaky bucket-based rate limiting algorithms in § 5.6. Experimental results show that PBSketch only requires negligible memory overhead to significantly reduce the number of rejections of the above two algorithms by about 14.1% and 18.6%, respectively.

Our Key Contributions:

- We propose a new problem called finding PB items for the first time, which is important in many big data-related application scenarios but has never been studied.
- We propose a novel sketch algorithm, namely PBSketch, which can accurately find PB items in data streams with only a small memory overhead.
- We provide theoretical guarantees for PBSketch through rigorous mathematical analysis.
- We conduct extensive experiments, and the results show not only the great advantages of PBSketch over the baseline solution, but also verify that PBSketch has deployment flexibility and can effectively optimize rate limiting.

2 Related Work

2.1 BurstSketch

BurstSketch [20] is the SOTA sketch algorithm for finding burst items in data streams. It defines a burst item for the first time in such a way that an item must satisfy both the burst start (sudden increase) and the burst end (sudden decrease), as shown below. Initially, a given data stream needs to be divided into many fixed-length time windows. For any item, assuming that its frequency in the $(i+1)$-th window is more than $k\times$ that of the i-th window (burst start), and its frequency in the $(j+1)$-th window is less than $\frac{1}{k}\times$ that of the j-th window (burst end), where $j-i$ is less than a maximum length L and the frequencies of windows from $i+1$ to j exceed a burst threshold H, then the item is a burst item. BurstSketch consists of two stages. In the first stage, infrequent items with a frequency lower than H' $(H' < H)$ are eliminated because they will not become burst items and account for the majority, and the remaining potential burst items are transferred to the second stage. The second stage records the information related to the sudden increase and decrease of the frequencies of potential burst items and reports the burst items that meet the definition mentioned.

2.2 PeriodicSketch

PeriodicSketch [21] is the first and SOTA sketch algorithm for finding periodic items in data streams. It defines periodic items as follows. For any item, assuming that t_i and t_{i+1} are its i-th and $(i+1)$-th occurrences on the time axis, respectively, so the i-th time interval can be calculated as $t_{i+1}-t_i$, then the item is a periodic item when all its time intervals have values around $t_{i+1}-t_i$ and the total number of such intervals is the K largest. PeriodicSketch also consists of two stages. The first stage records and reports the time interval of each item in real time. Before entering the second stage, any item ID/key is bound to its time interval and treated as a new item. The second stage records the interval frequency of the item and keeps those items with high interval frequencies as the final reported periodic items.

2.3 Baseline Solution

Since there is no prior work on finding PB items[1], we adopt a direct combination of BurstSketch and PeriodicSketch as a baseline solution. Specifically, we input the given data stream into BurstSketch and output the reported burst items at the end of each time window. Meanwhile, we use these burst items as the input stream to PeriodicSketch to obtain PB items, where each input item is adjusted to burst type.

3 PBSketch Design

3.1 Problem Statement

We provide the problem definition of periodic burst (PB) items as follows. Primarily, PB items must be generated from burst items (refer to § 2.1) with the same item key. Assume that there is a burst item set $B = \{b^e_{t_1}, b^e_{t_2}, \cdots, b^e_{t_i}, \cdots\}$ with the same key e, where $t_1, t_2, \cdots, t_i$ are the corresponding burst start times, so the burst interval v of any adjacent items can be calculated as $v_1 = t_2 - t_1, v_2 = t_3 - t_2, \cdots, v_i = t_{i+1} - t_i, \cdots$, then B is also a PB item set under the following conditions: $v_1, v_2, \cdots, v_i, \cdots$ all fall within the range $[v - \delta, v + \delta)$, where δ is the preset acceptable error, and the *interval frequency (the number of intervals falling within the range)* r is the top-K (used in this paper) or exceeds a threshold.

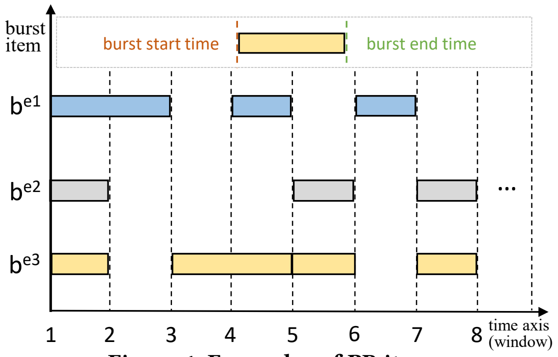


Figure 1: Examples of PB items.

As shown in Figure 1, we explain the above definition more clearly through the following examples, where $\delta = 1$ is assumed. ① For the burst item b^{e_1}, it starts to burst at time $t_1 = 1$, $t_2 = 4$, and $t_3 = 6$, and the intervals are $v_1 = 3$ and $v_2 = 2$, then: both v_1 and v_2 fall in the range $[2, 4)$, so $v = 3$ and $r = 2$; ② For the burst item b^{e_2}, it starts to burst at time $t_1 = 1$, $t_2 = 5$, and $t_3 = 7$, so there are two different intervals $v = v_1 = 4$ and $v = v_2 = 2$, and their r is both 1; ③ For the burst item b^{e_3}, it starts to burst at time $t_1 = 1$, $t_2 = 3$, $t_3 = 5$, and $t_4 = 7$, and the intervals v_1, v_2, v_3 are all 2, so $v = 2$ and $r = 3$. Finally, we can get the order of PB items in the form of $\langle key, v, r\rangle$ as follows: $\langle e_3, 2, \mathbf{3}\rangle$ is top 1, $\langle e_1, 3, \mathbf{2}\rangle$ is top 2, and $\langle e_2, 4, \mathbf{1}\rangle$ and $\langle e_2, 2, \mathbf{1}\rangle$ is top 3.

[1]In this paper, our definition of PB items is inspired by by the definitions of the burst items and periodic items in BurstSketch and PeriodicSketch, respectively. Users can also appropriately modify the definition according to their actual needs, *e.g.*, the burst items only consider burst starts [18].

3.2 Data Structure

As shown in Figure 2, the data structure of PBSketch comprises Part 1 and Part 2 that cooperate with each other: The former detects high-frequency items as potential burst items, examines them, and reports eligible ones with burst intervals; The latter tracks the K periodic items with the largest interval frequency as PB items.

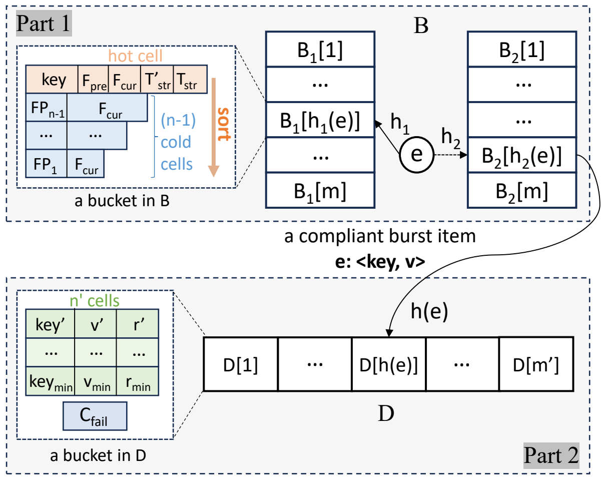


Figure 2: Data structure of PBSketch.

Part 1 is a sketch with two arrays $\mathcal{B}_1$ and $\mathcal{B}_2$, associated with pairwise independent hash functions $h_1(.)$ and $h_2(.)$, respectively. Each array has m buckets, each of which contains n cells, denoted as $cell_1, cell_2, \ldots, cell_n$, where the cell sizes increase sequentially. The maximum size cell (hot cell), $cell_n$, is used to record the complete information of a potential burst item $\langle key, F_{pre}, F_{cur}, T'_{str}, T_{str}\rangle$, where: key represents the full key of the item; F_{pre} and F_{cur} represent the frequencies of the item in the previous window and the current window, respectively, for determining burst compliance; and T'_{str} and T_{str} represent the time of the last burst start and the current burst start, respectively, for calculating the interval. The remaining $n - 1$ smaller cold cells only store basic information $\langle FP, F_{cur}\rangle$ (where FP denotes the item's fingerprint) for less active items, with variable-sized F_{cur} counters to adapt to the skewness of data streams.

Part 2 is a customized hash table $\mathcal{D}$ with m' buckets, associated with a hash function $h(.)$. Each bucket contains n' cells and one failure counter C_{fail}: Each cell records $\langle key, v, r\rangle$, where key and v are the full key of the active burst item and its burst interval from Part 1, respectively, and r is the interval frequency of the $\langle key, v\rangle$ pair; C_{fail} denotes the number of replacement failures in the bucket, which will be used in a probabilistic replacement equation mentioned later to retain potential periodic items.

3.3 Algorithm and Operations

Insertion (Part 1): The insertion in Part 1 includes an *item insertion* stage and a *window centralized processing* stage. When an item e (its key is also denoted as e) arrives in the current time window T, we map it into two candidate buckets $\mathcal{B}_1[h_1(e)]$ and $\mathcal{B}_2[h_2(e)]$ of the two arrays by calculating hash functions $h_1(e)$ and $h_2(e)$, traversing the cells in $\mathcal{B}_1[h_1(e)]$ and $\mathcal{B}_2[h_2(e)]$ and checking if it already exists. There are two cases as follows.

Case 1: If e exists in $\mathcal{B}_1[h_1(e)]$, we first increment F^e_{cur} of the cell where e is located by 1. Then, we sort all the cells in the bucket according to the size of F_{cur} to ensure that the most potential item always occupies the hot cell, where whether the current hot cell participates in the sorting depends on whether ① 'T_{str} is not empty' or ② '$T - T'_{str} < P$, P is the longest retention period' are true: If so, it participates; otherwise, it does not. After sorting, if the updated F^e_{cur} is already the 2nd largest F_{cur} in the bucket, and is still below a given threshold H', the insertion ends here; If the updated F^e_{cur} exceeds H', e qualifies as a potential burst item, and we try to insert it into the hot cell of another candidate bucket $\mathcal{B}_2[h_2(e)]$. If this hot cell needs to participate in sorting and its F_{cur} is less than F^e_{cur}, the item in it will be replaced by e: e's fingerprint FP^e is converted into the *key* e and occupies the hot cell together with its F^e_{cur}, while the original item of the hot cell is shifted down to the cold cell $cell_{n-1}$ and the *key* is converted to FP, and the item information in the bottom $cell_1$ will be cleared.

Case 2: If e is not in either candidate bucket, it is inserted into the first empty cell of $\mathcal{B}_1[h_1(e)]$ or $\mathcal{B}_2[h_2(e)]$, and the corresponding F_{cur} is set to 1; If both candidate buckets are full, $cell_1$ in $\mathcal{B}_1[h_1(e)]$ or $\mathcal{B}_2[h_2(e)]$ is randomly selected and the recorded F_{cur} is decremented by 1: Once this F_{cur} is decremented to 0, $cell_1$ is cleared and $\langle FP^e, 1\rangle$ is recorded in it, otherwise e leaves.

After we have completed the *item insertion* stage of each window, it is time for the *window centralized processing* stage. First, We calculate the ratio $\frac{F_{cur}}{F_{pre}}$ in the hot cells of all buckets: 1) If $\frac{F_{cur}}{F_{pre}} \geq k$ and $F_{cur} \geq H$ (burst threshold), a burst start occurs, and T_{str} is set to the time T of the current window; 2) If $\frac{F_{cur}}{F_{pre}} < \frac{1}{k}$, $F_{pre} \geq H$ and $T - T_{str} < L$, the current burst ends: As long as T'_{str} is not empty, we directly subtract it from T_{str} to calculate the burst interval v, *i.e.*, $v = T_{str} - T'_{str}$, and report the burst item information $\langle key, v\rangle$; After that, T'_{str} is set to T_{str}, and T_{str} is cleared (time switching). Next, F_{pre} of all hot cells is set to F_{cur} and F_{cur} is cleared in all buckets (frequency switching).

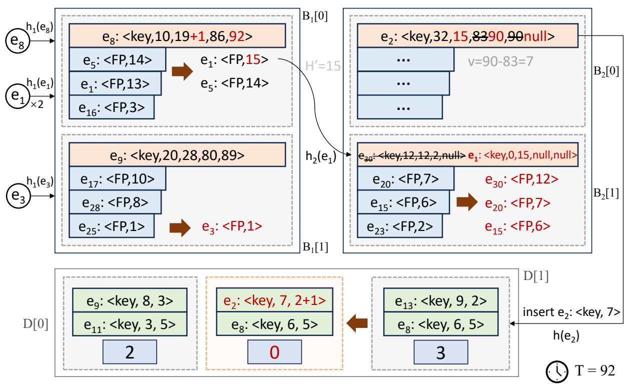


Figure 3: Examples of PBSketch.

Time Information Compression: We can reduce memory overhead by compressing T'_{str} and T_{str} recorded in the hot cell in each bucket. Here, we set the compression ratio $R_C = \frac{\textit{bits used after compression}}{\textit{bits used before compression}}$. Users can adjust R_C according to actual needs. However, in order to make a trade-off between R_C and accuracy, we provide the tuning experiments in § 5.2.

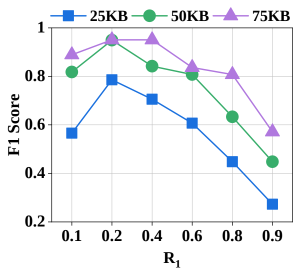
Window Smoothing Processing (WSP) Optimization: During the *window centralized processing* stage, PBSketch has to deal with the following challenge. When crossing windows, there will be a period of unified processing, which requires pausing item insertion. This will cause the throughput to drop during this period, causing fluctuations in the overall throughput of PBSketch. Thus, we further propose this WSP optimization, which does not require unified processing at the end of each window, avoiding a decrease in throughput with minimal memory cost. Specifically, we set an additional field T_{sync} in the hot cell to indicate the time of the current centralized processing operation. When item e is inserted into Part 1 at T, T_{sync} in the hot cells of its two candidate buckets needs to be checked. If $T_{sync} \neq T$, the current bucket has not been processed at T, then the operations in the window centralized processing stage are performed, and finally T_{sync} is set to T; Otherwise, the current bucket has been processed at T, and nothing needs to be done.

Insertion (Part 2): When a recently active burst item $\langle e, v\rangle$ arrives, we first map it to the bucket $\mathcal{D}[h(e)]$ by calculating the hash function $h(e)$ and check if it already exists. Note that this only works if there actually exists an e in $\mathcal{D}[h(e)]$ whose v is sufficiently close to or exactly the same as the incoming v (see § 3.1). There are two cases as follows.

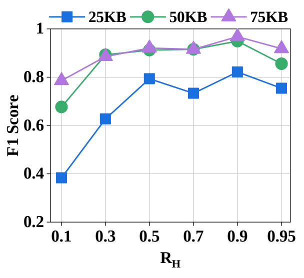
Case 1: If $\langle e, v\rangle$ exists in $\mathcal{D}[h(e)]$, we first increment the corresponding r of the cell in which it is located by 1.

Case 2: If $\langle e, v\rangle$ is not in $\mathcal{D}[h(e)]$, but there is at least one empty cell, then record $\langle e, v, 1\rangle$ in an arbitrary empty cell; If $\mathcal{D}[h(e)]$ is full, we try to replace the item with the smallest r (denoted as r_{min}) with probability $\mathcal{P} = \frac{1}{2*r_{min} - C_{fail} + 1}$ [21]: If it holds, the replacement is successful, we record $\langle e, v, r\rangle$ ($r = r_{min} + \lfloor C_{fail}/r_{min}\rfloor$) in the free cell and reset C_{fail} to 0; Otherwise, we just increment C_{fail} by 1.

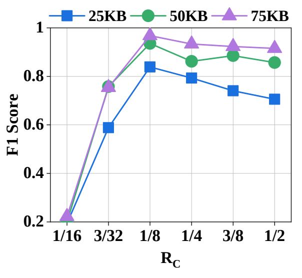
Examples (Figure 3): Below we use several examples to illustrate the insertion operation of PBSketch, where $k = 2$, $H = 20$, $H' = 15$, $L = 50$, $m = m' = 2$, $n = 4$, $n' = 2$, $T = 92$, and the frequency switching in the window centralized processing is not shown for clarity. ① e_8 is mapped to bucket $\mathcal{B}_1[0]$. e_8 happens to be already in the hot cell, whose F_{cur} is changed to 20 after being incremented by 1. Its position remains unchanged after sorting. Since its $\frac{F_{cur}}{F_{pre}} = \frac{20}{10} = 2$, a burst begins, and its T_{str} is set to 92. ② e_1 is mapped twice to bucket $\mathcal{B}_1[0]$. e_1 is in the 2nd cold cell, and its F_{cur} is incremented by 2 to become 15. After sorting, e_1 is promoted to the 1st cold cell, and e_5 originally occupying the 1st cold cell is demoted to the 2nd cold cell. Since its $F_{cur} = H' = 15$, we try to store e_1 into the hot cell of another candidate bucket $\mathcal{B}_2[1]$. Since the original resident of the current hot cell, e_{30}, has an empty T_{str} and its $F_{cur} = 12 < 15$, e_1 successfully occupies this hot cell and updates it to $\langle e_1, 0, 15, null, null\rangle$. In this case, the item information in the original 4 cells occupying $\mathcal{B}_2[1]$ is shifted downward in order, and the bottom e_{23} is cleared; e_5 and e_{16} in $\mathcal{B}_1[0]$ are shifted upward in order as e_1 has left. ③ e_3 is mapped to bucket $\mathcal{B}_1[1]$. $\mathcal{B}_1[1]$ is full, so F_{cur} of e_{25} at the bottom is decremented by 1. Since $F_{cur} = 0$ at this time, e_{25} is cleared and replaced by e_3. ④ F_{cur} of e_2 in the current window is updated to 15, *i.e.*, the recorded information is temporarily $\langle e_2, 32, 15, 83, 90\rangle$. Since $\frac{F_{cur}}{F_{pre}} = \frac{15}{32} < \frac{1}{2}$ and the burst interval $v = 90 - 83 = 7$, the current burst ends: $\langle e_2, 7\rangle$ will be inserted into $\mathcal{D}$, and T'_{str} is set to 90 and T_{str} is cleared. Next, e_2 is mapped to $\mathcal{D}[1]$. Since $\mathcal{D}[1]$ is full, it tries to replace e_{13} with



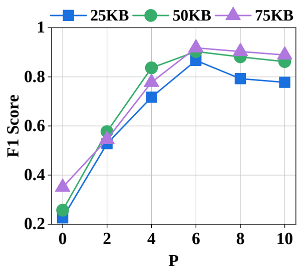
(a) F1 Score vs. R_1



(b) F1 Score vs. R_H



(c) F1 Score vs. R_C



(d) F1 Score vs. P

Figure 4: Experiments on parameter tuning.

$r_{min} = 2$. With a probability of $\mathcal{P} = \frac{1}{2\times2-3+1} = 0.5$ and assuming that it holds, e_{13} is expelled and $\langle e_2, 7, 2+1\rangle$ occupies the cell where e_{13} originally was.

Query: We just iterate over all buckets in $\mathcal{D}$ and return the $\langle key, v\rangle$ pair with the top-K largest r as PB items.

4 Mathematical Analysis

In this section, we theoretically derive the error bounds for each part of PBSketch and prove that the time complexity of PBSketch can be considered as $O(1)$. Formal details and proofs are available in Appendix A.

For Part 1, assume that: (1) *Hash Function*: h_1 and h_2 are ideal 32-bit hash functions with uniform distribution; (2) *Frequency Distribution*: item frequencies follow a power-law distribution $P(f) \propto f^{-\alpha}$; (3) *Independence:* arrivals of different items are mutually independent; (4) *Parameter Setting*: m denotes the total number of buckets in each hash table, n is the number of cells per bucket, and k is the burst ratio threshold; (5) There are N distinct items observed in the window, among which γN items have frequencies higher than the threshold for e. Then, the probability that e is *not recorded* by Part 1 is at most,

$$\begin{aligned} P_{\text{miss}} &= \left(1-e^{-\lambda}\right)^2 + \left[1-\left(1-\left(\frac{kf_{\min}}{f}\right)^{\alpha-1}\right)^n\right] \\ &\quad - \left(1-e^{-\lambda}\right)^2\left[1-\left(1-\left(\frac{kf_{\min}}{f}\right)^{\alpha-1}\right)^n\right] \end{aligned} \tag{1}$$

where $\lambda = \frac{\gamma N}{2m}$.

For Part 2, suppose each bucket contains n' cells (with m' buckets in total), and that the burst event $\langle key, v\rangle$ frequencies follow a power-law distribution. With probabilistic eviction, for the top-K most frequent burst events (among M total pairs), the probability that such an item is retained in Part 2 is at least $1-\frac{K}{M}$, ensuring that most heavy burst events are preserved.

5 Performance Evaluation

5.1 Experimental Setup

Implementation: We implement PBSketch, the baseline solution, and two typical rate limiting algorithms in C++. The hash function used is the 32-bit Bob Hash [41] with different initial seeds. The machine is equipped with a 16-core processor (24 threads, Intel(R) Core(TM) i7-13700KF CPU @ 3.40GHz) and 32GB DRAM memory. All relevant code has been released on GitHub[2].

[2]https://github.com/wenjunpaper/PBSketch

Datasets: We use the following two real-world datasets. 1) IP Trace Dataset: It is composed of streams of anonymized IP traces collected in 2018 by CAIDA [42]. In this dataset, a source IP address and a destination IP address together are considered as an item, and there are around 35M items and 17M distinct items. 2) MAWI Dataset: It is composed of anonymized packet traces from the WIDE backbone collected by the MAWI Working Group [43]. In this dataset, there are around 1M items and 80K distinct items.

Metrics:

- **Precision Rate (PR)** is the proportion of the number of correctly PB items to the number of PB items reported.
- **Recall Rate (CR)** is the proportion of the number of correctly reported PB items to the number of correctly PB items.
- **F1 Score** is calculated as $\frac{2\cdot CR\cdot PR}{CR+PR}$.
- **Average Absolute Error (AAE)** is defined as $\frac{1}{|\Psi|}\sum_{(e_i\in\Psi)}|r_i-\tilde{r}_i|$, where r_i is the real interval frequency of the PB item e_i, $\tilde{r}_i$ is the estimated interval frequency of e_i, and Ψ is the estimated set of PB items.
- **Average Relative Error (ARE)** is defined as $\frac{1}{|\Psi|}\sum_{(e_i\in\Psi)}\frac{|r_i-\tilde{r}_i|}{r_i}$.
- **Throughput** is defined as Million of operations (insertions) per second (Mops) to evaluate speed.
- **Number of Rejections** is obtained by counting the total number of requests/items that are automatically denied by the system due to exceeding the rate limits.
- **Number of Boundary Problems** is defined as the number of significant bursts in traffic that occur at the transition point between two rate limiting windows.

5.2 Parameter Tuning

We tune four hyper-parameters: The ratio R_1 of the memory size of Part 1 to the memory size of PBSketch, the ratio R_H of the frequency threshold H' (qualification for being a potential burst item, $H' = R_H \times H$) to the burst threshold H in Part 1, the compression ratio R_C, and the retention time window threshold P of potential burst items in hot cells.

Effects of R_1 (Figure 4(a)): R_1 determines the memory allocation of Part 1 and Part 2, which obviously affects the performance of PBSketch. R_1 that is too small or too large will weaken the number and accuracy of active burst items and PB items, respectively. We find that as R_1 increases, the F1 Scores of PBSketch first increase and then gradually decrease and reach a peak at $R_1 = 0.2$ with three memory capacities. Thus, we choose $R_1 = 0.2$ for our subsequent experiments.

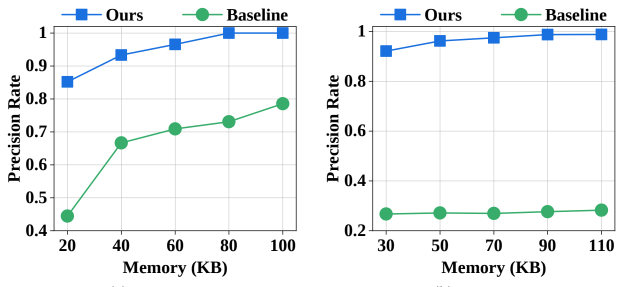
Effects of R_H (Figure 4(b)): R_H is an important parameter in Part 1 and is responsible for selecting potential burst items. If R_H is set too small, a large number of non-compliant items will occupy the space of hot cells, affecting the output of active burst items. If R_H is too large, some items with burst potential will be trapped in cold cells. As R_H increases, the F1 Scores of PBSketch show a trend of gradually increasing and then decreasing, and reach a peak at $R_H = 0.9$ with three memory capacities. Hence, we choose $R_H = 0.9$ for our subsequent experiments.

Effects of R_C (Figure 4(c)): R_C also affects the output of Part 1. If R_C is too low, although it saves some memory on the surface, the accuracy of the recorded time information is weakened and many real active burst items are missed. As R_C increases, the F1 Scores of PBSketch first increase and then gradually decrease and reach a peak at $R_C = \frac{1}{8}$ with three memory capacities. Consequently, we set R_C to 0.125 for our subsequent experiments.

Effects of P (Figure 4(d)): P directly affects the report results of Part 1. If P is too small, many active burst items that should have been reported will be lost; If P is too large, it may increase the difficulty in selecting the real target items. As P increases, the F1 Scores of PBSketch tend to increase first and then decrease slightly, reaching a peak at $P = 6$ with three memory capacities. Therefore, we select the best performing $P = 6$ for subsequent experiments.

5.3 Experiments on Accuracy

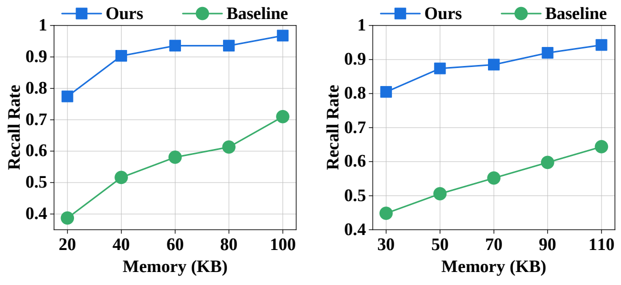
PR (Figure 5(a)-5(b)): The results show that the PR of PBSketch is about 28.3% and 69.3% higher than that of the baseline solution on the two datasets, respectively.



(a) IP Trace (b) MAWI

Figure 5: PR vs. Memory.

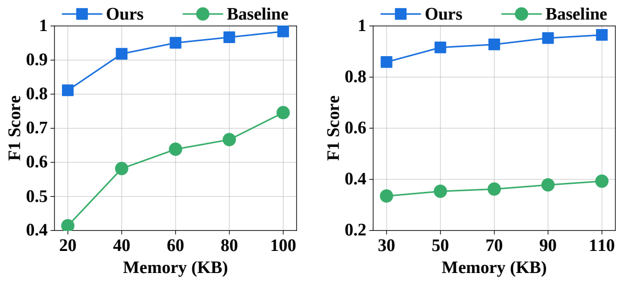
CR (Figure 6(a)-6(b)): The results show that the CR of PBSketch is about 34.2% and 33.6% higher than that of the baseline solution on the two datasets, respectively.



(a) IP Trace (b) MAWI

Figure 6: CR vs. Memory.

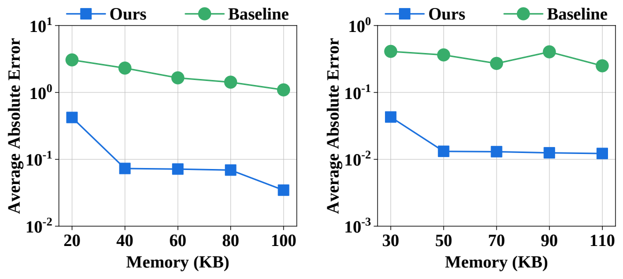
F1 Score (Figure 7(a)-7(b)): The results show that the F1 Score of PBSketch is about 31.7% and 60.0% higher than that of the baseline solution on the two datasets, respectively.



(a) IP Trace (b) MAWI

Figure 7: F1 Score vs. Memory.

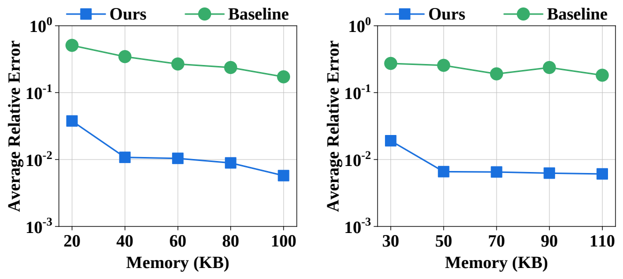
AAE (Figure 8(a)-8(b)): The results show that the AAE of PBSketch is about 22.9× and 22.2× lower than that of the baseline solution on the two datasets, respectively.



(a) IP Trace (b) MAWI

Figure 8: AAE vs. Memory.

ARE (Figure 9(a)-9(b)): The results show that the ARE of PBSketch is about 25.6× and 30.1× lower than that of the baseline solution on the two datasets, respectively.



(a) IP Trace (b) MAWI

Figure 9: ARE vs. Memory.

5.4 Experiments on Processing Speed

In this subsection, we first evaluate the average throughput of PBSketch and the baseline on two datasets.

Throughput (Figure 10): The results show that PBSketch achieves higher throughput than the baseline solution, being on average about 1.38× and 1.43× faster on the two datasets, respectively.

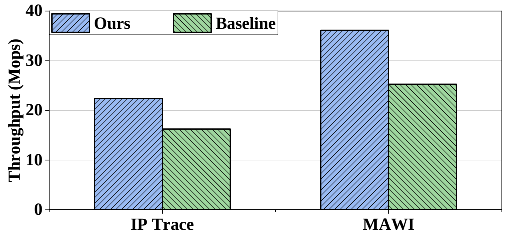


Figure 10: Insertion speed.

Then, we conduct ablation experiments to evaluate the effect of WSP optimization on PBSketch: we sample two adjacent windows (each with a length of 30000 items) to observe the fine-grained changes in throughput.

Results (Figure 11): The throughput of the optimized version only dropped by about 12.6% at the window switching point (0/30000/60000), while the unoptimized version dropped sharply by about 91.7%, verifying its effectiveness.

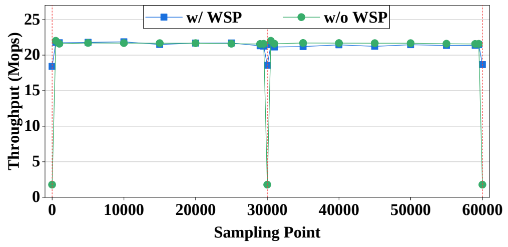


Figure 11: Ablation experiments.

5.5 FPGA Implementation

We implement PBSketch on the Xilinx Ultrascale+ VU13P FPGA to evaluate its practical deployment flexibility and further verify the effectiveness of our WSP optimization. The architecture of PBSketch FPGA version is shown in Figure 12, where the functions of the upper and lower halves correspond to Part 1 and Part 2 of the CPU version, respectively. The overall FPGA implementation adopts the fully-pipelined architecture, where input items are processed in the hardware pipeline. This architecture enables PBSketch to receive and insert a new item in each hardware clock cycle.

Upon receiving a new item, two different hash function modules will calculate the corresponding hash value and key, with two cold tables being triggered to read the data indexed by these hash value and key. One storage word of the cold table comprises several segments: hit flag, indicating whether the current word is occupied; timestamp, documenting the last update time of the word; and frequency F of the item. The hardware will update the segments in the word according to the hit flag and the WSP optimization method. Simultaneously, following our insertion operations, the cold inserting and hot inserting modules filter the most active items from two cold tables and attempts to insert these items into hot tables. Once an item within the hot tables is detected as the burst, its window is utilized to update the Hash Table C for the value of period. In our algorithm, the updating of Table D requires the calculation of the dropping probability, which is difficult to directly complete on FPGA. To address the issue, we introduce the Failing Table C, which shares the same structure of Hash Table C to document C_{fail}, and Bucket min_r Table to document the min_r value of Table D. The stored C_{fail} and min_r are sent to the look-up table (Probability Table) to get the dropping probability instead of a complex calculation of the float-point data. With dropping probability and data from Hash Table C, Table D performs the final insertion within the P stages, where P denotes the maximum count of Table D insert attempt. The insertion is neither successfully completed nor failed controlled by the dropping probability. In the final pipeline stage, it will update the Failing Table if the inserting operation fails, or updating the Bucket min_r Table based on the latest updated value.

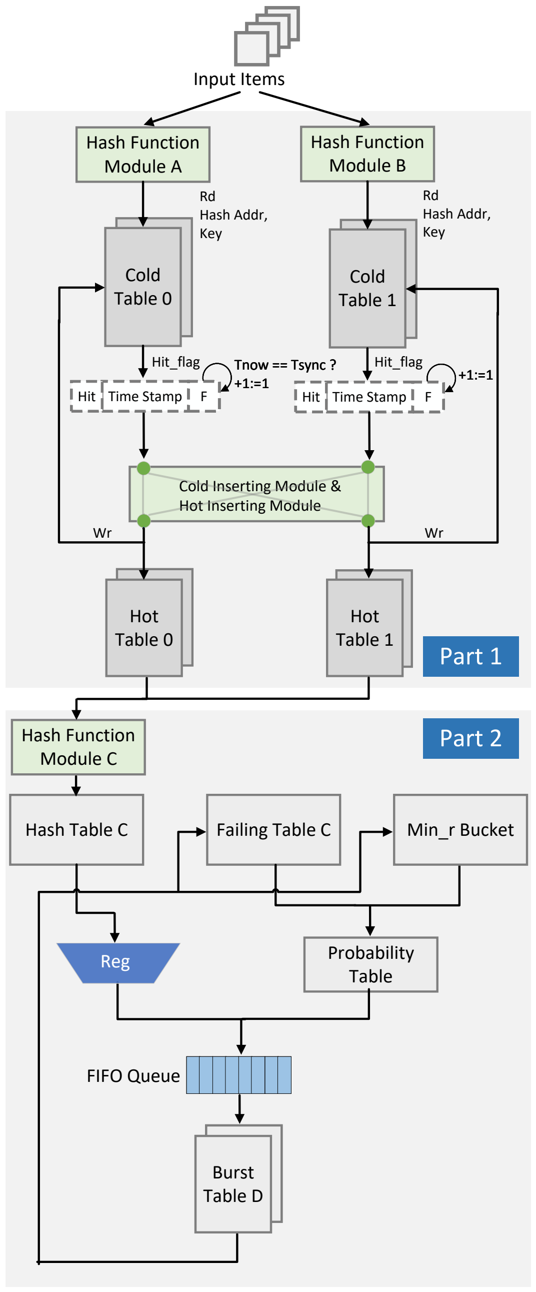


Figure 12: PBSketch (w/ WSP) FPGA implementation architecture.

Results: The resource usage is listed in Table 1. The clock frequency of PBSketch (w/ WSP) FPGA version reaches 223.2 MHz, *i.e.*, the throughput is **223.2 Mops**. If WSP optimization is not used, a window-by-window clearing overhead of 13.45 μs will be incurred. Assuming that each time window is 100 μs, the actual throughput is only $\frac{100-13.45}{100} \times 223.2 = 193.2$ Mops.

Table 1: PBSketch performance on the FPGA platform

Resource	Usage	Percentage
Logics/LUTs	123879	0.27%
Block RAM	12	0.45%
DSP Blocks	0	0%

5.6 Experiments on a Practical Application

5.6.1 *Background.*

Rate limiting is a key mechanism in network management, designed to prevent system overload and crash due to burst traffic [44, 45]. It is necessary in many scenarios, including servers, API gateways, load balancers, and firewalls. For example, both in Nginx itself, the meter table of Open vSwitch (OVS), and the filter of Spring Cloud Gateway all have rate limiting modules that support one or all of the following rate limiting algorithms: 1) counter-based; 2) leaky bucket-based.

1) ***Counter*-based rate limiting.** It refers to counting requests (items) within a fixed time window and stops accepting new items when a limit is reached. However, it may suffer from the boundary problem, where bursts in items during window transition may overload the system.

2) ***Leaky bucket*-based rate limiting.** It refers to smoothing input traffic with a fixed-capacity bucket that leaks items at a constant rate and blocks or drops excess traffic. However, its memory overhead is high because it needs to maintain individual buckets for each user/connection to track the state of their items.

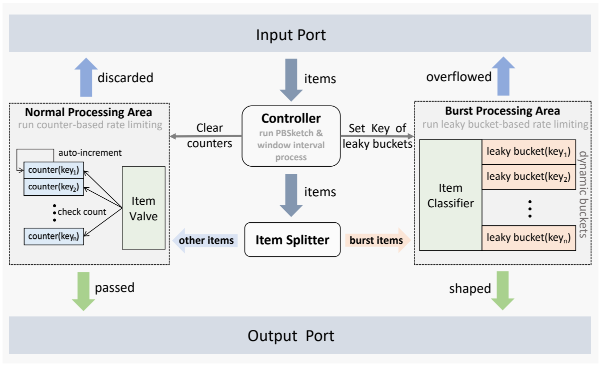
Next, we will propose our optimization solutions, called Opt-1 and Opt-2, to improve the performance of counter-based and leaky bucket-based rate limiting algorithms using PBSketch, respectively.

5.6.2 *Optimizations.*

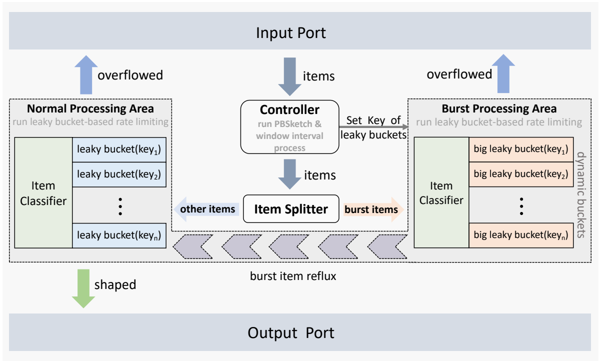
Rationale: The key is to predict bursts using detected PB items. For a PB item $\langle e, T\rangle$, if a burst is detected where item e starts to increase at time t, then it is predicted that item e will also have a burst at time $t + T$. Therefore, PBSketch can be used to predict bursts before they occur.

Architecture: For both Opt-1 and Opt-2, the overall structure can be divided into: Normal Processing Area (NPA) and Burst Processing Area (BPA). *The key of our optimizations is to introduce PBSketch in the controller to specifically handle possible burst items for BPA.* Next, we describe the structural differences between Opt-1 and Opt-2: **1)** For Opt-1, a small number of dynamic leaky buckets are added to BPA based on the original counter-based rate limiting, as shown in Figure 13(a); **2)** For Opt-2, a very small number of dynamic big leaky buckets are added to BPA based on the original leaky bucket-based rate limiting, as shown in Figure 13(b).

Implementation: 1) For Opt-1, during the data processing period between adjacent time windows, PBSketch can be used to detect burst items and predict the possible burst traffic in the next window later, which will be inserted into the priority queue. When new possible burst items appear in the next window, the leaky buckets



(a) Opt-1



(b) Opt-2

Figure 13: Architecture of our optimization solutions.

in BPA can be allocated to store these possible burst items in this window. In other words, this part of the burst traffic and the rest of the normal traffic are applied to the leaky bucket-based and counter-based rate limiting algorithms, respectively. In this way, the ability of the leaky bucket to cache items can be used to achieve the functions of reducing rejected items and traffic shaping, alleviating the shortcoming of the counter-based rate limiting algorithm. **2)** For Opt-2, its implementation is similar to that of Opt-1, except that both areas are handled by leaky buckets. In this way, the ability to handle burst traffic becomes stronger.

Hence, after optimizations of PBSketch, the main processes and steps of Opt-1 and Opt-2 are as follows:

- **Item Passing Process:**
  - **[Step 1]** After the item enters the port, copy the key of the item and insert the key into PBSketch.
  - **[Step 2]** The controller's traffic diversion algorithm (item splitter) diverts the item: If there is a leaky bucket in BPA that corresponds to the key, it is determined that the item belongs to burst traffic and it is diverted to BPA (**[Step 3]**); Otherwise, the item is diverted to NPA (**[Step 4]**).
  - **[Step 3]** The item is passed according to the specific algorithm used by BPA.
  - **[Step 4]** The item is passed according to the specific algorithm used by NPA.
- **Window Interval Process:**
  - **[Step 1]** Clear all counters in NPA to 0 and restart counting (only for Opt-1).

Table 2: Memory consumption (MB) of each part in the four solutions

Solution	Normal Processing Area (NPA)	Burst Processing Area (BPA)	PBSketch	Total
Counter-Based Rate Limiting	98	0	0	98
Opt-1	98	16	0.2	114.2
Leaky Bucket-Based Rate Limiting	6594	0	0	6594
Opt-2	6594	16	0.2	6610.2

– **[Step 2]** Recycle the empty leaky buckets in BPA and make them free buckets, *i.e.*, they no longer belong to the original items.
– **[Step 3]** Utilize PBSketch to predict the burst items, and insert the predicted burst items into the priority queue in the form of $\langle predicted\_time, burst\_item\_key \rangle$. The queue can be easily checked to find the possible bursts in the next window.

- **Leaky Bucket Allocation:**

– While item passing, simultaneously check the priority queue, extract all items that are predicted to generate bursts in the next window. If there are still free buckets, allocate leaky buckets to these items in BPA, *i.e.*, allocate a free bucket to each item so that the bucket is no longer free.

5.6.3 *Experimental Results.*

Experiment 1: Firstly, we measure the impact of the number of dynamic buckets in BPA on the *number of rejections* as well as *number of boundary problems* (only for Opt-1 and its unoptimized version). In this experiment, we fix the number of rate limits per window[3] to 50.

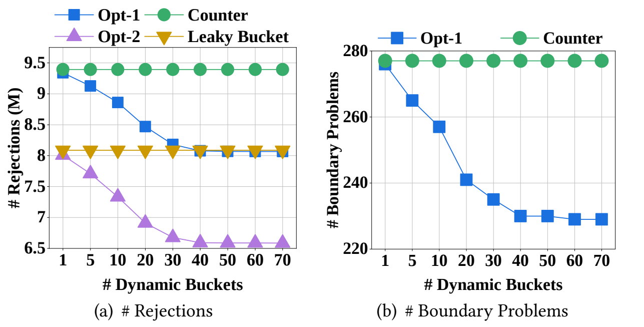


Figure 14: Evaluation on Opt-1 and Opt-2 by varying # dynamic buckets (Experiment 1).

Result 1 (Figure 14(a)-14(b)): 1) We find that Opt-1 and Opt-2 reduce the number of rejections by 14.1% and 18.6% when the number of dynamic buckets is 70, respectively; **2)** We find that Opt-1 reduces the number of boundary problems by 17.3% when the number of dynamic buckets is 70.

Experiment 2: Secondly, we measure the impact of the number of rate limits per window on the *number of rejections* and the *number of boundary problems* (only for Opt-1 and its unoptimized version). In this experiment, we fix the number of dynamic buckets to 50.

Result 2 (Figure 15(a)-15(b)): 1) We find that Opt-1 and Opt-2 can reduce the number of rejections by about 11.7% and 15.1% on average, respectively; **2)** We find that Opt-1 reduces the number of boundary problems by about 14.1% on average.

Analysis: The memory overhead of each part of all solutions in the above experiments is shown in Table 2. We find that Opt-1

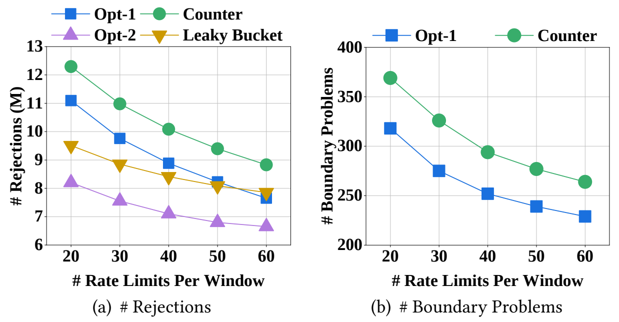


Figure 15: Evaluation on Opt-1 and Opt-2 by varying # rate limits per window (Experiment 2).

requires about 16% additional memory consumption, while the additional memory required by Opt-2 is almost negligible. In terms of the increase in memory, the memory consumption of the leaky bucket-based rate limiting algorithm is about 67× higher than that of the counter-based rate limiting algorithm, but it only reduces the number of rejections by less than 20% on average; While the total memory consumption required by PBSketch for optimizing the counter-based rate limiting algorithm is much smaller than that of the leaky bucket-based rate limiting algorithm, but it achieves a level of more than 50% reduction in the number of rejections, and is even better than that of the leaky bucket-based rate limiting algorithm in some cases.

6 Conclusion

Real-time detection of PB items in high-speed data streams plays an important role in many applications. In this paper, we propose a novel algorithm called PBSketch to detect PB items in data streams, which is the first sketch method to address this problem. Our experimental results show that PBSketch not only significantly improves accuracy and speed compared to the baseline solution, but is also successfully applied to optimize the two rate-limiting algorithms and clearly improve their performance.

Acknowledgments

We sincerely thank Jiarui Guo (Peking University) for his valuable suggestions on mathematical analysis. We also sincerely thank the anonymous reviewers for their constructive comments. This work was supported in part by the National Key Research and Development Program of China under grant No. 2025YFE0200100, the Basic and Frontier Research Project of Pengcheng Laboratory under grant No. 2025QYB004, the Major Key Project of Pengcheng Laboratory under grant No. PCL2025A07, the Young Top-notch Talent Project of Guangdong Province under grant No. 2023TQ07X362, and the National Natural Science Foundation of China under grant No. 62402012 and under grant No. 62102203.

[3]The number of items allowed through each window.

References

[1] Siddharth Bhatia, Mohit Wadhwa, Kenji Kawaguchi, Neil Shah, Philip S Yu, and Bryan Hooi. Sketch-based anomaly detection in streaming graphs. In *ACM SIGKDD*, pages 93–104, 2023.

[2] Qilong Shi, Chengjun Jia, Wenjun Li, Zaoxing Liu, Tong Yang, Jianan Ji, Gaogang Xie, Weizhe Zhang, and Minlan Yu. BitMatcher: Bit-level counter adjustment for sketches. In *IEEE ICDE*, pages 4815–4827, 2024.

[3] Yiyan Qi, Rundong Li, Pinghui Wang, Yufang Sun, and Rui Xing. Qsketch: An efficient sketch for weighted cardinality estimation in streams. In *ACM SIGKDD*, pages 2432–2443, 2024.

[4] Yiping Wang, Yanhao Wang, and Cen Chen. Dpsw-sketch: A differentially private sketch framework for frequency estimation over sliding windows. In *ACM SIGKDD*, pages 3255–3266, 2024.

[5] Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Randomized admission policy for efficient top-k and frequency estimation. In *IEEE INFOCOM*, pages 1–9, 2017.

[6] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *ACM SIGCOMM*, pages 561–575, 2018.

[7] Lu Tang, Qun Huang, and Patrick PC Lee. Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In *IEEE INFOCOM*, pages 2026–2034, 2019.

[8] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In *ACM SIGKDD*, pages 1574–1584, 2020.

[9] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In *ACM SIGCOMM*, pages 207–222, 2021.

[10] Zhuochen Fan, Ruixin Wang, Yalun Cai, Ruwen Zhang, Tong Yang, Yuhan Wu, Bin Cui, and Steve Uhlig. Onesketch: A generic and accurate sketch for data streams. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12887–12901, 2023.

[11] Qilong Shi, Yuchen Xu, Jiuhua Qi, Wenjun Li, Tong Yang, Yang Xu, and Yi Wang. Cuckoo counter: Adaptive structure of counters for accurate frequency and top-k estimation. *IEEE/ACM Transactions on Networking*, 31(4):1854–1869, 2023.

[12] Lu Cao, Qilong Shi, Yuxi Liu, Hanyue Zheng, Yao Xin, et al. Bubble sketch: A high-performance and memory-efficient sketch for finding top-k items in data streams. In *ACM CIKM*, pages 3653–3657, 2024.

[13] Haipeng Dai, Muhammad Shahzad, Alex X Liu, and Yuankun Zhong. Finding persistent items in data streams. *Proc. VLDB Endow.*, 10(4):289–300, 2016.

[14] He Huang, Yu-E Sun, Shigang Chen, Shaojie Tang, Kai Han, Jing Yuan, and Wenjian Yang. You can drop but you can't hide: k-persistent spread estimation in high-speed networks. In *IEEE INFOCOM*, pages 1889–1897, 2018.

[15] Haipeng Dai, Meng Li, Alex X Liu, Jiaqi Zheng, and Guihai Chen. Finding persistent items in distributed datasets. *IEEE/ACM Transactions on Networking*, 28(1):1–14, 2019.

[16] 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. *Proc. VLDB Endow.*, 14(2):128–140, 2020.

[17] Lu Cao, Qilong Shi, Weiqiang Xiao, Nianfu Wang, Wenjun Li, Zhijun Li, Weizhe Zhang, and Mingwei Xu. Hypersistent sketch: Enhanced persistence estimation via fast item separation. In *IEEE ICDE*, pages 3030–3042, 2025.

[18] Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. Topicsketch: Real-time bursty topic detection from twitter. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2216–2229, 2016.

[19] Debjyoti Paul, Yanqing Peng, and Feifei Li. Bursty event detection throughout histories. In *IEEE ICDE*, pages 1370–1381, 2019.

[20] Zheng Zhong, Shen Yan, Zikun Li, Decheng Tan, Tong Yang, and Bin Cui. Burstsketch: Finding bursts in data streams. In *ACM SIGMOD*, pages 2375–2383, 2021.

[21] Zhuochen Fan, Yinda Zhang, Tong Yang, Mingyi Yan, Gang Wen, Yuhan Wu, Hongze Li, and Bin Cui. Periodicsketch: Finding periodic items in data streams. In *IEEE ICDE*, pages 96–109, 2022.

[22] Zirui Liu, Chaozhe Kong, Kaicheng Yang, Tong Yang, Ruijie Miao, Qizhi Chen, Yikai Zhao, Yaofeng Tu, and Bin Cui. Hypercalm sketch: One-pass mining periodic batches in data streams. In *IEEE ICDE*, pages 14–26, 2023.

[23] Zhuochen Fan, Zhoujing Hu, Yuhan Wu, Jiarui Guo, Sha Wang, Wenrui Liu, Tong Yang, Yaofeng Tu, and Steve Uhlig. Pisketch: Finding persistent and infrequent flows. *IEEE/ACM Transactions on Networking*, 31(6):3191–3206, 2023.

[24] Jiayao Wang, Qilong Shi, Xiyan Liang, Han Wang, Wenjun Li, Ziling Wei, Weizhe Zhang, and Shuhui Chen. Pssketch: Finding persistent and sparse flow with high accuracy and efficiency. In *ACM SIGKDD*, pages 2950–2961, 2025.

[25] Zhuochen Fan, Jiarui Guo, Xiaodong Li, Tong Yang, Yikai Zhao, Yuhan Wu, Bin Cui, Yanwei Xu, Steve Uhlig, and Gong Zhang. Finding simplex items in data streams. In *IEEE ICDE*, pages 1953–1966, 2023.

[26] Xiaodong Li, Zhuochen Fan, Haoyu Li, Zheng Zhong, Jiarui Guo, Sheng Long, Tong Yang, and Bin Cui. Steadysketch: Finding steady flows in data streams. In *IEEE/ACM IWQoS*, pages 01–09, 2023.

[27] Zhuochen Fan, Xiangyuan Wang, Xiaodong Li, Jiarui Guo, Wenrui Liu, et al. Steadysketch: A high-performance algorithm for finding steady flows in data streams. *IEEE/ACM Transactions on Networking*, 32(6):5004–5019, 2024.

[28] Jiaqian Liu, Haipeng Dai, Rui Xia, Meng Li, Ran Ben Basat, Rui Li, and Guihai Chen. Duet: A generic framework for finding special quadratic elements in data streams. In *ACM WWW*, pages 2989–2997, 2022.

[29] Fangming Liu, Jian Guo, Xiaomeng Huang, and John CS Lui. eba: Efficient bandwidth guarantee under traffic variability in datacenters. *IEEE/ACM Transactions on Networking*, 25(1):506–519, 2016.

[30] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. Alibaba hpn: A data center network for large language model training. In *ACM SIGCOMM*, pages 691–706, 2024.

[31] Xiaoyang Zhao, Chuan Wu, and Xia Zhu. Dynamic flow scheduling for dnn training workloads in data centers. *IEEE Transactions on Network and Service Management*, 2024.

[32] Rafael Ramos Regis Barbosa, Ramin Sadre, and Aiko Pras. Towards periodicity based anomaly detection in scada networks. In *IEEE ETFA*, pages 1–4, 2012.

[33] Zhiyuan Zheng and AL Narasimha Reddy. Safeguarding building automation networks: The-driven anomaly detector based on traffic analysis. In *IEEE ICCCN*, pages 1–11, 2017.

[34] Alexandr Kuznetsov, Sergii Kavun, Oleksii Smirnov, Vitalina Babenko, Oleksandr Nakisko, and Kateryna Kuznetsova. Malware correlation monitoring in computer networks of promising smart grids. In *IEEE ESS*, pages 347–352, 2019.

[35] Aanshi Bhardwaj, Veenu Mangat, Renu Vig, Subir Halder, and Mauro Conti. Distributed denial of service attacks in cloud: State-of-the-art of scientific and commercial solutions. *Computer Science Review*, 39:100332, 2021.

[36] Tianrui Hu, Daniel J Dubois, and David Choffnes. Behaviot: Measuring smart home iot behavior using network-inferred behavior models. In *ACM IMC*, pages 421–436, 2023.

[37] Lianjin Ye, Qing Li, Xudong Zuo, Jingyu Xiao, Yong Jiang, Zhuyun Qi, and Chunsheng Zhu. Puff: A passive and universal learning-based framework for intra-domain failure detection. In *IEEE IPCCC*, pages 1–8, 2021.

[38] Xudong Zuo, Qing Li, Jingyu Xiao, Dan Zhao, and Jiang Yong. Drift-bottle: A lightweight and distributed approach to failure localization in general networks. In *ACM CoNEXT*, pages 337–348, 2022.

[39] Chengkun Wei, Xing Li, Ye Yang, Xiaochong Jiang, Tianyu Xu, Bowen Yang, Taotao Wu, Chao Xu, Yilong Lv, Haifeng Gao, et al. Achelous: Enabling programmability, elasticity, and reliability in hyperscale cloud networks. In *ACM SIGCOMM*, pages 769–782, 2023.

[40] Jingyu Xiao, Qing Li, Dan Zhao, Xudong Zuo, Wenxin Tang, and Yong Jiang. Themis: A passive-active hybrid framework with in-network intelligence for lightweight failure localization. *Computer Networks*, 255:110836, 2024.

[41] The source code of bob hash. http://burtleburtle.net/bob/hash/evahash.html.

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

[43] Mawi working group traffic archive. https://mawi.wide.ad.jp/mawi/.

[44] Monia Ghobadi, Yuchung Cheng, Ankur Jain, and Matt Mathis. Trickle: Rate limiting youtube video streaming. In *USENIX ATC*, pages 191–196, 2012.

[45] Donatella Firmani, Francesco Leotta, and Massimo Mecella. On computing throttling rate limits in web apis through statistical inference. In *IEEE ICWS*, pages 418–425, 2019.

A Mathematical Analysis

A.1 Insertion of Part 1

A.1.1 ***Time complexity.***

The insertion algorithm for Part 1 in PBSketch consists of the following steps:

- **Hash Computation:** Given an input item e, two hash functions $h_1(e)$ and $h_2(e)$ are computed to determine the candidate buckets in two arrays $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. Here, calculating two independent hash functions $h_1(e)$ and $h_2(e)$ takes $O(1)$ time.
- **Cell Scan:** For each candidate bucket, n cells are scanned to check whether e already exists, which takes a total of $O(n)$ time.
- **Sorting:** If an update or insertion is triggered, we will use pop sorting to sort the cell, so it will cost $O(n)$.
- **Replacement/Update:** In case of replacement, update the corresponding cell(s) with new values, it costs $O(1)$ time.

Combining these steps, the overall time complexity for inserting an item is dominated by the scanning and sorting process and can be expressed as $O(n)$, where n is the number of cells per bucket. Since n is small and constant in practical deployments, the actual insertion time is $O(1)$ per item.

A.1.2 ***Error bound.***

Suppose the following assumptions hold: (1) *Hash Function*: h_1 and h_2 are ideal 32-bit hash functions with uniform distribution; (2) *Frequency Distribution*: item frequencies follow a power-law distribution $P(f) \propto f^{-\alpha}$; (3) *Independence:* arrivals of different items are mutually independent; (4) *Parameter Setting*: m denotes the total number of buckets in each hash table, n the number of cells per bucket, H is the burst frequency threshold, k is the burst ratio threshold, and the target item's frequency satisfies $f^e \geq H$.

A burst item e with frequency f^e may not be recorded in Part 1 due to three main reasons.

First, in the *insertion stage*, a burst item e may fail to be recorded if its candidate buckets are already fully occupied by other items with higher frequencies. To rigorously estimate this probability, we perform the following analysis under the assumption that hash functions are ideal and the distribution of items is sufficiently large and sparse.

Suppose there are N distinct items observed in the window, among which γN items have frequencies higher than the threshold for e. Each such high-frequency item is independently and uniformly mapped into m buckets using 2-hashing. For a target item e, we ask: what is the probability that both of its 2 candidate buckets are fully occupied by high-frequency items and hence e cannot be inserted? We proceed step by step:

Lemma 1. *To rigorously quantify the proportion of high-frequency items, we provide the explicit derivation of γ based on the power-law assumption. For analytical tractability, we approximate the discrete frequency distribution with a continuous probability density function. Let $f_{\min}$ be the minimum frequency and f_e be the threshold frequency for a target item e (where $f_e \geq H$). The probability density function is defined as $p(f) = Cf^{-\alpha}$ for $f \in [f_{\min}, \infty)$, where C is the normalization constant. First, we determine C such that the integral over the domain is 1:*

$$\int_{f_{\min}}^{\infty} Cf^{-\alpha} df = C \cdot \left[\frac{f^{1-\alpha}}{1-\alpha}\right]_{f_{\min}}^{\infty} = C \cdot \frac{0 - f_{\min}^{1-\alpha}}{1-\alpha} = 1 \qquad (2)$$

Solving for C, we obtain $C = (\alpha - 1)f_{\min}^{\alpha-1}$. Next, γ represents the proportion of items with frequencies exceeding the threshold f_e. It is calculated as:

$$\gamma = \int_{f_e}^{\infty} p(f)df = \int_{f_e}^{\infty} (\alpha - 1)f_{\min}^{\alpha-1} f^{-\alpha} df = \left(\frac{f_e}{f_{\min}}\right)^{1-\alpha} \qquad (3)$$

Remark: In our experiments, $f_{\min}$ was consistently observed as 1. Using Maximum-Likelihood Estimation (MLE), we obtained mean α values of 3.0 for the IP Trace dataset and 6.7 for the MAWI dataset, satisfying the condition $\alpha > 1$.

Lemma 2. *Suppose there are γN items with frequencies higher than a target item e in the observation window, and each item is independently and uniformly hashed into m buckets using 2-hash functions. Let X be the number of such high-frequency items that get hashed into a particular bucket. When N is large and $\gamma N \gg m$, by the law of rare events, X can be approximated by a Poisson distribution with mean*

$$\lambda = \frac{\gamma N}{2m}.$$

This lemma enables us to capture the number of high-frequency items colliding in any given bucket.

Lemma 3. *Given a bucket with n cells, the probability that it receives at least n or more high-frequency items (and thus is fully occupied by them) is*

$$P_{\text{bucket}}(n) = \Pr[X \geq n] = 1 - \sum_{k=0}^{n-1} \frac{e^{-\lambda}\lambda^k}{k!},$$

where $X \sim \text{Poi}(\lambda)$ and $\lambda = \frac{\gamma N}{2m}$.

Lemma 4. *Let e be an item of interest that has been mapped to 2 certain candidate buckets via its hash functions. If, for all of these 2 buckets, each one happens to be fully occupied by high-frequency items, then e cannot be inserted or recorded. Therefore, the overall probability that e is not recorded (misses all its candidate buckets) can be estimated as*

$$P_{\text{miss}} \approx [P_{\text{bucket}}(n)]^2$$

where $P_{\text{bucket}}(n)$ is defined in Lemma 3.

Second, after an item e is successfully inserted into the candidate bucket (*i.e.*, it is among the recorded items in at least one of its two buckets), it may still fail to be promoted into a hot cell if both of its candidate buckets have at least one item whose frequency is higher than e's current frequency, *i.e.*, e is always out-competed and never occupies a hot position. We now estimate this masking probability.

Lemma 5. *Suppose bucket B has n cells, and γN items in the window have frequencies strictly greater than e. As in Lemma 2, let $X \sim \text{Poi}(\lambda)$ be the (Poisson-approximated) number of such strong items mapped to B, where $\lambda = \frac{\gamma N}{2m}$. The probability that at least one stronger item is present in B is*

$$P_{\text{dom}} = 1 - \Pr[X = 0] = 1 - e^{-\lambda}$$

Lemma 6. *After insertion, an item e will never reside in a hot cell (in either of its two candidate buckets), if and only if, in both buckets, there is at least one item with strictly higher frequency than e. The probability that e is masked by stronger items in both its buckets is*

$$P_{\text{mask}} \approx [P_{\text{dom}}]^2 = \left(1 - e^{-\lambda}\right)^2.$$

Proof. For each candidate bucket, the event that at least one stronger item resides in the bucket is independent (under the hashing and sparsity assumption). e can only be in the hot cell if it is the bucket's top item; otherwise, it's masked. Thus, e can only be "hot" if at least one of its buckets does not have a stronger item.

Therefore, the probability that e is not hot in any bucket equals the probability that both its candidate buckets each have at least one stronger item, hence the formula above.

□

Third, we rigorously analyze the probability that e fails to be detected as bursty in the centralized window processing stage, after successful insertion in a hot cell.

Lemma 7. *Let $f_{\min}$ be the minimum frequency, and items were independently inserted into a hot cell, and their frequencies $X_1, X_2, \ldots, X_n$ are independent random variables drawn from a power-law distribution:*

$$F_X(x) = \Pr(X \leq x) = \begin{cases} 0 & x < f_{\min} \\ 1 - \left(\frac{f_{\min}}{x}\right)^{\alpha-1}, & x \geq f_{\min} \end{cases}$$

where $\alpha > 1$, and $f_{\min}$ is the minimum item frequency.

Let

$$F_{pre}^{\max} = \max\{X_1, \ldots, X_n\}$$

be the maximum in the previous window. Then the Cumulative Distribution Function (CDF) of $F_{pre}^{\max}$ is:

$$\Pr\left(F_{pre}^{\max} \leq t\right) = [F_X(t)]^n$$

and thus the Probability Density Function (PDF) is:

$$f_{F_{pre}^{\max}}(t) = n\,[F_X(t)]^{n-1} f_X(t)$$

where for $x \geq f_{\min}$,

$$f_X(x) = \frac{dF_X}{dx} = (\alpha-1) f_{\min}^{\alpha-1} x^{-\alpha}$$

Lemma 8. *Let e be an item with frequency $f \geq H$ in the current window, and k be the burst detection ratio. Let $f_{\min}$ be the minimum frequency, then the probability that e's frequency ratio to the previous-window hot cell maximum is less than k, i.e., e fails to be detected as bursty, is:*

$$P_{\text{miss,burst}} = \Pr\left(\frac{f}{F_{pre}^{\max}} < k\right) = \Pr\left(F_{pre}^{\max} > \frac{f}{k}\right) = 1 - \left[F_X\left(\frac{f}{k}\right)\right]^n$$

Proof.

$$P_{\text{miss,burst}} = \Pr\left(f < kF_{pre}^{\max}\right) = \Pr\left(F_{pre}^{\max} > \frac{f}{k}\right)$$
$$= 1 - \Pr\left(F_{pre}^{\max} \leq \frac{f}{k}\right) = 1 - [F_X(f/k)]^n$$

□

If $f/k \geq f_{\min}$,

$$F_X\left(\frac{f}{k}\right) = 1 - \left(\frac{kf_{\min}}{f}\right)^{\alpha-1}$$

Thus,

$$P_{\text{miss,burst}}(f, n, \alpha, k, f_{\min}) = 1 - \left[1 - \left(\frac{kf_{\min}}{f}\right)^{\alpha-1}\right]^n$$

If $f/k < f_{\min}$, then $F_X(f/k) = 0$, so $P_{\text{miss,burst}} = 1$ (leak is certain).

Overall Error Bound Summary. Combining the above analyses, the probability that a burst item e with frequency $f^e \geq H$ is missed in Part 1 can be comprehensively expressed as follows. The overall miss event occurs if e is not successfully inserted (*insertion failure*), or after successful insertion, it never becomes hot (*promotion failure*), or it is not registered as bursty in the centralized detection phase (*detection failure*). However, as discussed above, the event of insertion failure (Part 1) is strictly stronger than that of promotion failure (Part 2), *i.e.*, an item that cannot be inserted must necessarily fail to become hot.

Furthermore, under the assumption that the promotion and detection stages are statistically independent (conditioned on successful insertion), the probability that e is missed after insertion can be given by the union of the two independent events (promotion failure or detection failure):

$$\begin{aligned}P_{\text{miss,total}} &\approx 1 - (1 - P_{\text{mask}})(1 - P_{\text{miss,burst}})\\ &= P_{\text{mask}} + P_{\text{miss,burst}} - P_{\text{mask}}P_{\text{miss,burst}}\\ &= \left(1 - e^{-\lambda}\right)^2 + \left[1 - \left(1 - \left(\frac{kf_{\min}}{f}\right)^{\alpha-1}\right)^n\right]\\ &\quad - \left(1 - e^{-\lambda}\right)^2 \left[1 - \left(1 - \left(\frac{kf_{\min}}{f}\right)^{\alpha-1}\right)^n\right]\end{aligned}$$

where $\lambda = \frac{\gamma N}{2m}$, P_{mask} is the probability e never becomes hot (Lemma 6), and $P_{\text{miss,burst}}$ is the probability that e is not registered as bursty in the window centralized processing stage (Lemma 8).

A.2 Insertion of Part 2

A.2.1 *Time complexity*.

- **Hash Computation:** For a newly arrived burst item $\langle e, v\rangle$, compute a hash function $h(e)$ to map it to a specific bucket in $\mathcal{D}$. The calculation of $h(e)$ requires $O(1)$ time.
- **Cell Scan:** Scan all n' cells in the target bucket $\mathcal{D}[h(e)]$ to check if an equivalent $\langle e, v\rangle$ already exists, costing $O(n')$ time.
- **Update/Insertion:** If the item exists, increment the r value; if there is an empty cell, insert $\langle e, v, 1\rangle$ into that cell. Both operations are in $O(1)$ time.
- **Replacement with Probabilistic Eviction:** If the bucket is full, compute the minimum r_{min} among the cells ($O(n')$), decide on replacement with a constant-time probability calculation and, if selected, update the corresponding cell. Overall, this step is dominated by finding r_{min}, which is $O(n')$.

Combining these steps, the total time complexity for inserting a burst item in Part 2 is dominated by the cell scanning and minimum-value search process, expressed as $O(n')$, where n' is the number of cells per bucket in $\mathcal{D}$. Since n' is a small constant in practical settings, the actual insertion time per item is effectively $O(1)$ in real-world deployments.

A.2.2 *Error bound*.

To analyze the error bound in burst tracking for Part 2, let us consider the process of inserting each burst event $\langle e, v\rangle$ into the Part 2 table, which employs m' buckets, each with n' cells, and uses the probabilistic eviction rule detailed previously.

Assume there are M different $\langle e, v\rangle$ pairs, whose burst frequencies follow a power-law distribution f^{-2}.

Guarantee for Frequent Bursty Events. Let f_{new} denote the burst count of a newly inserted pair, and f_{min} denote the minimum burst count among the current items in the corresponding bucket. Due to the eviction rule, when $f_{new} \geq 2f_{min}$, the new pair is guaranteed to replace the item with the minimum count after at most $2f_{min}$ failed attempts, because the replacement probability becomes $P = \frac{1}{2f_{min} - C_{fail} + 1}$, and when C_{fail} reaches $2f_{min}$, $P = 1$. Consequently, all bursty events whose burst count is at least twice as large as the current smallest count in their hashed bucket will be eventually inserted and preserved in the Part 2 table.

Guarantee for Top-K Frequent Events. Define the burst count of the K-th most frequent pair as f_K. Using the power-law assumption, no more than $2K$ pairs have frequency above $f_K/2$:

$$P(f \leq \frac{f_K}{2}) \approx 1 - \frac{2K}{M}.$$

This implies that, among all pairs, at least a fraction $1 - \frac{2K}{M}$ has a burst count less than $f_K/2$.

Assume that items are assigned to buckets randomly, and let ρ denote the load factor (*i.e.*, probability that a given cell is occupied). For a top-K pair, the insertion succeeds directly if it finds an empty cell ($1 - \rho$), or it collides with a lower-frequency item (probability $\rho(1 - \frac{2K}{M})$), in which case probabilistic eviction ensures the higher-frequency item can replace the lower-frequency one, as shown above. Hence, the lower bound for the insertion probability (for a top-K pair) is:

$$1 - \rho + \rho\left(1 - \frac{2K}{M}\right).$$

Averaging over all possible load factors by integrating ρ from 0 to 1, we get the average lower bound:

$$\int_0^1 \left(1 - \rho + \rho\left(1 - \frac{2K}{M}\right)\right) d\rho = 1 - \frac{K}{M}.$$

That is, at least $1 - \frac{K}{M}$ fraction of top-K bursty events can be made to remain in the Part 2 table under power-law input, thanks to the probabilistic eviction.