PR-Sketch: Monitoring Per-key Aggregation of Streaming Data with Nearly Full Accuracy

Siyuan Sheng
University of Chinese Academy of Sciences
SKL of Computer Architecture, ICT, CAS
shengsiyuan@ict.ac.cn

Qun Huang
Peking University
huangqun@pku.edu.cn

Sa Wang
University of Chinese Academy of Sciences
SKL of Computer Architecture, ICT, CAS
wangsa@ict.ac.cn

Yungang Bao
University of Chinese Academy of Sciences
SKL of Computer Architecture, ICT, CAS
baoyg@ict.ac.cn

ABSTRACT

Computing per-key aggregation is indispensable in streaming data analysis formulated as two phases, an update phase and a recovery phase. As the size and speed of data streams rise, accurate per-key information is useful in many applications like anomaly detection, attack prevention, and online diagnosis. Even though many algorithms have been proposed for per-key aggregation in stream processing, their accuracy guarantees only cover a small portion of keys. In this paper, we aim to achieve nearly full accuracy with limited resource usage. We follow the line of sketch-based techniques. We observe that existing methods suffer from high errors for most keys. The reason is that they track keys by complicated mechanism in the update phase and simply calculate per-key aggregation from some specific counter in the recovery phase. Therefore, we present PR-Sketch, a novel sketching design to address the two limitations. PR-Sketch builds linear equations between counter values and per-key aggregations to improve accuracy, and records keys in the recovery phase to reduce resource usage in the update phase. We also provide an extension called fast PR-Sketch to improve processing rate further. We derive space complexity, time complexity, and guaranteed error probability for both PR-Sketch and fast PR-Sketch. We conduct trace-driven experiments under 100K keys and 1M items to compare our algorithms with multiple state-of-the-art methods. Results demonstrate the resource efficiency and nearly full accuracy of our algorithms.

PVLDB Reference Format:
Siyuan Sheng, Qun Huang, Sa Wang, and Yungang Bao. PR-Sketch: Monitoring Per-key Aggregation of Streaming Data with Nearly Full Accuracy. PVLDB, 14(10): 1783-1796, 2021.
doi:10.14778/3467861.3467868

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/N2-Sys/PR-Sketch/.

Proceedings of the VLDB Endowment, Vol. 14, No. 10 ISSN 2150-8097.
doi:10.14778/3467861.3467868

1 INTRODUCTION

Stream processing becomes an indispensable paradigm in today's data analysis, given that streaming data is continuously increasing in many areas, including click streams [59], market basket transactions [13], sensor data [38], and network traffic [14]. In stream processing, each item is typically formulated as a key-value tuple for computing. For example, the values of items belonging to each key are accumulated as *per-key aggregations*. In general, stream processing uses a two-phase architecture, an *update phase* and a *recovery phase*. The update phase maintains a lightweight data structure to process items in real time with limited memory and computation resources, and sends the data structure to the recovery phase with limited bandwidth usage. The recovery phase retrieves the statistics from the received data structure with sufficient resources.

Since data streams are continuous, one critical issue in stream processing is to deal with the limited resources in the update phase while retaining high processing accuracy. Existing methods include sampling-based methods [20, 26, 36, 57], counter-based methods [15, 48, 49], and sketch-based methods [21, 23, 28, 33, 34, 42, 46, 54, 58, 63]. Prior studies show that these methods already precisely compute various statistics for specific keys in data streams. For example, [33, 42, 58] can detect most keys whose aggregations are extremely larger than others. However, existing methods fail to cover *all* keys, i.e., most keys still suffer from high errors.

In this paper, we target to achieve high cover proportion in streaming data. Cover proportion refers to the proportion of keys (e.g., 95%) whose relative errors reach a target level (e.g., 0.1%) over all keys. Achieving high cover proportion is critical in many streaming analysis applications like online diagnosis [37, 41], attack prevention [25], and failure detection [31, 43].

We observe that the poor cover proportion of existing methods is caused by simple estimation and complicated key tracking. To this end, we pose a novel sketching design *PR-Sketch* to boost the cover proportion in two aspects. First, PR-Sketch exploits equation-based recovery in the recovery phase. Specifically, PR-Sketch builds a system of linear equations between counter values and per-key aggregations, and finds the best solution. It tolerates hash collisions and hence improves cover proportion. Also, PR-Sketch offloads the key recording part from the update to recovery phase. In particular, the update phase identifies new keys by a lightweight data structure.

Qun Huang is the corresponding author.

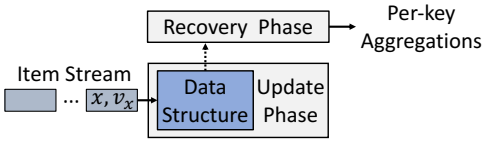


Figure 1: Processing architecture.

Then, it sends the keys to the recovery phase with little bandwidth cost and records them for further analysis. It saves memory for aggregation recording in the update phase to reduce hash collisions, and can record any number of keys with abundant resources in the recovery phase to avoid loss of critical information. Moreover, we propose an extension called *fast PR-Sketch* with a significant improvement of processing rate. Fast PR-Sketch reduces unnecessary hash operations without affecting the cover proportion.

In summary, we make the following contributions:

- We design PR-Sketch, which enables to achieve high cover proportion with limited resource usage.
- We also propose an extension called fast PR-Sketch, which can improve the processing rate further.
- We derive time complexity, space complexity, and guaranteed error probability for both PR-Sketch and fast PR-Sketch.
- We conduct trace-driven experiments to show high cover proportion and resource efficiency of our algorithms by comparing with state-of-the-art methods. With 100K keys and 1M items, we reach 93% cover proportion with 4 MB memory and 96.4% with 8 MB. In the recovery phase, the key recording part only requires less than 750 KB memory and the equation-based recovery requires less than 30ms. Fast PR-Sketch can even achieve 1.72X ~ 6.9X the throughput of baselines.

2 BACKGROUND AND MOTIVATION

2.1 Problem Formulation

Stream processing. We consider various stream processing scenarios in which streaming data continuously arrive. Here, streaming data includes click streams [59], market basket transactions [13], sensor data [38], and network traffic [14]. We abstract a stream as a sequence of items $(a_1, a_2, ..., a_S)$ as the input during a fixed-length time interval called *epoch*, where S is the number of all items at the epoch. We formulate each item as a tuple (x, v_x), where x is the *key* and v_x is the associated value ($v_x \geq 0$). We aim to compute *per-key aggregations* over a stream. Specifically, for each key x, we compute the sum of v_x for all items (x, v_x) belonging to x. Per-key aggregation is the most fundamental statistic in streaming data analysis. For instance, given the user-supplied threshold, we could report *hot keys* whose aggregations are larger than the threshold as heavy hitters [23, 28, 33, 58]. We could also report the keys whose aggregation changes in two consecutive epochs are larger than the threshold as heavy changers [33, 54, 58].

Figure 1 shows the processing architecture with two phases, an *update phase* and a *recovery phase*. The update phase is deployed in distributed entities where streaming data travels. In the update phase, we maintain some data structure to record per-key aggregations in each epoch. Since the underlying entities typically have limited resources, the data structure must be lightweight. At the end of epoch, we send the data structure from the update to recovery phase. The recovery phase is logically centralized, which analyzes the received data structure to retrieve final results. We mainly focus on the scenarios with limited bandwidth between the update and recovery phase such as data center. The scenarios with infinite bandwidth are out of our scope.

Table 1: Major notation used in the paper.

Notation	Description
Defined in Section 2	
a_i	ith item in the stream
S	number of all items
(x, v_x)	data item with key x and value v_x
χ	set of true existing keys
r_x	relative error of the aggregation of key x
r_t	target relative error
Defined in Section 3	
V_x	authentic aggregation of x
A_f	array in the filter part
A_c	array in the count part
m_f	number of buckets of array A_f
m_c	number of buckets of array A_c
k_f	number of hash functions in the filter part
k_c	number of hash functions in the count part
h_{fi}	ith hash function in the filter part
h_{ci}	ith hash function in the count part
M	coefficient matrix in the recovery phase
n	number of recorded keys
$\hat{\chi}$	set of recorded keys
ϕ	threshold for deterministic strategy
Defined in Section 4	
(ϵ_f, σ_f)	coefficient and error probability in filter part
(ϵ_c, σ_c)	coefficient and error probability in count part
N	size of entire key space
F	number of true existing keys
H	number of hot keys
c_t	target cover proportion
$\hat{V_x}$	estimated aggregation of x
Defined in Section 5	
F_i	authentic number of i-aggregation keys
$\hat{F_i}$	estimated number of i-aggregation keys

Goal 1: high cover proportion. In this paper, we aim to achieve a novel objective: *high cover proportion*. Cover proportion refers to the proportion of keys (e.g., 95%) whose relative errors [1] reach a target level (e.g., 0.1%). Formally, it equals $\frac{|\{x|x \in \chi, r_x \leq r_t\}|}{|\chi|}$, where χ is the set of true existing keys, r_x is the relative error of the aggregation of key x, and r_t is a target relative error. Prior studies advocate that approximate monitoring of specific keys like heavy hitters is acceptable. Nevertheless, high cover proportion is more general and benefits more applications. For instance, in anomaly detection like blackhole [31, 42, 47, 67] or incorrect routing [30, 42, 67], each involved key could have small aggregation values. If the results are nearly perfect with high cover proportion, administrators can focus on dealing with reports with limited concerns on false alarms or

[1] Both absolute [33, 66] and relative error [32, 46, 63, 64] are used. We adopt the latter one to better evaluate all keys with different aggregations under nearly full accuracy.

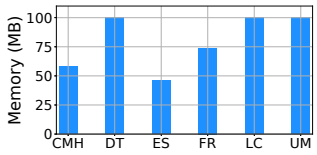


Figure 2: Required memory of existing methods for high cover proportion.

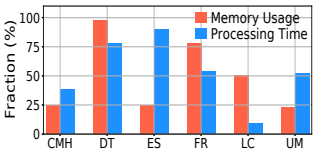


Figure 3: Fraction of resource usage about complicated key tracking.

undetected events. High cover proportion can also infer an accurate distribution of per-key aggregations. It benefits distribution-based tasks like DDoS detection [25] focusing on the keys with a single item each. However, if we only address heavy hitters, the distribution can be biased and DDoS detection can get hard. We evaluate the fine-grained applications in Section 5.3.

Goal 2: limited resource usage. In addition to high cover proportion, maintaining limited resource usage in the update phase is also significant for high efficiency. It includes limited computational overhead, memory usage, and bandwidth consumption.

- **Computational overhead:** We must keep limited overhead with lightweight operations for linear processing rate. For instance, a high-speed stream like OC-768 with 40 Gbps rate only tolerates 25 ns for 1000-bit item [39]. Otherwise, the update phase could be flooded by a torrential input stream. Although sampling the input stream can match a low processing rate, it could break high cover proportion due to loss of critical items.
- **Memory usage:** On-chip memory like SRAM is limited in commodity entities, e.g., 8 MB in FPGA [5, 63], 10-20 MB in switch [4, 6, 50], and ≤10 MB in NIC [9, 11, 56]. To this end, we must keep limited memory usage under the sheer volume of a stream over the entire lifetime. Otherwise, we have to deploy the update phase in DRAM. Since DRAM is much slower than SRAM [39], it could break the requirement of linear processing rate.
- **Bandwidth consumption:** Each commodity entity in the update phase has fixed bandwidth limitation, e.g., 0.6 Tbps in FPGA [5, 63], 1.28 Tbps in switch [4, 6, 50], and 0.2 Tbps in NIC [9, 11, 56]. Thus, we must keep limited bandwidth usage for data transmission. Otherwise, it could degrade performance of user applications and even cause congestion with serious item loss.

2.2 Sketch-based Techniques

Reversible sketching. We follow the line of sketch-based techniques especially for *reversible* ones to achieve our goals. Sketch-based techniques are a family of approximate solutions. Their key idea is to allow counters shared by multiple keys instead of keeping a dedicated one for each key. Thus, they only require limited memory. Moreover, they only need to access several mapped counters rather than enumerating all counters, which incur limited computational overhead. However, some works [21, 23, 28, 29, 61, 64, 65] are *irreversible* which do not record keys. They have to try the entire key space (e.g., 2^{104} for 5-tuple in network stream) to recover per-key aggregations, which is time-consuming. Therefore, we focus on *reversible* sketching which records keys in some way.

Low cover proportion in existing methods. Existing sketch-based techniques produce highly accurate per-key aggregations for specific keys. For example, LD-Sketch [33] and Elastic Sketch [63] can detect around 90% heavy hitters with 2 MB memory (see Section 5.3). However, existing methods cannot achieve high cover proportion with limited resource usage. To illustrate it, we implement 6 state-of-the-art methods including Count-Min-Heap (CMH) [23], Deltoid (DT) [24]), Elastic Sketch (ES) [63], FlowRadar (FR) [42], LossyCounting (LC) [48], and UnivMon (UM) [46].

We fix the cover proportion of 95% as the target. Then, we measure the amount of memory required by each existing approach under the target. The detailed experimental settings can be found in Section 5. Figure 2 shows that they require at least 46 MB memory, and some even need more than 100 MB for the target cover proportion. Their memory usage significantly exceeds the capability of common entities (e.g., 10-20 MB in commodity switches [4, 6, 50]).

Root cause 1: Simple estimation. We observe that existing sketch-based techniques estimate per-key aggregations with simple counter calculation, which is one of the root causes that lead to low cover proportion. For instance, [23, 29, 63] use the minimum counter value of mapped buckets as the estimation. [29, 46] compute the median one to estimate the aggregation. Due to the simple estimation, existing methods cannot tolerate hash collisions. Specifically, a key x can be hashed into the same bucket with multiple co-located keys. It means that the estimation of x can be biased by its co-located keys. This hash collision could incur a large relative error. Within limited memory, existing methods could suffer from serious hash collisions. Note that all of them cannot be tolerated by simple estimation, which could incur serious large relative errors. Thus, the final cover proportion can be undermined.

Root cause 2: Complicated key tracking. The second root cause is that existing methods exploit complicated key tracking entirely performed in the update phase. The key tracking mechanism can be classified into two categories: coding-based and bucket-based. For example, FlowRadar [42] exploits XOR operation to encode keys into FlowXOR field of counting table. Elastic Sketch [63] uses dedicated buckets to accommodate keys in a heavy part. It degrades the cover proportion in two aspects. On one hand, it incurs insufficient memory and computational resources for aggregation recording. To illustrate it, we implement 6 state-of-the-art methods (i.e., CMH, DT, ES, FR, LC, and UM) with the same setting in Section 5. We count the fraction of resource usage about their key tracking mechanisms in the update phase. We present the result with 8 MB memory. Figure 3 shows that for complicated key tracking, DT, FR, and LC require more than 50% memory, while other methods require more than 40% processing time. The limited resources for aggregation recording can aggravate hash collisions and undermine the cover proportion. On the other hand, the limited memory in the update phase only accommodates a limited number of keys. For unrecorded keys, their per-key aggregations are treated as 0 which must be smaller than their authentic aggregations. Therefore, it could significantly undermine the cover proportion.

3 PR-SKETCH OVERVIEW

3.1 Key Ideas

Observation. We observe that per-key aggregations follow a heavy-tailed distribution in common practical streams. Under the heavy-tailed distribution, the majority of stream volume is contributed by

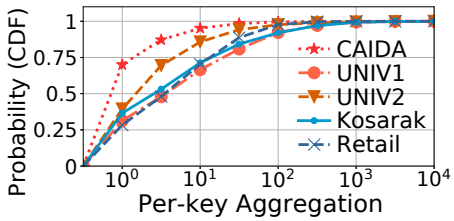


Figure 4: Heavy-tailed distribution of per-key aggregations.

a few *hot keys* with aggregations larger than a threshold. The minority is contributed by a large number of *cold keys*, whose aggregations are smaller than a threshold. [12, 16, 39, 52, 53, 59] have confirmed the heavy-tailed distribution on streams such as data center traffic, Internet traffic, and click data. We count per-key aggregations of all workloads used in Section 5, including network traffic (CAIDA, UNIV1, and UNIV2), click stream (Kosarak), and market basket data (Retail). Figure 4 shows the cumulative distribution function (CDF). More details like skewness are given in Table 3 (see Section 5). Note that we do not need to know the exact skewness in advance. Although PR-Sketch is motivated by the observation, it just means that we can perform better under the distribution than worst case. However, even if without the stream characteristics, PR-Sketch still theoretically outperforms other sketching (see Section 4).

Equation-based recovery. PR-Sketch leverages an equation-based approach to restore per-key aggregations in the recovery phase. First, PR-Sketch forms linear equations from the given keys, hash functions, and sketch. Specifically, for each bucket in the sketch, it tries all given keys by hash functions to find those hashed there. Then, the counter value of the bucket equals the sum of aggregations of those keys. After processing all buckets, PR-Sketch forms a system of linear constraints. Second, by solving the linear system, PR-Sketch can *tolerate* hash collisions, which differentiates it with prior works. Specifically, suppose that key x is co-located with others in its mapped buckets. Simple estimation of existing sketching only considers the limited mapped buckets of x. It cannot tolerate hash collisions and biases the aggregation of x. However, PR-Sketch considers the extra mapped buckets of each co-located key to obtain new constraints. Note that the extra buckets may provide more co-located keys. Thus, the above process can iterate until getting enough constraints to find a closed form of the aggregations.

However, a challenge is that the linear system could be an under-constrained problem, where we cannot get sufficient constraints to unambiguously determine per-key aggregations. To tolerate hash collisions in this case, PR-Sketch finds the "best" estimate with minimum ℓ_2-norm (i.e., $\sum_x V_x^2$, where V_x is the aggregation of key x). Since cold keys dominate the total number of keys under heavy-tailed distribution, they incur most hash collisions. Note that they share almost the same small aggregation [34] just like noises in signals. It is well suited by ℓ_2-norm minimization, which penalizes the existence of large aggregations. Thus, PR-Sketch eliminates feasible but irrelevant solutions and finds a nearly optimal one.

Key recording offloading. PR-Sketch splits the key tracking mechanism into two parts, *key identification* and *key recording*. It offloads the latter one to the recovery phase. Specifically, key identification part maintains a lightweight data structure to identify keys in the update phase. Then, it transfers each identified key from the update

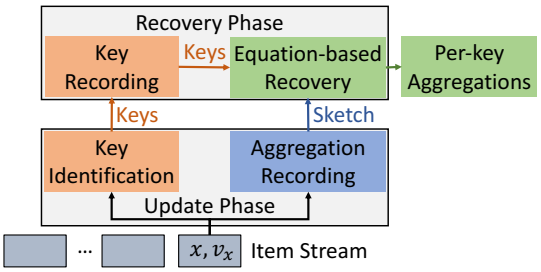


Figure 5: PR-Sketch architecture.

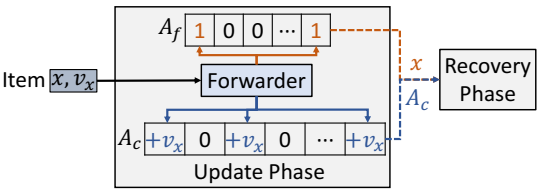


Figure 6: Data structure of PR-Sketch.

phase towards recovery phase. In the recovery phase, key recording part records those keys for further analysis. Such offloading reduces the resource usage of key tracking mechanism in the update phase, thereby we can assign more memory and computational resources for aggregation recording. We can also utilize sufficient memory in the recovery phase to record any number of keys.

A challenge is that sending keys from the update to recovery phase incurs extra bandwidth overhead. To reduce it, PR-Sketch only identifies and sends each unique key at most once. Since a few hot keys contribute the majority of stream volume under heavy-tailed distribution, many items share a same hot key. Thus, the bandwidth requirement of sending keys can be limited relative to the item arrival rate. Even if many keys emerge within a short time interval (e.g., microburst [55]) which temporarily exceeds the bandwidth limitation between the update and recovery phase, PR-Sketch can cache them in off-chip memory and sends them when bandwidth is available in each epoch rather than immediately.

Architecture. Figure 5 depicts the architecture of PR-Sketch. In the update phase, PR-Sketch deploys key identification and aggregation recording to process the item stream. The former part identifies the keys and transfers them to the recovery phase. The latter one records aggregations in some lightweight sketch. In the recovery phase, PR-Sketch deploys key recording and equation-based recovery. Key recording stores the keys transferred from key identification with abundant memory. In equation-based recovery, PR-Sketch restores per-key aggregations based on the keys in key recording and the sketch data sent from aggregation recording.

3.2 PR-Sketch

Data structure. Figure 6 shows the data structure of PR-Sketch. The update phase comprises two parts, a *filter part* for key identification and a *count part* for aggregation recording, which are connected by a *forwarder*. The forwarder forwards each item (x, v_x) to filter and count part simultaneously. The filter part contains a one-dimensional array A_f with m_f buckets. Each bucket has one bit to identify whether x is a new key, of which we have not received any items before. The filter part is associated with k_f independent hash functions, denoted by h_{fi} where $1 \le i \le k_f$. For each item,

Algorithm 1 PR-Sketch Update Algorithm

Input: Item (x, v_x)

```
1:  procedure UPDATE(x, v_x)
2:      for i = 1, 2, ..., k_f do
3:          if A_f[h_fi(x)] = 0 then
4:              Identify x as a new key
5:              A_f[h_fi(x)] ← 1
6:          end if
7:      end for
8:      if x is identified as a new key then
9:          Send key x to the recovery phase
10:     end if
11:     for i = 1, 2, ..., k_c do
12:         A_c[h_ci(x)] ← A_c[h_ci(x)] + v_x
13:     end for
14: end procedure
```

Algorithm 2 PR-Sketch Recovery Algorithm

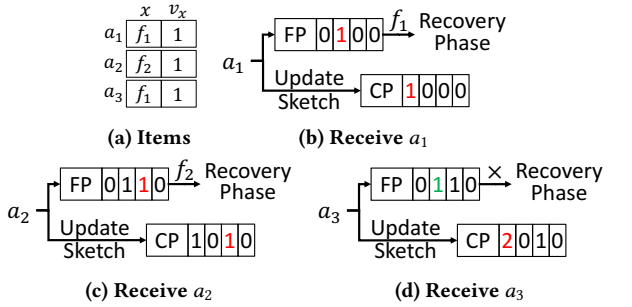
Input: Set of recorded keys $\hat{\chi}$; Array of count part A_c
Output: Per-key aggregations V

```
1:  procedure RECOVER(χ̂, A_c)
2:      Initialize the coefficient matrix M with 0
3:      idx ← 0
4:      for all x ∈ χ̂ do
5:          for i = 1, 2, ..., k_c do
6:              M[h_ci(x)][idx] ← M[h_ci(x)][idx] + 1
7:          end for
8:          idx ← idx + 1
9:      end for
10:     V ← LEASTSQUAREEQUATIONSOLVER(M, A_c)
11:     return V
12: end procedure
```

it checks the bit statuses of k_f mapped buckets before updating them. If x is newly identified, the filter part sends it to the recovery phase. The count part contains a one-dimensional array A_c with m_c buckets. For each bucket, PR-Sketch uses a counter of normal size (e.g., 32-bit) to record aggregations. The count part adopts k_c independent hash functions denoted by h_{ci} where $1 \le i \le k_c$. For each item, it increases k_c mapped buckets by v_x. It sends A_c to recovery phase at the end of each epoch. Even if the data structure of each part has been widely used in [17, 27, 29, 42, 51, 61], we emphasize that PR-Sketch combines them in a different way. For the recovery phase, PR-Sketch maintains a large array to record received keys. Since the key recording part is straightforward, we do not show the details for brevity. In equation-based recovery, PR-Sketch exploits a matrix M to represent the coefficients of linear constraints. M is a $m_c \times n$ matrix, where n is the number of recorded keys. The details of M are given in the recovery algorithm.

Update algorithm. Algorithm 1 details how to update PR-Sketch in the update phase. Each bucket in A_f and A_c is initialized as zero. For each item (x, v_x), we call the procedure UPDATE. In the filter part, PR-Sketch hashes key x via k_f hash functions one by one (Line 2). For hash function h_{fi} where $1 \le i \le k_f$, it calculates the index $h_{fi}(x)$ and checks the bit status of bucket $A_f[h_{fi}(x)]$. If the bit is 0, PR-Sketch identifies x as a new key and update $A_f[h_{fi}(x)]$ by 1 (Lines 3-6). After checking and updating the filter part, only if x is identified as a new key, PR-Sketch sends x to the recovery phase (Lines 8-10). It guarantees that each key is only sent at most once for its first item, which keeps limited bandwidth overhead as discussed in Section 3.1. In the count part, PR-Sketch hashes the key x via k_c hash functions in turn (Line 11). For hash function h_{ci} where $1 \le i \le k_c$, it calculates the index $h_{ci}(x)$ and increases the counter value of the bucket $A_c[h_{ci}(x)]$ by v_x (Line 12). Although hash collisions in A_f could incur underestimation of missing keys and overestimation of recorded keys, we provide a theoretical guarantee in Theorem 2 of Section 4. Our evaluation in Section 5 also confirms the limited effect in practice.

Update example. Figure 7 provides an example of updating PR-Sketch. We set $k_f = k_c = 1$ which are the numbers of hash functions in the filter part (FP) and count part (CP) respectively, and $m_f = m_c = 4$ which are the numbers of buckets in those parts. Suppose three items are coming in order, each of which satisfies $v_x = 1$



(a) Items (b) Receive a_1 (c) Receive a_2 (d) Receive a_3

Figure 7: Update example of PR-Sketch.

(Figure 7a). When the first item a_1 arrives, PR-Sketch checks the corresponding bucket in FP. Since the initial bit of the bucket is 0, PR-Sketch can identify f_1 is a new key. Then, it changes the bit from 0 to 1 in FP and sends the key f_1 to the recovery phase. It also increases the counter value of the bucket in CP by 1 (Figure 7b). For the second item a_2, since the key f_2 does not conflict with f_1 in FP, PR-Sketch can identify that f_2 is a new key similarly. Then, it changes the corresponding bit from 0 to 1 in FP and sends f_2 to the recovery phase. It also increases the counter value of the bucket in CP by 1, where f_2 still does not conflict with f_1 (Figure 7c). For the last item a_3, since it belongs to the same key f_1 as a_1, it is hashed in the same bucket in FP. Since the bit status has been updated as 1 by a_1, f_1 is not identified as a new key. Therefore, PR-Sketch does not send anything to the recovery phase and only increases the counter value in CP by 1 (Figure 7d).

Recovery algorithm. Algorithm 2 details how to restore per-key aggregations by equation-based recovery. Given the set of recorded keys $\hat{\chi}$ and the array of the count part A_c, we call the procedure RECOVER. First, PR-Sketch initializes the coefficient matrix M by setting each coefficient as 0 (Line 2). It uses idx to index the column of interest in M which starts from 0 (Line 3). Then, for each key x in $\hat{\chi}$, PR-Sketch updates the corresponding coefficients in M (Line 4-9). Specifically, given the key x, it calculates the hash value for each hash function h_{ci} in the count part, where $1 \le i \le k_c$. With $h_{ci}(x)$, it increases $M[h_{ci}(x)][idx]$ by 1 to reflect the mapping of x to $A_c[h_{ci}(x)]$ (Line 5-7). After updating coefficients for x, PR-Sketch increases idx by 1 to switch to the column corresponding to the next key (Line 8). Finally, given M and A_c, we call the procedure

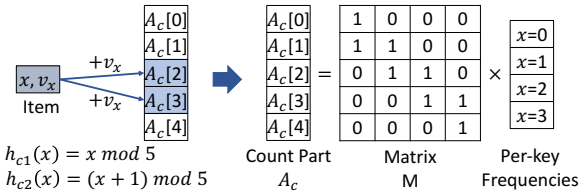


Figure 8: Recovery example of PR-Sketch.

LeastSquareEquationSolver to restore per-key aggregations V (Line 10). PR-Sketch fixes the system of linear equations by an equation solver. In particular, it is a conjugate gradient solver for least-square problems by iterative method [8]. If M is full-rank (i.e., the rank of matrix equals the number of columns), the solver finds a closed form of per-key aggregations. Otherwise, it chooses a unique solution with the minimum ℓ_2-norm.

Recovery example. Figure 8 gives an example of equation-based recovery. We set $m_c = 5$ and $k_c = 2$ in the count part. For each item, we follow Algorithm 1 to update A_f (the left part of Figure 8). Given four recorded keys ($x = 0, 1, 2, 3$), we build the coefficient matrix M by hash functions. For example, for key $x = 0$, the hash values of $h_{c1}(x)$ and $h_{c2}(x)$ are 0 and 1. It means that only $A_c[0]$ and $A_c[1]$ are mapped. Thus, in the first column of M, only the first two entries are 1 and the rests are 0. Given A_c and M, we construct and solve the linear problem to restore per-key aggregations (the right part of Figure 8). Note that M is full-rank of 4, thereby we get the closed form to tolerate hash collisions. If M is not full-rank (not shown in Figure 8), PR-Sketch finds the solution with minimum ℓ_2-norm to tolerate hash collisions. For instance, suppose that a bucket accommodates two cold keys, x_1 and x_2, both aggregations of which are 1. PR-Sketch can restore each aggregation as 1 by ℓ_2-norm minimization with 0% relative error.

3.3 Fast PR-Sketch

To further improve the processing rate, we pose an extension called *fast PR-Sketch*. We leave out the data structure and recovery algorithm in the recovery phase since they are the same as PR-Sketch.

Reducing hash operations. Fast PR-Sketch reduces hash operations without affecting cover proportion. Specifically, it marks the first item of each key with some lightweight operation. Then for the unmarked subsequent items, it reduces their hash operations in the filter part. Note that we use different hash functions in each part like [63]. It reduces hash collisions under independent configurations between the two parts. Note that PR-Sketch reports each key only for its first item. Therefore, reducing those hash operations does not affect the number of recorded keys and final cover proportion.

Data structure. Figure 9 shows the data structure of fast PR-Sketch. The update phase comprises two parts connected with a forwarder like PR-Sketch. The difference is that between the two parts, fast PR-Sketch introduces a *pruner*. For each item (x, v_x), before updating the count part, the forwarder queries the minimum counter value of mapped buckets in A_c and forwards it to the pruner. The count part updates mapped buckets by v_x for each item and sends A_c to the recovery phase at the end of each epoch. The pruner exploits a *deterministic strategy* to mark the item as the first one of x, if the minimum counter value is not larger than a predefined threshold. More strategies can be considered and we leave them in future work.

Algorithm 3 Fast PR-Sketch Update Algorithm

Input: Item (x, v_x)

```
1:  procedure FastUpdate(x, v_x)
2:      for i = 1, 2, ..., k_c do
3:          if A_c[h_ci(x)] ≤ φ then
4:              Mark (x, v_x) as a first item
5:          end if
6:          A_c[h_ci(x)] = A_c[h_ci(x)] + v_x
7:      end for
8:      if (x, v_x) is marked as a first item then
9:          for i = 1, 2, ..., k_f do
10:             if A_f[h_fi(x)] = 0 then
11:                 Identify x as a new key
12:                 A_f[h_fi(x)] = 1
13:             end if
14:         end for
15:     end if
16:     if x is identified as a new key then
17:         Send key x to the recovery phase
18:     end if
19: end procedure
```

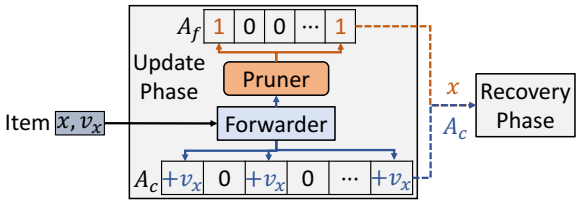


Figure 9: Data structure of fast PR-Sketch.

Only if the item is marked, the pruner transfers x to the filter part. By doing it, fast PR-Sketch can reduce unnecessary hash functions. The filter part checks bit statuses in A_f to identify whether x is a new key before updating A_f. Only if x is identified as a new key, it sends x to the recovery phase. For the recovery phase, we keep the same data structure as PR-Sketch.

Update algorithm. Algorithm 3 details how to update fast PR-Sketch in the update phase. We initialize each bucket in A_f and A_c as zero. For each item (x, v_x), fast PR-Sketch calls the procedure FastUpdate. First, in the count part, it hashes the key x via k_c hash functions one by one (Line 2). For each hash function h_{ci} where $1 \le i \le k_c$, it queries $A_c[h_{ci}(x)]$ to compare it with a predefined threshold ϕ. If the counter value is not larger than ϕ, it marks (x, v_x) as the first item (Lines 3-5). Whatever the result of pruner is, it increases $A_c[h_{ci}(x)]$ by v_x as normal (Line 6). After processed by the count part and the pruner, if (x, v_x) is marked as a first item, fast PR-Sketch processes it in the filter part (Line 8). Otherwise, it discards the unmarked item before entering the filter part. Next, in the filter part, fast PR-Sketch has the same process as PR-Sketch. Specifically, for each hash function h_{fi} where $1 \le i \le k_f$, it checks the bit status of $A_f[h_{fi}(x)]$. If the bit is 0, it identifies x as a new key and updates the bit as 1 (Lines 9-14). Finally, if x is identified as a new key, it sends x to the recovery phase (Lines 16-18).

4 THEORETICAL ANALYSIS

In this section, we follow the classical works [22, 23, 28, 58] to present theoretical analysis for both PR-Sketch and fast PR-Sketch.

Specifically, we analyze the upper bound of time complexity, space complexity, and error probability in the worst case without considering heavy-tailed distribution. We denote the logarithm with base of Euler number e as ln, and that with base of 2 as log.

Parameter configuration. Given user-supplied theoretical requirements, we follow [22] to configure parameters. Without loss of generality, we use (ϵ_f, σ_f) as the approximation coefficient and required error probability of the filter part, where $0 < \epsilon_f < 1$ and $0 < \sigma_f < 1$. Similarly, we use (ϵ_c, σ_c) as the approximation coefficient and required error probability of the count part, where $0 < \epsilon_c < 1$ and $0 < \sigma_c < 1$. Given above requirements, we set $k_f = \log \frac{1}{\epsilon_f \sigma_f}$, which is the number of hash functions in the filter part. We set $m_f = \frac{F}{\ln 2} \log \frac{1}{\epsilon_f \sigma_f}$, where m_f is the number of buckets in the filter part and F is the number of true existing keys. We set $k_c = \log \frac{1}{\sigma_c}$ and $m_c = \frac{2}{\epsilon} \log \frac{1}{\sigma_c}$, where m_c and k_c indicate the number of hash functions and buckets in the count part respectively.

4.1 Analysis of PR-Sketch

We first give the theoretical analysis of PR-Sketch. Theorem 1 provides the time and space complexity.

Theorem 1. *The time complexity of PR-Sketch is* $O(\log \frac{1}{\epsilon_f \sigma_f} + \log \frac{1}{\sigma_c})$. *The space complexity of is* $O(F \log \frac{1}{\epsilon_f \sigma_f} + \frac{1}{\epsilon_c} \log \frac{1}{\sigma_c})$.

Proof. The process of each item requires k_f hash operations in the filter part and k_c hash operations in the count part, thus the time complexity is $O(k_f + k_c) = O(\log \frac{1}{\epsilon_f \sigma_f} + \log \frac{1}{\sigma_c})$. Each bucket of filter part occupies one bit and that of count part contains a counter of normal size, thereby the space complexity is $O(m_f + m_c) = O(\frac{F}{\ln 2} \log \frac{1}{\epsilon_f \sigma_f} + \frac{2}{\epsilon_c} \log \frac{1}{\sigma_c}) = O(F \log \frac{1}{\epsilon_f \sigma_f} + \frac{1}{\epsilon_c} \log \frac{1}{\sigma_c})$. □

Theorem 2 guarantees the error probability of missing keys based on Lemma 1 and Lemma 2.

Lemma 1. *For each bucket in* A_f, *the number of contained keys obeys the binomial distribution with parameters* $k_f F$ *and* $\frac{1}{m_f}$.

Proof. Since each of the F keys is processed in the filter part for k_f times, we need $k_f F$ decisions to locate buckets. For each decision, as we use independent hash functions with uniformly distributed outputs and allocate m_f buckets in the filter part, the probability of locating any bucket equals $\frac{1}{m_f}$. □

Lemma 2. *Let* X *be the variable that denotes the number of false positives in the filter part, which are unreported yet treated as reported. Then the expectation of* X *satisfies* $E(X) = F(1 - e^{-\frac{k_f F}{m_f}})^{k_f}$.

Proof. According to Lemma 1, given a bucket in the filter part, the probability that a certain decision does not locate the bucket is $1 - \frac{1}{m_f}$. Since we have $k_f F$ decisions, the probability that the bucket is empty (i.e., with no key located here) is $(1 - \frac{1}{m_f})^{k_f F}$. Thus, the probability that the bucket is non-empty (i.e., with at least one key located here) is $1 - (1 - \frac{1}{m_f})^{k_f F}$. We treat a given key as a reported one, if it is hashed into non-empty buckets by all hash functions in the filter part. Therefore, the possibility of being treated as reported (i.e., false positive rate) is $\text{fp} = (1 - (1 - \frac{1}{m_f})^{k_f F})^{k_f}$. According to [22], it is well approximated by $(1 - e^{-\frac{k_f F}{m_f}})^{k_f}$.

Note that we query the filter part in the update phase while processing the items of F true existing keys. Each key x_i suffers from a false positive rate fp_i, where $1 \le i \le F$. Since the false positive rate fp corresponds to the worst situation that all items have been processed by the filter part, $\text{fp}_i \le \text{fp}$ holds for each key x_i. Therefore, $E(x) = \sum_{1 \le i \le F} \text{fp}_i \le F \cdot \text{fp} \approx F(1 - e^{-\frac{k_f F}{m_f}})^{k_f}$. □

Theorem 2. *PR-Sketch can miss no more than* $\epsilon_f F$ *keys with the probability of at least* $1 - \sigma_f$.

Proof. If a key is treated as a reported one, we will never report it to the recovery phase. Thus, the false positives in the filter part can incur missing keys. Specifically, the probability of missing no more than $\epsilon_f F$ keys is equal with $\Pr(X \le \epsilon_f F)$. By Markov's inequality along with Lemma 2, $\Pr(X \ge \epsilon_f F) \le \frac{1}{\epsilon_f}(1 - e^{-\frac{k_f F}{m_f}})^{k_f} = \sigma_f$. Thus, $Pr(X \le \epsilon_f F) \ge 1 - \sigma_f$ has been proved. □

Theorem 3 gives the necessary condition of the count part to achieve high cover proportion with equation-based recovery.

Theorem 3. *Given the coefficient matrix* M *and the target cover proportion* c_t, $rank(M)$ *cannot be smaller than* $c_t F$.

Proof. As mentioned in Section 3, in the recovery phase we construct the coefficient matrix M with the size of $m_c \times F$, where m_c is the number of buckets in count part and F is that of true existing keys. To achieve the target cover proportion c_t, we must get the unique feasible solution for at least $c_t F$ keys. Without loss of generality, suppose that we recover the exact aggregations for the first $c_t F$ keys. They correspond to the first $c_t F$ columns of matrix M, which constitute the matrix M^* with the size of $m_c \times c_t F$. Since M^* is the submatrix of M, $\text{rank}(M) \ge \text{rank}(M^*)$. To find the unique solution, the rank of matrix M^* must equal the number of columns, which means $\text{rank}(M^*) = c_t F$. Therefore, $\text{rank}(M) \ge c_t F$. □

However, Theorem 3 is difficult to correlate parameter configuration with cover proportion. Therefore, Theorem 4 directly guarantees the error probability of the count part. For each key, we use the minimum counter value of mapped buckets in the count part as the approximate aggregation. Note that equation-based recovery can tolerate hash collisions as discussed in Section 3. Thus, it is equivalent to providing an upper bound of error probability.

Theorem 4. *For each recorded true existing key, the relative error of estimated aggregation in the count part can exceed* $\epsilon_c S$ *with a probability at most* σ_c, *where* S *is the number of all items.*

Proof. We formulate relative error $r_x = \frac{|\hat{V}_x - V_x|}{V_x}$, where V_x is the authentic aggregation of key x and $\hat{V}_x$ is the corresponding estimation. Since x has been recorded in the recovery phase, $\hat{V}_x$ cannot be smaller than V_x. Note that x is true existing (i.e., $V_x \ge 1$), so $Pr(r_x \ge \epsilon_c S) = Pr(\hat{V}_x - V_x \ge \epsilon_c S V_x) \le Pr(\hat{V}_x \ge \epsilon_c S)$.

Let Y be the variable that denotes the counter value of some mapped bucket for each key. Since we estimate aggregation with the minimum value of mapped buckets, $Pr(\hat{V}_x \ge \epsilon_c S) \le Pr(Y \ge \epsilon_c S)^{k_c}$.

Table 2: Theoretical comparison with state-of-the-art methods.

Method	Space Complexity	Time Complexity	Cover Proportion	# of False Existing Keys
Count-Min Sketch [23]	$O(\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c})$	$O(\log\frac{1}{\sigma_c})$	$1-\sigma_c$	$O(N(1-e^{-\epsilon_c F})^{\log\frac{1}{\sigma_c}})$
Count-Min-Heap [23]	$O(\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c}+H\log N)$	$O(\log\frac{H}{\sigma_c})$	$\frac{H}{F}(1-\sigma_c)$	0
Deltoid [24]	$O(\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c}\log N)$	$O(\log\frac{1}{\sigma_c}\log N)$	$\frac{H}{F}(1-\delta)(1-\sigma_c)$	$O(H\delta\log_{\sigma_c}\frac{1}{2})$
Elastic Sketch [63]	$O(\frac{-H\log N}{\ln(1-\epsilon_f\sigma_f)}+\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c})$	$O(\log\frac{1}{\sigma_c})$	$\frac{H}{F}(1-\epsilon_f\sigma_f)(1-\sigma_c)$	0
FlowRadar [42]	$O(F\log\frac{1}{\epsilon_f\sigma_f}+\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c}\log N)$	$O(\log\frac{1}{\epsilon_f\sigma_f}+\log\frac{1}{\sigma_c})$	$(1-\epsilon_f\sigma_f)(1-\sigma_c)$	0
LossyCounting [48]	$O(\frac{\log(\epsilon_c S)}{\epsilon_c}\log N)$	$O(\log\frac{\log(\epsilon_c S)}{\epsilon_c}+\epsilon_c S\frac{\log(\epsilon_c S)}{\epsilon_c})$	$\frac{\log(\epsilon_c S)}{\epsilon_c F}$	0
UnivMon [46]	$O(\log N(\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c}+H\log N))$	$O(\log\frac{H}{\sigma_c}\log N)$	$\frac{H}{F}(1-\sigma_c{}^{\log N})$	0
PR-Sketch	$O(F\log\frac{1}{\epsilon_f\sigma_f}+\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c})$	$O(\log\frac{1}{\epsilon_f\sigma_f}+\log\frac{1}{\sigma_c})$	$(1-\epsilon_f\sigma_f)(1-\sigma_c)$	0
fast PR-Sketch	$O(F\log\frac{1}{\epsilon_f\sigma_f}+\frac{1}{\epsilon_c}\log\frac{1}{\sigma_c})$	$O(\frac{F\phi(1-P_d)}{S}\log\frac{1}{\epsilon_f\sigma_f}+\log\frac{1}{\sigma_c})$	$(1-P_d)(1-\epsilon_f\sigma_f)(1-\sigma_c)$	0

Note that we use independent hash functions with uniformly distributed outputs, each key could enter any bucket with the probability $\frac{1}{m_c}$. The expectation of Y satisfies $E(Y)=\frac{k_c S}{m_c}$. Therefore, $Pr(r_x\geq\epsilon_c S)\leq Pr(\hat{V_x}\geq\epsilon_c S)\leq(\frac{k_c}{\epsilon_c m_c})^{k_c}=\sigma_c$. □

Finally, we provide the expected cover proportion in Theorem 5.

Theorem 5. *The expected cover proportion cp is at least* $(1-\epsilon_f\sigma_f)(1-\sigma_c)$.

Proof. The cover proportion is defined as $\frac{|\{x|x\in\chi,r_x\leq r_t\}|}{|\chi|}$, where χ is the set of true existing keys and r_t is the target relative error. Note that $|\chi|=F$ and $r_t=\epsilon_c S$. On the one hand, according to Theorem 2, the false positive rate in the filter part equals $(1-e^{-\frac{k_f F}{m_f}})^{k_f}=\epsilon_f\sigma_f$. It means that $\epsilon_f\sigma_f F$ keys are not recorded in the recovery phase. Since we estimate aggregations of unrecorded keys as zero, their relative errors are 100% which cannot be covered by PR-Sketch. Thus, we only need to consider $(1-\epsilon_f\sigma_f)F$ recorded keys. On the other hand, according to Theorem 4, the possibility that the relative error of each recorded key does not exceed the target $\epsilon_c S$ is at least $1-\sigma_c$. Therefore, $\text{cp}\geq\frac{(1-\epsilon_f\sigma_f)(1-\sigma_c)F}{F}=(1-\epsilon_f\sigma_f)(1-\sigma_c)$. □

4.2 Analysis of Fast PR-Sketch

Here we give the theoretical analysis of fast PR-Sketch. We only consider the error probability of missing keys, the time complexity, and expected cover proportion. We do not consider the space complexity and the error probability of the count part. For space complexity, it still equals $O(m_f+m_c)$ in fast PR-Sketch. Thus, the result in Theorem 1 still holds. Since the items processed by the count part do not change, Theorem 4 still holds.

Theorem 6 gives the guaranteed error probability of missing keys based on Lemma 3.

Lemma 3. *Let ϕ be the threshold of the pruner. Given a new key, the probability of being discarded by pruner is P_d is at most* $(\frac{\epsilon_c S}{2\phi})^{\log\frac{1}{\sigma_c}}$.

Proof. According to the deterministic strategy of the pruner, a new key must be discarded only if it is estimated larger than ϕ. Let Y be the variable that denotes the counter value of some mapped bucket for each key. Then, combined with Markov's inequality, $P_d=Pr(Y\geq\phi)^{k_c}\leq(\frac{k_c S}{m_c\phi})^{k_c}=(\frac{\epsilon_c S}{2\phi})^{\log\frac{1}{\sigma_c}}$. □

Theorem 6. *The fast PR-Sketch can miss no more than $\epsilon_f F$ keys with the probability of at least* $1-\sigma_f+P_d(\sigma_f-\frac{1}{\epsilon_f})$.

Proof. The missing keys are composed of two parts. The first one is $P_d F$ keys whose first items are discarded by the pruner. Another part is caused by the filter part, which treats some of $(1-P_d)F$ keys as reported ones. Let X be the variable that denotes the number of missing keys. Then the expectation $E(X)=P_d F+(1-P_d)F(1-e^{-\frac{k_f F}{m_f}})^{k_f}$. According to Markov's inequality, $P_r(X\leq\epsilon_f F)\geq 1-\frac{E(X)}{\epsilon_f F}=1-\sigma_f+P_d(\sigma_f-\frac{1}{\epsilon_f})$. □

Theorem 7 provides the time complexity of fast PR-Sketch based on Lemma 4.

Lemma 4. *In fast PR-Sketch, the number of items entering the filter part of any key cannot be larger than ϕ.*

Proof. Given the key x, if the aggregation of x is not larger than ϕ, the lemma holds naturally. Therefore, we consider the key x whose aggregation is larger than ϕ. For each received item belonging to x, it must be processed by the count part. Therefore, the minimum counter value of its mapped buckets in the count part cannot be smaller than the number of received items belonging to x. For each subsequent item after the ϕth item of x, the minimum counter value must be larger than ϕ since the number of received items belonging to x exceeds ϕ. It means that all subsequent items after the ϕth item of x must be discarded by the pruner. Therefore, the lemma holds. □

Theorem 7. *The time complexity of fast PR-Sketch is* $O(k_c+k_f\frac{F\phi(1-P_d)}{S})$.

Proof. The time complexity is composed of two parts. In the count part, since all items must be processed here, each item requires k_c hash operations. In the filter part, fast PR-Sketch processes $\sum_x V'_x$ items, where V'_x is the number of items of x entering the filter part. Therefore, the time complexity of fast PR-Sketch equals $O(k_c+k_f\frac{\sum_x V'_x}{S})\leq O(k_c+k_f\frac{F\phi(1-P_d)}{S})$. □

Theorem 8. *The expected cover proportion of fast PR-Sketch is at least* $(1-P_d)(1-\epsilon_f\sigma_f)(1-\sigma_c)$.

Proof. According to Lemma 3, the expected number of discarded keys is $P_d F$. For each remaining key, the expected cover

Table 3: Workloads used in our evaluation.

Trace	# of Keys	# of Items	Skewness
CAIDA	10,743,292	239,686,200	29.73
UNIV1	557,445	18,351,743	14.82
UNIV2	191,791	98,863,335	46.92
Kosarak	41,270	8,019,015	104.08
Retail	16,470	908,576	77.22

proportion is basically the same as Theorem 5. Therefore, the final cover proportion is at least $(1 - P_d)(1 - \epsilon_f \sigma_f)(1 - \sigma_c)$. □

4.3 Theoretical Comparison

We theoretically compare PR-Sketch and fast PR-Sketch with 7 state-of-the-art methods in Table 2. It provides the space complexity, time complexity, cover proportion, and the number of false existing keys (i.e., the non-existent keys which are reported as existing). Although existing methods have similar or slightly better cover proportion than ours, they cost significantly more resources. Specifically, most of them require $\log N$ (i.e., the bit number of each key) times the space in the update phase. Some also require $\log H$ or $\log N$ times the time to update sketch. Though Count-Min Sketch is space and time efficient in the update phase, it is not reversible and incurs many false existing keys. As mentioned in Section 2.2, we do not consider such irreversible sketching. Note that equation-based recovery can tolerate hash collisions. Thus, the practical cover proportion of our algorithms can be higher than the theoretical result, which has been confirmed in Section 5.

5 EVALUATION

5.1 Setup

Platform. Our experiments run in a server with two 2.60 GHz CPUs and 125 GB DRAM. Each CPU contains 18 physical cores and 24 MB SRAM. Like prior works [33, 58, 63], we implement both PR-Sketch (PR) and fast PR-Sketch (FPR) in software. Specifically, for each algorithm, we implement the update phase in C++ and compile using GCC 5.4.0 with -O3 optimization. We use MurmurHash [1] as the hash functions. For the recovery phase, we implement the equation-based recovery with Eigen [3].

Practical workloads. We use five real-world streaming data as the datasets: CAIDA [2], UNIV1 [16], UNIV2 [16], Kosarak [7], and Retail [10]. Both UNIV1 and UNIV2 are collected from a campus network, while CAIDA is collected from a backbone network. Kosarak contains click-stream data of a Hungarian online news portal. Retail contains market basket data collected from an anonymous Belgian retail store. Table 3 summarizes the statistics of the five datasets including the number of keys, number of items, and skewness. Our experiments define different keys for the workloads, i.e., source and destination IP addresses for packet streams, news ID for click stream, and retail product ID for market basket data. For network traffic, we tune the epoch length such that the number of keys varies in our evaluation. Due to the interest of space, we present the results under 2.5s epoch length (around 100K keys) of CAIDA in most experiments. To reduce the effect of I/O overhead, we load all datasets into DRAM in advance.

Synthetic workloads. We exploit synthetic workloads only in our experiment about generality. Specifically, the workloads follow heavy-tailed distribution with different skewnesses and key numbers. We randomly generate items with distinctive keys and shuffle them into each synthetic stream.

Baselines. Since high cover proportion cannot be achieved by existing sketching, we choose six state-of-the-art methods designed for different issues as baselines, Count-Min-Heap (CMH) [23], Deltoid (DT) [24]), Elastic Sketch (ES) [63], FlowRadar (FR) [42], Lossy-Counting (LC) [60], and UnivMon (UM) [46]. They consist of 3 ad hoc methods (CMH and LC for heavy hitters, as well as DT for significant differences) and 3 general ones (ES, FR, and UM). They represent different kinds of key tracking mechanisms (detailed in Section 6). All algorithms are implemented on the same platform as mentioned before. Since some treat the value of each item as 1, we slightly adapt them by replacing 1 with more general v_x.

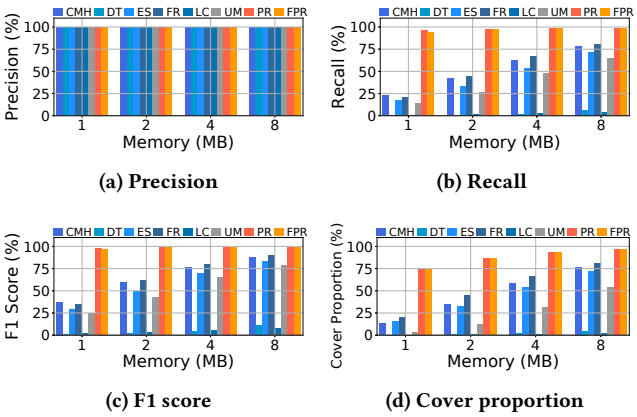
Configuration. We fix the number of hash functions as 2 for all methods including our algorithms ($k_f = k_c = 1$). We tune different amounts of memory in our evaluation. For each comparison among different methods, we keep the same memory setting. For CMH, we allocate 25% memory for its heap space. For FR, we allocate 12.5% memory for its flow filter. For ES, we allocate 25% memory for its heavy part. For LC, we conservatively set its error parameter as 0.1%, where smaller error means more supported keys and hence higher accuracy. For our algorithms, we allocate 12.5% memory for the filter part. We set $\phi = 10$ for fast PR-Sketch.

Metrics. We use a very large hash table to count correct per-key aggregations as the ground truth. Then, we evaluate the following six metrics for each approach under different memory settings:

- **Cover proportion:** It equals $\frac{|\{x|x\in\chi, r_x \le r_t\}|}{|\chi|}$, where χ is the set of true existing keys, r_x is the relative error of the aggregation of key x and r_t is a target relative error. We fix $r_t = 0.1\%$ in this paper. A larger cover proportion means higher recovery accuracy.
- **Precision:** It refers to the fraction of true existing keys recorded over all recorded keys. A higher precision means a lower possibility of misreporting.
- **Recall:** It refers to the fraction of true existing keys recorded over all true existing keys. A higher recall means better effectiveness of key tracking mechanism.
- **F1-score:** It equals $\frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$. A higher F1 score means more precise results.
- **Throughput:** It means the number of items processed per second (in units of million items per second (Mips)).
- **Bandwidth usage:** It is the fraction of bytes transported to the recovery phase over bytes received in the update phase.

5.2 Benchmark Experiments

Experiment 1 (Accuracy). In this experiment, we tune memory sizes and evaluate the accuracy. Figure 10a shows that all methods have high precision close to one, as they will not report any non-existent key. However, existing sketch-based techniques do not perform well in other metrics. As Figure 10b, both PR and FPR can achieve more than 96% recall with only 1 MB memory, while existing methods cannot achieve more than 80% even with 8 MB memory. It is because that we record keys with abundant memory in the recovery phase, while existing methods can only track limited



(a) Precision (b) Recall

(c) F1 score (d) Cover proportion

Figure 10: Experiment 1 (Accuracy).

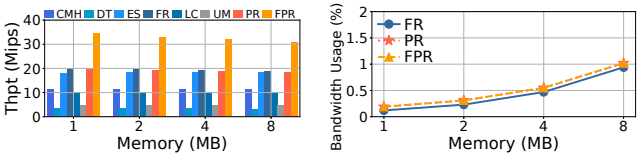
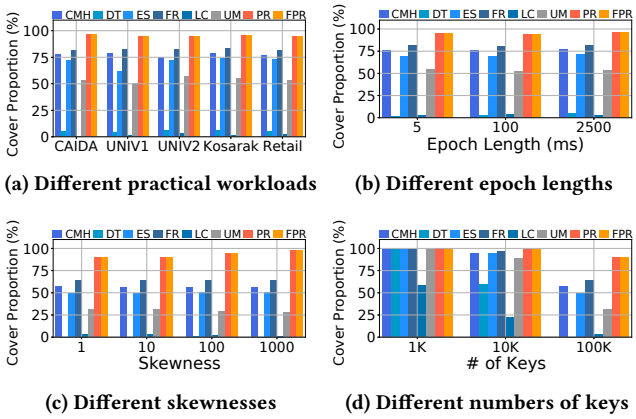


Figure 11: Experiment 2 (Throughput).

Figure 12: Experiment 3 (Bandwidth usage).



(a) Different practical workloads (b) Different epoch lengths

(c) Different skewnesses (d) Different numbers of keys

Figure 13: Experiment 4 (Generality).

keys in the update phase with strict memory constraint. The poor recall also leads to low F1 score in Figure 10c. In terms of cover proportion, Figure 10d shows that both PR and FPR can reach 93% with 4 MB memory and 96.4% with 8 MB memory. However, existing methods cannot achieve more than 81% cover proportion even with 8 MB memory. The main reason is that existing methods exploit simple counter calculation, which cannot tolerate hash collisions. They also use complicated key tracking mechanisms in the update phase, which is memory-consuming. Instead, our algorithms use equation-based recovery and key recording offloading.

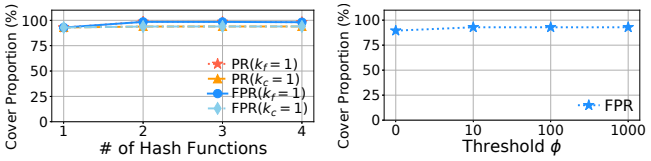
Experiment 2 (Throughput). We measure the throughput under different memory settings. Figure 11 shows that PR can achieve almost 20 Mips with 1 MB memory due to the lightweight data structure in the update phase. For existing sketch-based techniques, only FR and ES can reach this throughput, while UM and DT cannot achieve 5 Mips all the time. The reason is that FR and ES also keep lightweight data structure. However, UM exploits a layering structure with sampling and DT maintains a sketch for each bit of the key. Both of them incur heavy computational overhead. Moreover, FPR can achieve 34.5 Mips under 1 MB memory due to reducing unnecessary hash operations for most items. It is 1.72X the throughput of FR and ES, and 6.9X that of UM and DT. It is beneficial to reduce the requirement of hardware processing resources like ALUs.

Experiment 3 (Bandwidth usage). We measure the bandwidth usage with different memory sizes. Note that existing methods only consume bandwidth to send sketch data to the recovery phase, which equals the amount of allocated memory. Thus, we present the comparison with FR, and the results of other approaches are the same. Figure 12 shows that both PR and FPR cost 1.02% bandwidth usage with 8 MB memory. Since FR costs 0.94% under the same memory setting, the extra bandwidth usage of our algorithms to send keys is only 0.08%. The reason is that we only send each key for its first item instead of subsequent ones. It significantly reduces the bandwidth usage under heavy-tailed distribution.

Experiment 4 (Generality). This experiment first demonstrates the generality on practical workloads (Figure 13a and Figure 13b). We tune memory size according to the estimated number of keys F. Specifically, we estimate the number of items S within a fixed epoch through item collection rate. Although we cannot get an exact skewness in advance, a rough order of magnitude is available such as tens in data center [16]. Given S and rough skewness, F can be estimated based on heavy-tailed distribution. For each comparison, we fix the same memory setting for all methods. In Figure 13a, we exploit different practical workloads. Both PR and FPR reach 95% cover proportion. However, existing sketch-based techniques can only achieve at most 82%. In Figure 13b, we fix CAIDA and tune different epoch lengths. Our algorithms can achieve 96.4% cover proportion, while existing methods are below 81%. The reason is that we deploy the two design features, equation-based recovery and key recording offloading, in PR and FPR. It is more tolerant of hash collisions on different practical workloads and epoch lengths.

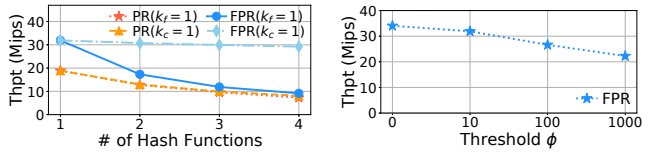
We also evaluate the generality on synthetic workloads (Figure 13c and Figure 13d). We fix 4 MB memory for all methods. In Figure 13c, we tune the skewness of heavy-tailed distribution with a fixed number of keys as 100K. The cover proportion of most methods increases as skewness rises. It is because that they rely on heavy-tailed distribution more or less. In Figure 13d, we tune the number of keys with a fixed skewness of 1. The cover proportion of all methods decreases as key number rises. It is because that a larger number of keys incurs more hash collisions. Note that even with the lowest skewness of 1 and the largest key number of 100K, our algorithms can still achieve 90% cover proportion. However, existing methods can only reach at most 64.1% in that case. The reason is the same as that on practical workloads.

Experiment 5 (Sensitivity on parameters). We measure the influence of different parameter settings on cover proportion, throughput, and bandwidth usage. For both PR and FPR, we change the number of hash functions in one part from 1 to 4 and hold that of another part as 1. For FPR, we also change the user-supplied threshold ϕ in the pruner. We fix 4 MB memory in this experiment. First, for cover proportion, Figure 14a shows that our algorithms



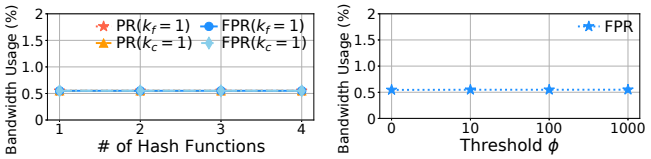
(a) Cover proportion on different numbers of hash functions

(b) Cover proportion on different thresholds



(c) Throughput on different numbers of hash functions

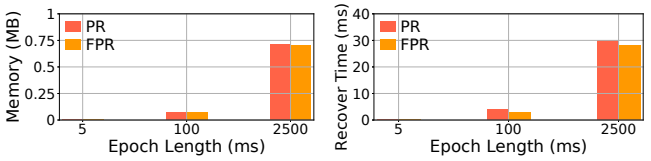
(d) Throughput on different thresholds



(e) Bandwidth usage on different numbers of hash function

(f) Bandwidth usage on different thresholds

Figure 14: Experiment 5 (Sensitivity on parameters).



(a) Key recording memory

(b) Equation solving time

Figure 15: Experiment 6 (Resource usage in recovery phase).

rise from 93% to 98% when $k_c > 1$ with $k_f = 1$, and from 93% to 94% when $k_f > 1$ with $k_c = 1$. It is because that larger k_c incurs stronger linear constraints and larger k_f entails more recorded keys due to fewer false positives in the filter part. Figure 14b shows that FPR rises from 90% to 93% when $\phi \geq 10$. The reason is that as ϕ rises, we can mark the first item and record the corresponding key with a larger possibility. Second, for throughput, Figure 14c shows that PR goes down to 7.3 Mips when $k_c = 4$ with $k_f = 1$ and 8.1 Mips when $k_f = 4$ with $k_c = 1$, while FPR goes down to 9.2 Mips when $k_c = 4$ with $k_f = 1$. It is due to the extra computational overhead caused by more hash functions. Note that FPR can still achieve 29.2 Mips when $k_f = 4$ with $k_c = 1$. The reason is that we reduce unnecessary hash operations for most items in the filter part. Thus, the increasing k_f can only affect a limited number of items. Figure 14d shows that FPR goes down to 22.3 Mips when $\phi = 1000$, as more items are processed by the filter part due to the loose condition of the pruner. Finally, for bandwidth usage, Figure 14e and Figure 14f show that our algorithms are not sensitive on the parameters. Both of them consume only 0.5% under various parameter settings since our algorithms send only once for almost all keys.

Experiment 6 (Resource usage in recovery phase). This experiment presents the resource usage of our algorithms in the recovery phase. We fix 4 MB memory and tune epoch length such that the number of keys varies. Figure 15a shows that both PR and FPR require less than 750 KB memory to record keys in the recovery phase. The reason is that we can exploit the flexibility of recovery phase and maintain a dynamically-increasing array for key recording, which is memory-efficient. Figure 15b shows that our algorithms need less than 30 ms to solve the linear system. It is due to the strong computation capability in the recovery phase.

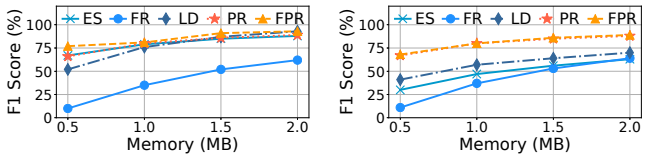


Figure 16: Heavy hitter detection.

Figure 17: Heavy changer detection.

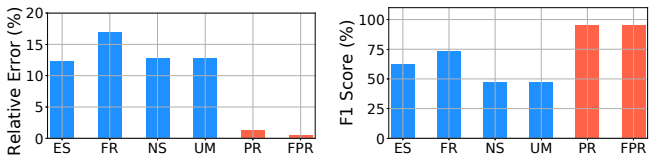


Figure 18: Entropy estimation.

Figure 19: Blackhole detection.

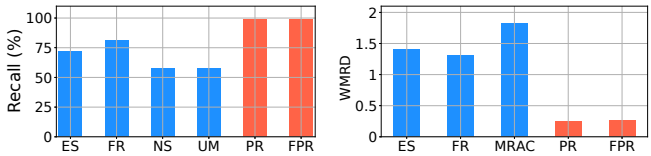


Figure 20: Incorrect routing detection.

Figure 21: Distribution estimation.

5.3 Use Cases

We consider six use cases including the normal applications and fine-grained ones. Since we achieve high cover proportion on per-key aggregations which is the most fundamental statistic, our algorithms perform well on all the following use cases.

Heavy hitter detection. Heavy hitters refer to the keys with aggregations larger than a predefined threshold. They can be used for failure detection [33] and QoS scheduling [19]. We implement three state-of-the-art methods, Elastic Sketch (ES) [63], FlowRadar (FR) [42], and LD-Sketch (LD) [33], as the baselines. We choose the threshold such that the number of heavy hitters is 10% over that of all keys. Figure 16 shows that our algorithms reach around 90% F1 score with only 2 MB memory. For existing methods, only LD and ES reach this F1 score, while FR only achieves 62% under the same memory setting. The reason is that LD and ES exploit the buckets to record the keys of heavy hitters only. However, in FR, the buckets for key tracking in the update phase are shared by all keys, and some heavy hitters cannot be decoded under hash collisions.

Heavy changer detection. Heavy changers are the keys whose aggregation changes between two adjacent epochs are larger than

the predefined threshold. They can imply the existence of DDoS attack [25]. We compare our algorithms with ES [63], FR [42], and LD [33]. We choose the threshold that the number of heavy changers is 10% over the total number of keys. Figure 17 shows that our algorithms can achieve 88% F1 score with only 2 MB memory. However, existing approaches cannot reach 70% with 2 MB memory. It is because that heavy changers might have a small aggregation in a certain epoch. Existing approaches could estimate those small aggregations as large ones and incur misses of heavy changers.

Entropy estimation. Let $-\Sigma\frac{V_x}{V}\log_2\frac{V_x}{V}$ be the authentic entropy, where V_x is the authentic aggregation of key x and $V = \Sigma V_x$. Similarly, the estimated entropy is $-\Sigma\frac{\hat{V_x}}{\hat{V}}\log_2\frac{\hat{V_x}}{\hat{V}}$, where $\hat{V_x}$ is the estimated aggregation of x and $\hat{V} = \Sigma\hat{V_x}$. It summarizes per-key aggregations for many tasks like behavior classification [62] and anomaly detection [40]. We demonstrate the advantage of our algorithms on entropy estimation compared with four state-of-the-art methods, ES [63], FR [42], NitroSketch (NS) [45], and UnivMon (UM) [46]. We consider the relative error of entropy with 1 MB memory for each approach. Figure 18 shows that the relative errors of our algorithms are smaller than 1.3%. However, all existing methods reach more than 12.72% relative error especially for FR with 16.89%. The reason is that they cannot estimate per-key aggregations accurately.

Blackhole detection. We simulate blackhole in the update phase with two entities, a normal entity A and a faulty entity B. Specifically, we transmit the item stream from entity A to entity B. Entity B drops the items of a certain set of keys due to interface failure. We randomly choose 50% true existing keys as the set of victims. We compare our algorithms with four existing methods, ES [63], FR [42], NS [45], and UM [46]. For each algorithm, we maintain one instance in each entity with 4 MB memory. We restore per-key aggregations and report the keys whose aggregation changes between the two entities are larger than 0 as victims. Figure 19 shows that our algorithms can achieve 95% F1 score, while existing methods cannot achieve more than 73%. It is due to the inaccurate estimation of per-key aggregations of existing approaches, which incurs many false positives and false negatives.

Incorrect routing detection. To simulate incorrect routing, we build a topology with three entities, entity A, B, and C. We correctly configure them such that entity A can transfer items to entity B and C. However, due to corruption of routing table in entity A, a certain number of keys which should be sent to entity B are incorrectly routed to entity C. We randomly choose 50% keys suffering from incorrect routing. We implement four sketch-based techniques, ES [63], FR [42], NS [45], and UM [46]. Each entity harbors one sketch instance with 4 MB memory. We focus on the keys with positive aggregations in entity C. Then, based on configuration, we report those which should be routed to entity B as victims. All algorithms can achieve 100% precision yet with different recalls. Figure 20 shows that our algorithms can reach 99% recall, while existing approaches cannot exceed 81%. The false negatives are incurred by the underestimated aggregations of untracked keys in entity C.

Distribution estimation. We compare our algorithms with three state-of-the-art methods, ES [63], FR [42], and MRAC [39]. We fix the memory setting as 1 MB. To evaluate each approach, we consider Weighted Mean Relative Difference (WMRD) $\frac{\sum_i |F_i-\hat{F}_i|}{\sum_i \frac{F_i+\hat{F}_i}{2}}$, where F_i is the number of keys with the aggregation of i from authentic distribution and $\hat{F}_i$ is that from estimated distribution. Figure 21 shows that both PR and FPR have a WMRD of less than 0.27. However, the WMRD values of existing methods are larger than 1.31 due to the inaccurate estimation of per-key aggregations.

6 RELATED WORK

Key tracking mechanisms. Existing key tracking mechanisms in the update phase can be classified into two categories: *coding-based* and *bucket-based*. The coding-based mechanism allows one bucket *shared* by multiple distinct keys based on an encoding rule. It includes XOR operation [27, 42, 61], multi-level hashing [18, 54], and group testing [24, 34, 44]. In the recovery phase, operators extract those keys according to a corresponding decoding rule. The bucket-based mechanism deploys a set of *dedicated* buckets to save keys explicitly, where each bucket contains at most one key. It has been used by [33, 46, 58, 63]. Operators query those keys directly without decoding. However, both of them are complicated to cost a large fraction of resource usage (see Section 2.2). Our algorithms exploit key recording offloading to save resources in update phase and track almost all keys in recovery phase.

Per-key aggregation recovery. Most existing sketch-based techniques [21, 23, 24, 29, 33, 42, 46, 58, 63] use simple counter calculation to estimate per-key aggregations. For instance, Count-Min[23] and Elastic Sketch [63] use the minimum counter value of mapped buckets. However, they suffer from hash collisions and hence significant false positives. [32] exploits compressive sensing on its fast path to improve accuracy. However, the result of compressive sensing is still a sketch instead of per-key aggregations. The final cover proportion still depends on the specific sketching in its slow path. [35] solves linear equations to recover per-key aggregations. However, it is posed from the perspective of a system, which introduces complexity to end-hosts. In the scenarios like backbone network, without the collaboration of end-hosts, it cannot solve the under-constrained problem. Our work focuses on the algorithmic improvement by equation-based recovery. We exploit the statistical property of streaming data to improve cover proportion.

7 CONCLUSION

PR-Sketch is a novel sketching design to achieve high cover proportion with limited resources. It exploits equation-based recovery and key recording offloading to improve cover proportion as well as reducing resource usage. We also present an extension called fast PR-Sketch to improve the processing rate further by reducing hash operations. For both PR-Sketch and fast PR-Sketch, we derive the time complexity, space complexity, and error probability through theoretical analysis. The trace-driven experiments show that our algorithms can achieve higher cover proportion and better resource efficiency than existing approaches.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2019YFB1802600), National Natural Science Foundation of China (U20A20179), and National Natural Science Foundation of China (61802365).

REFERENCES

[1] A. Appleby. https://github.com/aappleby/smhasher.
[2] CAIDA. https://www.caida.org/home.
[3] Eigen. https://eigen.tuxfamily.org.
[4] High Capacity StrataXGS®Trident II Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56850-series.
[5] Intel Altera. https://www.infinite-electronic.pt/pdf/d5b3139760/5SEEBF45I2N.pdf.
[6] Intel FlexPipe. https://goo.gl/kUqpU7.
[7] Kosarak. http://fimi.uantwerpen.be/data/kosarak.dat.gz.
[8] Least Squares Conjugate Gradient. https://eigen.tuxfamily.org/dox/classEigen_1_1LeastSquaresConjugateGradient.html.
[9] NVIDIA BlueField SmartNIC. https://www.mellanox.com/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf.
[10] Retail. http://fimi.uantwerpen.be/data/retail.dat.gz.
[11] Stingray SmartNIC. https://www.broadcom.com/products/ethernet-connectivity/network-adapters/smartnic/bcm58800.
[12] Lada A Adamic and Bernardo A Huberman. 2002. Zipf's Law and the Internet. *Trans. on Glottometric* 3, 1 (2002), 143–150.
[13] Rakesh Agarwal, Ramakrishnan Srikant, et al. 1994. Fast Algorithms for Mining Association Rules. In *Proc. of VLDB*.
[14] Sugam Agarwal, Murali Kodialam, and TV Lakshman. 2013. Traffic Engineering in Software Defined Networks. In *Proc. of IEEE INFOCOM*.
[15] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo C Luizelli, and Erez Waisbard. 2017. Constant Time Updates in Hierarchical Heavy Hitters. In *Proc. of ACM SIGCOMM*.
[16] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. of ACM SIGCOMM Conference on Internet Measurement Conference*.
[17] Burton H Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Trans. on Communications of the ACM* 13, 7 (1970), 422–426.
[18] Tian Bu, Jin Cao, Aiyou Chen, and Patrick PC Lee. 2010. Sequential Hashing: A Flexible Approach for Unveiling Significant Patterns in High Speed Networks. *Trans. on COMSNETS* 54, 18 (2010), 3309–3326.
[19] Andrew Campbell, Geoff Coulson, and David Hutchison. 1994. A Quality of Service Architecture. *ACM Trans. on SIGCOMM* 24, 2 (1994), 6–27.
[20] Marco Canini, Damien Fay, David J Miller, Andrew W Moore, and Raffaele Bolla. 2009. Per Flow Packet Sampling for High-Speed Network Monitoring. In *Proc. of IEEE COMSNETS*.
[21] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding Frequent Items in Data Streams. In *Proc. of International Colloquium on Automata, Languages, and Programming*.
[22] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Trans. on Foundations and Trends in Databases* 4, 1–3 (2012), 1–294.
[23] Graham Cormode and Shan Muthukrishnan. 2005. An Improved Data Stream Summary: the Count-Min Sketch and Its Applications. *Journal of Algorithms* 55, 1 (2005), 58–75.
[24] Graham Cormode and Shanmugavelayutham Muthukrishnan. 2005. What's New: Finding Significant Differences in Network Data Streams. *IEEE/ACM Trans. on Networking* 13, 6 (2005), 1219–1232.
[25] Christos Douligeris and Aikaterini Mitrokotsa. 2004. DDoS Attacks and Defense Mechanisms: Classification and State-of-the-art. *IEEE Trans. on COMSNETS* 44, 5 (2004), 643–666.
[26] Nick Duffield, Carsten Lund, and Mikkel Thorup. 2003. Estimating Flow Distributions from Sampled Flow Statistics. In *Proc. of ACM SIGCOMM*.
[27] David Eppstein, Michael T Goodrich, Frank Uyeda, and George Varghese. 2011. What's the Difference? Efficient Set Reconciliation without Prior Context. *ACM SIGCOMM* 41, 4 (2011), 218–229.
[28] Cristian Estan and George Varghese. 2003. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. *ACM Trans. on Computer Systems* 21, 3 (2003), 270–313.
[29] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. 2000. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. *IEEE/ACM Trans. on Networking* 8, 3 (2000), 281–293.
[30] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *Proc. of ACM SIGCOMM*.
[31] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. 2017. Gray Failure: The Achilles' Heel of Cloud-Scale Systems. In *Proc. of ACM SIGOPS HotOS Workshop*.
[32] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. Sketchvisor: Robust Network Measurement for Software Packet Processing. In *Proc. of ACM SIGCOMM*.
[33] Qun Huang and Patrick PC Lee. 2014. LD-Sketch: A Distributed Sketching Design for Accurate and Scalable Anomaly Detection in Network Data Streams. In *Proc. of IEEE INFOCOM*.
[34] Qun Huang, Patrick PC Lee, and Yungang Bao. 2018. Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *Proc. of ACM SIGCOMM*.
[35] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. OmniMon: Re-architecting Network Telemetry with Resource Efficiency and Full Accuracy. In *Proc. of ACM SIGCOMM*.
[36] Srikanth Kandula and Ratul Mahajan. 2009. Sampling Biases in Network Path Measurements and What To Do About It. In *Proc. of ACM SIGCOMM*.
[37] Anurag Khandelwal, Rachit Agarwal, and Ion Stoica. 2019. Confluo: Distributed Monitoring and Diagnosis Stack for High-Speed Networks. In *Proc. of USENIX NSDI*.
[38] George Kollios, John W Byers, Jeffrey Considine, Marios Hadjieleftheriou, and Feifei Li. 2005. Robust Aggregation in Sensor Networks. *IEEE Trans. on Data Eng. Bull.* 28, 1 (2005), 26–32.
[39] Abhishek Kumar, Minho Sung, Jun Xu, and Jia Wang. 2004. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution. *ACM Trans. on SIGMETRICS Performance Evaluation Review* 32, 1 (2004), 177–188.
[40] Anukool Lakhina, Mark Crovella, and Christophe Diot. 2005. Mining Anomalies Using Traffic Feature Distributions. *ACM Trans. on SIGCOMM computer communication review* 35, 4 (2005), 217–228.
[41] Yuliang Li, Rui Miao, Mohammad Alizadeh, and Minlan Yu. 2019. DETER: Deterministic TCP Replay for Performance Diagnosis. In *Proc. of USENIX NSDI*.
[42] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. Flowradar: A Better Netflow for Data Centers. In *Proc. of ACM SIGCOMM*.
[43] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. LossRadar: Fast Detection of Lost Packets in Data Center Networks. In *Proc. of ACM CoNEXT*.
[44] Yang Liu, Wenji Chen, and Yong Guan. 2012. A Fast Sketch for Aggregate Queries over High-Speed Network Traffic. In *Proc. of IEEE INFOCOM*.
[45] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and General Sketch-based Monitoring in Software Switches. In *Proc. of ACM SIGCOMM*.
[46] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with Univmon. In *Proc. of ACM SIGCOMM*.
[47] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P Brighten Godfrey, and Samuel Talmadge King. 2011. Debugging the Data Plane with Anteater. *ACM Trans. on SIGCOMM Computer Communication Review* 41, 4 (2011), 290–301.
[48] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate Frequency Counts over Data Streams. In *Proc. of VLDB*.
[49] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proc. of Springer ICDT*.
[50] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proc. of ACM SIGCOMM*.
[51] Michael Mitzenmacher. 2002. Compressed Bloom Filters. *IEEE/ACM Trans. on Networking* 10, 5 (2002), 604–612.
[52] Vern Paxson. 1994. Empirically Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Trans. on Networking* 2, 4 (1994), 316–336.
[53] Vern Paxson and Sally Floyd. 1995. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Trans. on networking* 3, 3 (1995), 226–244.
[54] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A Dinda, Ming-Yang Kao, and Gokhan Memik. 2007. Reversible Sketches: Enabling Monitoring and Analysis over High-Speed Data Streams. *IEEE/ACM Trans. on Networking* 15, 5 (2007), 1059–1072.
[55] Danfeng Shan and Fengyuan Ren. 2017. Improving ECN Marking Scheme with Micro-burst Traffic in Data Center Networks. In *Proc. of IEEE INFOCOM*.
[56] Brent Stephens, Aditya Akella, and Michael Swift. 2019. Loom: Flexible and Efficient NIC Packet Scheduling. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*.
[57] Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. 2008. On Unbiased Sampling for Unstructured Peer-To-Peer Networks. *IEEE/ACM Trans. on Networking* 17, 2 (2008), 377–390.
[58] Lu Tang, Qun Huang, and Patrick PC Lee. 2019. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In *Proc. of IEEE INFOCOM*.
[59] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao. 2013. You Are How You Click: Clickstream Analysis for Sybil Detection. In *Proc. of Usenix Security Symposium*.
[60] Kyu-Young Whang, Brad T Vander-Zanden, and Howard M Taylor. 1990. A Linear-Time Probabilistic Counting Algorithm for Database Application. *ACM Trans. on Database Systems* 15, 2 (1990), 208–229.
[61] Sisi Xiong, Yanjun Yao, Qing Cao, and Tian He. 2014. kBF: A Bloom Filter for Key-Value Storage with an Application on Approximate State Machines. In *Proc.*

of IEEE INFOCOM.

[62] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. 2005. Profiling Internet Backbone Traffic: Behavior Models and Applications. *ACM Trans. on SIGCOMM Computer Communication Review* 35, 4 (2005), 169–180.

[63] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proc. of ACM SIGCOMM.*

[64] Tong Yang, Lingtong Liu, Yibo Yan, Muhammad Shahzad, Yulong Shen, Xiaoming Li, Bin Cui, and Gaogang Xie. 2017. SF-sketch: A Fast, Accurate, and Memory Efficient Data Structure to Store Frequencies of Data Items. In *Proc. of IEEE ICDE.*

[65] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. 2017. Pyramid Sketch: A Sketch Framework for Frequency Estimation of Data Streams. *Trans. on VLDB* 10, 11 (2017), 1442–1453.

[66] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. 2018. Cold Filter: A Meta-Framework for Faster and More Accurate Stream Processing. In *Proc. of ACM SIGMOD.*

[67] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-Level Telemetry in Large Datacenter Networks. In *Proc. of ACM SIGCOMM.*