energies

MDPI

Article

Efficient Sketch-Based Joint Heavy-Hitter and Super-Spreader Detection for Microgrid Communication Networks

Ruofei Wang [1,*], Xuekai Hu [1], Shiwei Xue [1], Rui Ma [1], Wenhao Li [2], Hao Xun [2] and Zhuo Li [2,*]

[1] State Grid Hebei Electric Power Research Institute, Shijiazhuang 050021, China; dyy_huxk@he.sgcc.com.cn (X.H.); dyy_xuesw@he.sgcc.com.cn (S.X.); dyy_mar@he.sgcc.com.cn (R.M.)
[2] School of Microelectronics, Tianjin University, Tianjin 300072, China; w_li0813@tju.edu.cn (W.L.); hxun@tju.edu.cn (H.X.)
* Correspondence: dyy_wangrf@he.sgcc.com.cn (R.W.); zli@tju.edu.cn (Z.L.)

Abstract

Cyberattacks such as DoS and DDoS can block or delay data communications in microgrid networks, thereby impairing the control of power electronic converters. These attacks motivate the measurement of heavy hitters and super-spreaders in microgrid network traffic. Both measurement tasks commonly rely on sketches, a class of probabilistic data structures. The two tasks nevertheless impose conflicting requirements: super-spreader detection ignores duplicate items in data stream statistics, while heavy hitter detection retains flows with many duplicate items. Consequently, an existing sketch-based scheme generally specializes in one task even though a data stream may contain both traffic classes. DualMeasure Sketch (DMS) is proposed to resolve this conflict. DMS distinguishes super-spreaders from heavy hitters dynamically and maintains them separately, allowing both to be detected accurately at the same time. CPU and OVS platform implementations are used for performance evaluation. Experimental comparison with the state-of-the-art sketches shows concurrent support for both tasks. On CAIDA19 with 1 MB memory, DMS improves super-spreader F1-score by up to 40% and reduces heavy-hitter error by up to 80% compared with dedicated state-of-the-art sketches. In joint deployment on CAIDA19 with 1 MB total memory, DMS improves throughput by about 12% over Combine Sketch.

Keywords: data streams; sketch; approximate stream processing; heavy hitters; super-spreaders; microgrid

1. Introduction

Microgrids are used to improve the reliability and resilience of electrical grids, facilitate the integration of distributed renewable resources, and provide electricity in areas not served by centralized infrastructure [1]. However, cyberattacks such as DoS and DDoS can block or delay data communications in microgrid networks, thereby impairing the control of power electronic converters [2].

These cybersecurity concerns motivate the measurement of heavy hitters and super-spreaders in microgrid network traffic. Heavy hitter detection isolates flows with abnormally high packet volumes, a capability that supports Denial of Service (DoS) attack detection [3]. Super-spreader detection identifies source addresses contacting numerous distinct destination addresses, therefore assists network administrators in recognizing Distributed Denial of Service (DDoS) attacks [4,5].

High accuracy, fast processing speed and small memory usage have made probabilistic data structures known as sketches common in approximate stream processing [6–8].

Check for updates

Academic Editor: Daniele D. Giusto

Received: 3 September 2026
Revised: 11 October 2026
Accepted: 15 October 2026
Published: 22 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Many sketch-based schemes now address the two detection tasks. For heavy hitter detection, representative high-performing schemes are MV Sketch [9], Elastic Sketch [10], and Spacesaving [11]; accurate and efficient super-spreader detection is provided by Spread Sketch [12], Vector Bloom Filter [13], and FastSketch [14].

A single data stream may contain both super-spreaders and heavy hitters. For instance, the network nodes (e.g., end-host and switches) may simultaneously suffer from DoS and DDoS attacks (i.e., prone to be heavy hitters and super-spreaders) [15]. Sketch data structures, as a promising type of traffic measurement method, is often used for detecting heavy hitters and super-spreaders, which are statistical traffic features. These features though cannot by themselves determine whether a source is malicious but plays crucial role in assisting security analysis, such as DoS/DDoS investigation. However, accommodating both measurement objectives within one existing sketch-based scheme remains challenging. The key challenges are as follows:

Firstly, consider a network stream in which a source with high destination cardinality is treated as a super-spreader. Super-spreader measurement generally discards repeated source-destination pairs, for example through a Bloom filter, because only distinct destinations contribute to this cardinality. Heavy hitter detection instead identifies flows with large packet counts and therefore must retain repeated pairs. A sketch optimized for either task may consequently discard evidence required by the other. Running a dedicated sketch for each task avoids this conflict but increases switch memory consumption and computation overhead.

Secondly, real-world streams exhibit highly skewed and time-varying distributions of flow sizes and destination cardinalities [12,16]. Low-cardinality sources and small flows dominate the traffic, whereas super-spreaders and heavy hitters account for only a small fraction. Hash collisions introduced by numerous low-cardinality sources can bias super-spreader cardinality estimates and reduce detection accuracy. Similarly, numerous small flows also reduce heavy hitter detection accuracy.

To meet these challenges, this paper introduces a new sketch called DualMeasure Sketch (DMS), which can simultaneously detect super-spreaders and heavy hitters with high accuracy, throughput, and memory efficiency. DMS maintains the two traffic classes separately and distinguishes them dynamically during updates, thereby limiting interference from low-cardinality sources and small flows. Consequently, DMS performs both detection tasks while mitigating the effect of skewed traffic on measurement accuracy. The main contributions are listed below:

- We introduce DualMeasure Sketch (DMS), which stores super-spreaders and heavy hitters separately and supports both detection tasks without deploying a dedicated sketch for each.
- We design update algorithms that distinguish super-spreaders from heavy hitters while suppressing interference from low-cardinality sources and small flows. This mechanism enables accurate detection of both traffic classes with limited memory.
- We analyze DMS theoretically with respect to space complexity, update time complexity, the probability of hash collisions and error bounds.
- We implement DMS on CPU and OVS platform, where detection accuracy and throughput are evaluated. The resulting system performs both tasks and, relative to state-of-the-art dedicated sketch-based schemes, improves F1-score by 10% for heavy hitter detection and 22% for super-spreader detection while raising throughput by more than 50%.

The paper proceeds with the problem definitions and related work in Section 2. Section 3 then describes the DMS design, and Section 4 establishes its theoretical properties. Section 5 presents the performance evaluation and discussion; DMS performance in

OVS platform is examined separately in Section 6. Section 7 closes with conclusions and future work.

2. Backgrounds and Related Works

Formal definitions of the considered problems are given first, followed by a survey of existing super-spreader and heavy hitter detection schemes.

2.1. Problem Statement

Data Stream: An ordered sequence $(e_1, e_2, \ldots e_N)$ of N items defines data stream S, where N is the total item count and the same item may appear repeatedly in S. A *flow* comprises every item sharing an identical value. Each item is written as (x, y), where the flow label is x and the element is y. When the stream carries packets, one packet forms an item, with its source address mapped to x and destination address mapped to y.

Super-Spreaders Detection: The connection associated with flow label x equals the count of distinct elements y paired with x. If that count exceeds $T_S \cdot M$, x is classified as a super-spreader, where T_S denotes a predefined threshold.

Heavy Hitters Detection: For an item (x, y), classification as a heavy hitter occurs when the stream frequency is greater than $T_H \cdot M$; T_H is defined in advance.

2.2. Super-Spreader Detection

Sketch-based super-spreader detectors combine measurement accuracy with high throughput and low memory consumption. Representative designs include Fast Sketch (FAST) [14], Vector Bloom Filter (VBF) [13], Open Sketch (OPEN) [17], Count Degree Sketch (CDS) [18], The Random Aging Streaming Filters (RASF) [19], Spread Sketch (SS) [12], Super kJoin Sketch (SKS) [20], Extended Sketch (ES) [16]. FAST formulates super-spreader detection as channel coding and compresses the encoded flow labels of candidate super-spreaders. VBF partitions each flow label into eight components and accelerates updates with bit-extraction hash functions. OPEN couples CM sketch with a bitmap to improve super-spreader detection accuracy. CDS maps super-spreaders with hash functions derived from the Chinese Remainder Theorem (CRT) and reconstructs their flow labels through CRT. RASF randomly initialize columns when a block is nearly full [19], allowing super hosts to be identified without excessive memory use, though with reduced accuracy. SS estimates flow label connections from binary hash strings and gives retention priority to labels with high element cardinality. SS is proposed for network-wide super-spreader detection. The structure consumes considerable memory resources because each block needs to maintain a counter. The concept of generalized sketch families for network traffic measurement offers a flexible framework for analyzing traffic patterns, but requires further optimization for anomaly detection in super host scenarios [21]. Based on this, SKS was proposed for flow spread estimation, offering improved accuracy and efficiency for cardinality estimation. ES expands its structure as flow label connections increase, allowing the sketch size to adapt dynamically.

2.3. Heavy Hitter Detection

Accurate heavy hitter detection can likewise be achieved with sketches. The relevant methods considered here are Count-min Sketch with a min-heap (CMH) [22], UnivMon (UM) [23], Spacesaving [11], HashPipe (HP) [24], Elastic Sketch (ES) [10], MV Sketch (MV) [9], Pyramid Sketch (PS) [25], Bloom Sketch (BS) [26], Diamond Sketch (DS) [27], One Sketch (OS) [28], Funnel Sketch (FS) [29]. CMH combines Count-min Sketch for flow-size estimation with a min-heap that retains heavy hitters, thereby improving detection accuracy. UM partitions the input through universal sampling and uses multiple sketches to track heavy hitters in the resulting sub-streams. Within a fixed-size structure, Spacesaving

associates estimators with candidate heavy hitters and applies adaptive replacement to remove infrequent items, thereby maintaining detection accuracy. HP uses a two-stage update mechanism to retain heavy hitters within the pipeline and improve heavy-hitter detection accuracy. ES places heavy hitters and infrequent items in separate structures, reducing interference from infrequent items during detection. High accuracy and throughput are obtained in MV Sketch by tracking heavy hitters with the Boyer–Moore majority vote algorithm (MJRTY). PS sets two flag bits in the parent counter to record the overflow information of the left or right child counter. BS consists of L counter array and L-1 bloom filter. Each counter array stores a portion of flow size. DS is composed of three parts, which are increment part, carry part and deletion part. Increment part contains multiple layers for recording flows with different sizes, carry part records the depth of overflow, and deletion part supports the delete operation. OS replaces the CM in ES with TowerCU and modifies the replacement strategy, thereby improving the measurement accuracy. FS utilizes a funnel-shaped architecture to enable fine grained classification of elephant and mouse flows.

2.4. Summary

Existing sketch-based schemes perform well, but each is designed for only one of the two detection tasks. Because a stream may contain both super-spreaders and heavy hitters, a single existing sketch cannot readily cover both cases, as discussed in Section 1. Running separate sketches for the two tasks incurs additional memory and computation overhead.

DMS addresses this limitation by dynamically separating the two traffic classes and maintaining them independently. The resulting structure supports accurate joint detection of super-spreaders and heavy hitters under a small memory budget.

3. DualMeasure Sketch Design

The presentation begins with the DMS design goals and data structure, then turns to the update and query operations. Table 1 summarizes the principal notation for the subsequent discussion.

Table 1. Notation in this paper .

Notation	Meaning
$e_i = (x_i, y_i)$	The i-th item in data stream
$H[k]$	The k-th bucket in *hitter layer*
$H[k].key$	The flow ID of candidate heavy hitter
$H[k].v$	The size of the candidate heavy hitter
$H[k].p$	The weight of the candidate heavy hitter being evicted
$S(i,j)$	The j-th bucket in i-th row of *spreader layer*
$S(i,j).B$	The bitmap for flow label connections estimation
$S(i,j).N$	The size counter for heavy hitter that has hash conflicts
σ	The replacement threshold in *hitter layer*
$h_l(\cdot)$	The hash function in *hitter layer*
$h'_i(\cdot)$	The hash function in *spreader layer*
w_h	Number of buckets in *hitter layer*
r	Number of rows in *spreader layer*
w_s	Number of columns in *spreader layer*
H_{min}	The minimum value of m mapped $H[k].v$
N_{min}	The minimum value of r mapped $S(i,j).N$
M	The total number of items
T_S	The threshold for super-spreader detection
T_H	The threshold for heavy hitter detection

DMS aims for the following design goals:

- **Dual detection:** A single DMS instance performs both super-spreader and heavy hitter detection, eliminating separate dedicated sketches.
- **High detection accuracy:** DMS preserves accurate estimates for both super-spreaders and heavy hitters.
- **Fast processing speed:** DMS limits hash computation and memory accesses to process incoming items efficiently.
- **Small and static memory usage:** A compact static structure avoids the management overhead associated with dynamic memory allocation.

3.1. Data Structure

Figure 1 depicts the two components of the DMS data structure: a *hitter layer* and a *spreader layer*. A total of w_h buckets form the *hitter layer*. For $0 \leq k < w_h$, bucket k is represented by $H[k]$, and three fields are carried by $H[k]$: (1) the candidate heavy hitter flow ID in $H[k].key$; (2) $H[k].v$, representing the candidate heavy hitter size; and (3) $H[k].p$ for that candidate's eviction weight. Pairwise independence is required among the m hash functions $h_l(\cdot)$ used by *hitter layer*, with $0 \leq l < m$.

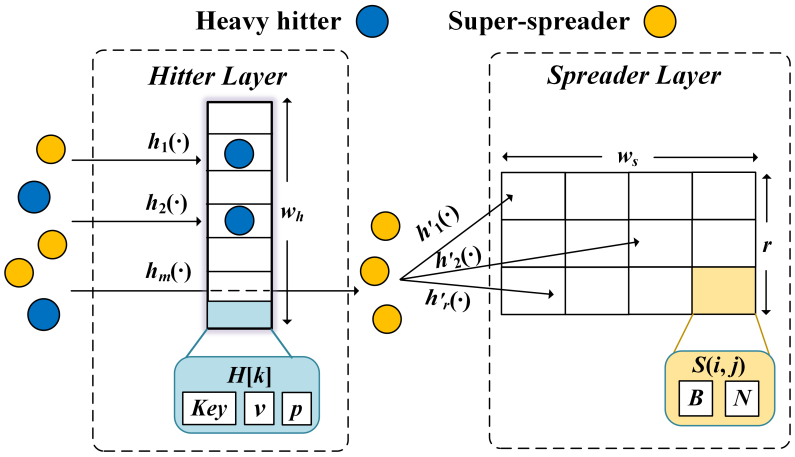


Figure 1. The data structure of DualMeasure Sketch.

The *spreader layer* contains r rows, each comprising w_s buckets. Bucket $S(i,j)$ occupies column j of row i, with $0 \leq i < r$ and $0 \leq j < w_s$. Every bucket $S(i,j)$ provides the following two fields: (1) $S(i,j).B$ serves as the flow label connections bitmap. The value $S(i,j).B[t]$ corresponds to bit t in $S(i,j).B$, with $0 \leq t < n$; (2) $S(i,j).N$, a size counter for heavy hitters involved in hash collisions in *hitter layer*. The *spreader layer* additionally uses r pairwise-independent hash functions $h'_i(\cdot)$, indexed by $0 \leq i < r$.

3.2. Basic Operation

Two operations define DMS behavior: (1) an incoming item e_i is incorporated through Update; and (2) at epoch completion, Query returns the estimated super-spreaders and heavy hitters.

Update Operation: The processing steps for a DMS update are specified in Algorithm 1. Every counter begins an epoch at 0. For an incoming item $e = (x, y)$, the hash functions $h_l(e)$ map it to m buckets $H[k]$ in *hitter layer*, and each $H[k]$ is examined sequentially, where $0 \leq k < w_h$, $0 \leq l < m$. The subsequent action depends on one of three cases.

Case 1: $H[k].v > 0$ and $H[k].key = e$. This match indicates that $H[k]$ contains item e; the update therefore adds 1 to $H[k].v$ (Line 1–9).

Case 2: $H[k].v = 0$. An available empty bucket receives e in $H[k].key$, followed by an addition of 1 to $H[k].v$ (Line 10–16).

Case 3: $H[k].v > 0$ and $H[k].key \neq e$. Here, e collides in every mapped bucket $H[k]$. DMS identifies the minimum counter value H_{min} among the m mapped $H[k].v$ entries and increments the $H[k].p$ associated with the smallest $H[k].v$ (Line 17–26). The replacement strategy in Algorithm 2 prioritizes retaining heavy hitters in *hitter layer* to preserve detection accuracy.

Algorithm 1 Update Procedure

1: **Input:** Receiving item $e = (x, y)$
2: **Procedure** Update (e)
3: Initialization: *Hitter layer*, *Spread layer*
4: **for** i = 0 to $m - 1$ **do**
5: **if** $H[h_i(e)].key == e$ **then**
6: $H[h_i(e)].v++$;
7: **return**;
8: **end**
9: **end**
10: **for** i = 0 to $m - 1$ **do**
11: **if** $H[h_i(e)].v == 0$ **then**
12: $H[h_i(e)].key \leftarrow (x, y)$;
13: $H[h_i(e)].v++$;
14: **return**;
15: **end**
16: **end**
17: $H_{min} \leftarrow$ the minimum value of all mapped $H[k].v$;
18: **for** i=0 to $m - 1$ **do**
19: **if** $H[h_i(e)].v == H_{min}$ **then**
20: $H[h_i(e)].p++$;
21: let $H[min]$ donates as $H[h_i(e)]$;
22: **break;**
23: **end**
24: **end**
25: **Replace** (e);
26: **End Procedure**

Replacement Strategy Design: Three conditions govern the replacement strategy.

Case 1: If $H[k].p \geq H_{min} \cdot \sigma$ (σ is a predefined threshold), DMS first hashes e into *spreader layer* and obtains the minimum value N_{min} among the r mapped $S(i, j).N$, where $0 \leq i < r$, $0 \leq j < w_s$. Suppose the bucket $H[k]$ with H_{min} stores $(e_1, H_{min}, H[k].p)$, then set the bucket to $(e, N_{min}, 0)$ and evict item e_1 to *spreader layer*: increment the mapped r $S(i, j).N$ by H_{min} (Line 1–11).

Case 2: If $H[k].p < H_{min} \cdot \sigma$, DMS hashes e into r buckets $S(i, j)$ in *spreader layer* by $h'_i(x)$, where $0 \leq i < r$. Then, each bitmap maps e into $S(i, j).B[t]$ by $h_l(e)$. In this case, for the bucket $S(i, j)$ where $S(i, j).B[t] = 1$, $S(i, j).N$ is increased by 1, and set the mapped bit 0 in other $S(i, j).B$ to 1 (Line 12–18).

Case 3: If $H[k].p < H_{min} \cdot \sigma$ and all the mapped $S(i, j).B[t]$ are 1, DMS compares N_{min} and H_{min}. If $N_{min} > H_{min}$, the bucket $H[k]$ storing $(e_1, H_{min}, H[k].p)$ is set to $(e, N_{min}, H[k].p)$ and all $S(i, j).N$ in mapped $S(i, j)$ are set to 0 (Line 19–27).

Every update and replacement case is visualized in Figure 2. The associated example fixes σ = 2, m = 2, $r = 3$, $w_h = 5$ and $w_s = 4$.

Algorithm 2 Replace Procedure

Input: Receiving item $e = (x, y)$
Procedure Replace (e)
if $H[k].p \geq H_{min} \cdot \sigma$ **then**
    $N_{min} \leftarrow min_{0 \leq i < r}\{S(i, h_i'(x)).N\}$;
    $(a, b) \leftarrow H[min].key$;
    $H[min] \leftarrow (e, N_{min}, 0)$;
    for $i = 0$ to $r - 1$ **do**
        $S[i, h_i'(a)].N+ = H_{min}$;
    end
    return;
end
for $k = 0$ to $r - 1$ **do**
    if $S(k, h_k'(x)).B[h_1(e)] == 1$ **then**
        $S(k, h_k'(x)).N++$;
    else
        $S(k, h_k'(x)).B[h_1(e)] \leftarrow 1$;
    end
end
if all $S(k, h_k'(x)).B[h_1(e)]$ are 1 **then**
    if $H_{min} < N_{min}$ **then**
        $H[min] \leftarrow (e, H_{min}, H[k].p)$;
        for $i = 0$ to $r - 1$ **do**
            $S(i, h_i'(x)).N \leftarrow 0$;
        end
    end
end
End Procedure

Case 1: Figure 2a depicts the arrival of $e_1 = (x_1, y_1)$. Within *hitter layer*, $H[0].key$ matches e_1, leading to an addition of 1 to $H[0].v$.

Case 2: In Figure 2b, item $e_4 = (x_4, y_4)$ matches neither $H[2]$ nor $H[4]$. Because $H[4]$ is empty, $(e_4, 1, 0)$ is inserted into $H[4]$.

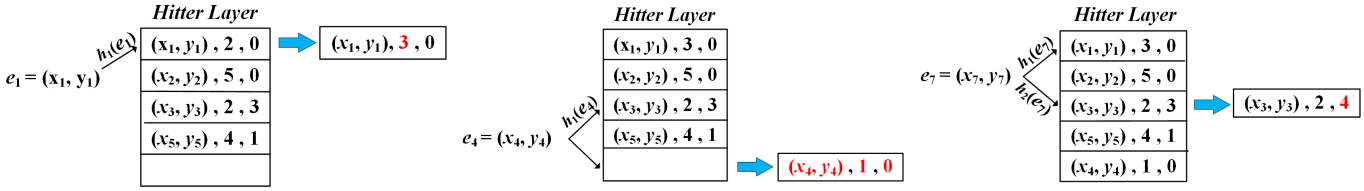
Case 3: In Figure 2c, the arriving item $e_7 = (x_7, y_7)$ is absent from $H[0]$ and $H[2]$, and neither bucket is empty. The minimum $H_{min} = 2$ is obtained from $H[2].v$, after which $H[2].p$ is incremented by 1. After that, DMS implements the replacement strategy.

Examples of Replacement Strategy:

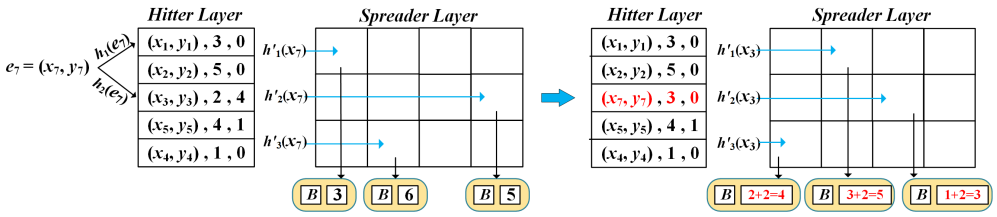
Case 1: As shown in Figure 2d, at this moment, $H[2].p = 4 \geq H_{min} \cdot \sigma = 2 \times 2 = 4$. DMS inserts e_7 into $S(0, 0)$, $S(1, 3)$ and $S(2, 1)$, obtains $N_{min} = 3$, and replaces $H[2]$ with $(e_7, 3, 0)$. Then hash the item e_3 into *spreader layer* and increase $S(0, 1).N$, $S(1, 2).N$ and $S(2, 0).N$ by $N_{min} = 2$.

Case 2: The arrival of item $e_6 = (x_6, y_6)$ is depicted in Figure 2e, $H[k].p = 1 < H_{min} \cdot \sigma = 4 \times 2 = 8$. DMS hashes e_6 into $S(0, 2)$, $S(1, 1)$ and $S(2, 3)$ and each bucket independently update its bitmap: $S(0, 2).B[1] = 0$, then set it to 1; $S(1, 1).B[1] = 1$, then increase $S(1, 1).N$ by 1; $S(2, 3).B[1] = 0$, then set it to 1.

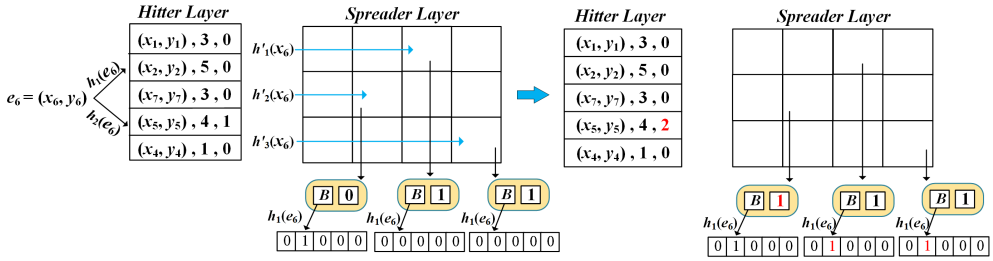
Case 3: The arrival of item $e_8 = (x_8, y_8)$ appears in Figure 2f, all mapped bits in bitmap are 1 and $N_{min} = 2 + 1 > H_{min} = 1$, then set the bucket $H[4]$ to $(e_8, 3, 0)$ and set $S(0, 3).N$, $S(1, 0).N$ and $S(2, 2).N$ to 0.



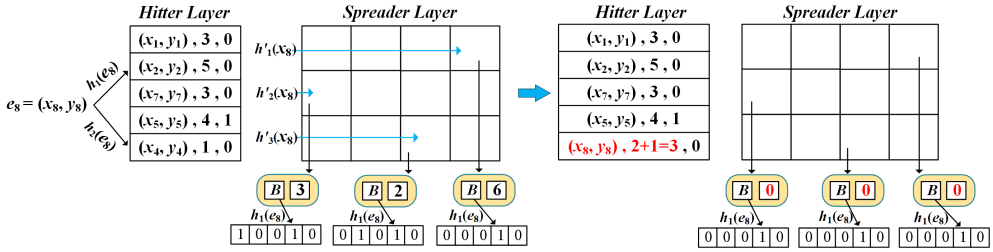
(**a**) Case 1 in Update Operation (**b**) Case 2 in Update Operation (**c**) Case 3 in Update Operation



(**d**) Case 1 in Replacement Strategy



(**e**) Case 2 in Replacement Strategy



(**f**) Case 3 in Replacement Strategy

Figure 2. The update operation and replacement strategy of DMS.

Query Operation: DMS provides separate query procedures for super-spreaders and heavy hitters.

- **Super-spreader Query:** Algorithm 3 gives the super-spreader query procedure. At epoch end, $h'_i(x)$ assigns flow label x to r spreader-layer buckets $S(i,j)$, where $0 \leq i < r, 0 \leq j < w_s$. Linear counting estimates the connections recorded by every $S(i,j).B$ [30], and the smallest estimate is selected. Flow label x is reported as a super-spreader when this value is greater than $T_S \cdot M$, with M denoting the total item count and T_S a predefined threshold.
- **Heavy hitter Query:** Algorithm 4 defines this query procedure. Once an epoch finishes, every hitter-layer bucket is scanned. An item is reported when the examined $H[k].v$ is greater than $T_H \cdot M$; the comparison uses the item total M and predefined threshold T_H, while the corresponding $H[k].v$ supplies the estimated size.

Algorithm 3 Super-spreaders Query Procedure

1: **Input:** Flow label x
2: **Procedure** Query for super-spreaders
3: $|S(i,j).B| \leftarrow$ the estimated connections in $S(i,j).B$;
4: **if** $min_{0\le i<r}\{|S(i,h_i'(x)).B|\} > T_S \cdot M$ **then**
5: report x as a super-spreader;
6: **end**
7: **End Procedure**

Algorithm 4 Heavy Hitters Query Procedure

1: **Procedure** Query for heavy hitters
2: **for** $i = 0$ to $w_h - 1$ **do**
3: **if** $H[i].v > T_H \cdot M$ **then**
4: report $H[i].key$ as a heavy hitter;
5: **end**
6: **end**
7: **End Procedure**

4. Theoretical Analysis

The analysis covers four properties: storage complexity, update/query time, *hitter layer* hash collisions, and the estimation-error bounds for flow label connections.

4.1. Space Complexity and Time Complexity

For the DMS *hitter layer*, w_h specifies how many buckets are available and m specifies how many hash functions are used. In the *spreader layer*, r gives the row count, w_s gives the column count, and every bucket contains the n-bit bitmap $S(i,j).B$.

Theorem 1. *The space complexity of DMS is* $O(w_h \cdot \log q + n \cdot r \cdot w_s)$.

Proof. Every hitter-layer bucket allocates one $\log q$-bit candidate heavy hitter together with two counters; all such buckets therefore occupy $O(w_h \cdot \log q)$ space. A spreader-layer bucket instead contains the n-bit bitmap and one counter, giving $O(n \cdot r \cdot w_s)$ space over that layer. Combining the layers yields the DMS bound $O(w_h \cdot \log q + n \cdot r \cdot w_s)$. □

Theorem 2. *Processing an item has update time complexity* $O(m + r)$. *Heavy hitter and super-spreader queries require* $O(w_h)$ *and* $O(r)$ *time, respectively.*

Proof. Updating one item requires at most $m + r$ hash operations (i.e., m hash operations in *hitter layer* and r hash operations in *spreader layer*). A heavy hitter query scans all w_h buckets in *hitter layer*, resulting in $O(w_h)$ time. A super-spreader query performs r hash operations to obtain flow label connections from *spreader layer*, and therefore requires $O(r)$ time. □

4.2. Heavy Hitter Collision Rate in Hitter Layer

Consider an epoch containing H heavy hitters in the data stream.

Theorem 3. *The probability that no hash conflicts occur between all heavy hitters of hitter layer is:*

$$P = e^{-\frac{Hm(Hm-1)}{2w_h}} \tag{1}$$

Proof. Each hash function is assumed to map every heavy hitter independently and uniformly across the buckets. Consequently, an item is assigned to any particular bucket with probability $\frac{1}{w_h}$. Each heavy hitter consequently has w_h^m possible bucket combinations. With

$K = H \times m$, the H frequent items admit w_h^K assignments, of which $\frac{w_h!}{(w_h-K)!}$ are collision-free, therefore, the probability of no hash collision is:

$$P = \frac{w_h!}{(w_h - K)! w_h^K} \tag{2}$$

When the hitter-layer bucket count w_h is much larger than the heavy hitter count H, so that $w_h \gg K$, the no-collision probability admits the approximation:

$$P = \prod_{i=0}^{K-1} \frac{w_h - i}{w_h} = \prod_{i=0}^{K-1} (1 - \frac{i}{w_h}), \tag{3}$$

taking the logarithm yields:

$$\ln P = \sum_{i=0}^{K-1} \ln(1 - \frac{i}{w_h}) \tag{4}$$

Applying $\ln(1 - x) \approx -x$ for $|x| \ll 1$, which holds here when $w_h \gg K$, gives:

$$\ln P \approx \sum_{i=0}^{K-1} (-\frac{i}{w_h}) = -\frac{i}{w_h} \sum_{i=0}^{K-1} i, \tag{5}$$

The remaining finite sum is:

$$\sum_{i=0}^{K-1} i = \frac{(K-1)K}{2} \tag{6}$$

Substituting this sum into Formula (3) yields the following no-collision probability for the heavy hitters in *hitter layer*:

$$P \approx e^{-\frac{K(K-1)}{2w_h}} = e^{-\frac{Hm(Hm-1)}{2w_h}} \tag{7}$$

□

4.3. Error Bounds of Connection Estimation in Spreader Layer

Consider one fixed flow. Its actual connection value is c, whereas $\hat{c}$ is the final estimate. The r mapped $S(i, j)$ buckets individually produce $\hat{c_1}, ..., \hat{c_r}$; the approximation parameter is ϵ, and the sketch error probability is δ.

Theorem 4. *For any flow label, the estimated connection value $\hat{c}$ satisfies the condition with a probability of at least $1 - \delta$:*

$$c - \epsilon c \leq \hat{c} \leq c + \epsilon c \tag{8}$$

Proof. Let the bitmap length n be sufficiently large. In a bucket bitmap, u_i describes the probability of an unoccupied position; its binomial distribution has the expectation and variance below:

$$E[u_i] = ne^{-\frac{c}{n}} \tag{9}$$

$$\mathrm{Var}(u_i) \approx ne^{-\frac{c}{n}} \left(1 - e^{-\frac{c}{n}}\right) \tag{10}$$

This distribution gives the following expectation and relative bias for $\hat{c}$:

$$E[\hat{c_i}] \approx c + \frac{e^t - t - 1}{2}, \tag{11}$$

$$\text{Bias}(\hat{c}_i) = \frac{E[\hat{c}_i] - c}{c} = \frac{e^t - t - 1}{2c}, \tag{12}$$

The corresponding variance of $\hat{n}$ is:

$$\text{Var}(\hat{n}_i) \approx m(e^t - t - 1), \tag{13}$$

where $t = -\frac{c}{n}$.

A normal approximation bounds the tail probability as follows:

$$P\left(\left|\frac{\hat{c}_i - c}{c}\right| > \epsilon\right) \leq 2\exp\left(-\frac{\epsilon^2 c^2}{2n(e^t - t - 1)}\right) \triangleq 2e^{-K}, \tag{14}$$

where $K = \frac{\epsilon^2 c^2}{2\sigma^2}$, $\sigma^2 = \text{Var}(\hat{c}_i)$.

The per-bucket connection estimates $\hat{c_1}, ..., \hat{c_r}$ are independent and identically distributed. Because their minimum $\hat{c}$ has an asymmetric error distribution, the two tails satisfy:

$$P(\hat{c} < c - \epsilon c) \leq \sum_{i=1}^{r} P(\hat{c}_i < c - \epsilon c) \leq 2re^{-K}, \tag{15}$$

$$P(\hat{c} > c + \epsilon n) = \prod_{i=1}^{r} P(\hat{c}_i > c + \epsilon c) \leq \left(e^{-K}\right)^r = e^{-rK}, \tag{16}$$

The sum of these tail bounds provides the overall error probability:

$$P\left(\left|\frac{\hat{c} - c}{c}\right| > \epsilon\right) \leq 2re^{-K} + e^{-rK} \tag{17}$$

When t is less than 0.1, $e^t - t - 1 \approx \frac{t^2}{2}$, substitute K and simplify the expression:

$$\delta(\epsilon) \approx 2re^{-\epsilon^2 n} + e^{-r\epsilon^2 n}, \tag{18}$$

setting $\delta(\epsilon) = \delta$ allows us to solve for ϵ, which gives $\epsilon \sim \sqrt{\frac{\ln r}{n}}$. Therefore, it can be concluded that:

$$P\left(\left|\frac{\hat{c} - c}{c}\right| \leq \epsilon\right) \geq 1 - (2re^{-K} + e^{-rK}) = 1 - \delta \tag{19}$$

□

The analysis provides conservative upper bounds on the estimation error under hash collisions. The collision probability in Theorem 3 is the probability that all heavy-hitter hash placements are collision-free. Algorithms 1 and 2 explicitly handle collisions and perform replacement, which further reduce the actual error. Therefore, the derived bounds are interpreted as worst-case bounds, and the actual error is expected to be lower.

5. Experimental Results

This section reports measurements obtained from real-world data streams. Even when memory is limited, DMS attains higher accuracy in super-spreader and heavy hitter detection than state-of-the-art sketch-based schemes.

5.1. Experimental Setup

Testbed: Experiments run under Ubuntu 18.04.6 LTS on a workstation containing 64 GB RAM and an Intel (R) Core (TM) i9-10920X CPU clocked at 3.50 GHz.

Datasets: CAIDA16, CAIDA18, and CAIDA19 [31] serve as the evaluation datasets. CAIDA captured them on 10 GigE backbone links in 2016, 2018, and 2019, respectively; CAIDA19 has the greatest skewness, whereas CAIDA16 has the least. CAIDA traces are highly skewed, and contain mixed small/low-cardinality flows and a few heavy hitters and super-spreaders, similar to microgrid traffic. Since no public microgrid dataset is available, CAIDA traces serve as representative datasets. Each dataset contains 60 one-second intervals. For each experiment, 10 intervals were randomly selected, and this random selection was independently repeated three times. The reported results are the average over the three independent repetitions.

Metrics: Performance is quantified using the following metrics.

- *Precision Rate (PR):* Among all reported super-spreaders/heavy hitters N_d, the share corresponding to correct detections N_{td}, namely

$$Precision = \frac{N_{td}}{N_d} \tag{20}$$

- *Recall Rate (RR):* Correct detections N_{td} expressed as a fraction of all actual super-spreaders/heavy hitters N_r, namely

$$Recall = \frac{N_{td}}{N_r} \tag{21}$$

- *F1-score:* One measure formed by taking the harmonic mean of Precision and Recall,

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{22}$$

- *Average Relative Error (ARE):* The mean of the relative errors separating estimated flow label connections/item sizes from the actual quantities:

$$ARE = \frac{1}{n} \sum_{i=1}^{n} \frac{|\hat{V}_i - V_i|}{V_i}, \tag{23}$$

  Here, n counts the flow labels/items, $\hat{V}_i$ is the estimated flow label connections/item size, and V_i is its actual counterpart.
- *Throughput:* The number of incoming items processed each second (Mpps).

Measurement Tasks: DMS versatility is assessed through two tasks on the datasets described above.

- *Super-spreader Detection:* For total item count M and threshold T_S, an epoch-end flow label is classified as a super-spreader if its estimated connections are greater than $T_S \cdot M$. The baselines used to evaluate DMS on this task are OPEN, Fast, CDS, VBF, and SS.
- *Heavy hitter Detection:* Classification occurs at epoch completion: an estimated item size above $T_H \cdot M$ identifies a heavy hitter, with T_H serving as the threshold. For this task, the baselines for DMS performance are Spacesaving, MV, UM, HP, and CMH.

Methodology: Every evaluated scheme shares a C++ implementation basis. The DMS configuration sets m = 2, r = 3, and n = 284 bits for the bitmap, while allocating w_h as one-half of w_s.

Following the parameter recommendations in the original papers, OPEN uses 3, 1, and 5 hash functions for Count-min Sketch, bitmap, and reversible Sketch, respectively, with a 98-bit bitmap. SS uses a 438-bit bitmap and 4 rows. UM is configured with 14

levels, CMH with a heap capacity of 4096, MV with 4 rows, and HP with 6 table stages. A source-destination address pair constitutes one item. The super-spreader and heavy hitter thresholds T_S and T_H are 0.01% and 0.05%, respectively. All schemes use BoBhash [32] as their hash function.

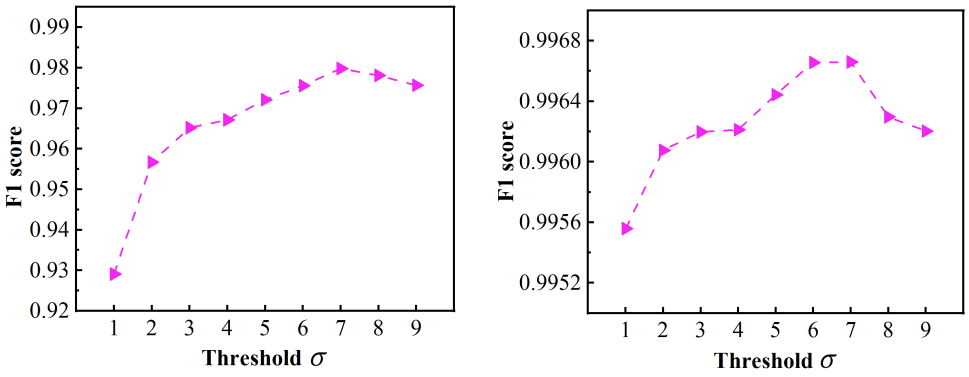
5.2. Results

The parameters σ and w_s/w_h are selected through experiments on CAIDA19. CAIDA16 and CAIDA18 then serve as independent test sets. The parameter was selected based on a trade-off between the performance of super-spreader and heavy hitter detection.

Impact of replacement threshold σ: In *hitter layer*, σ controls heavy hitter replacement and therefore affects detection accuracy. Selecting an appropriate σ is therefore important for DMS. The sensitivity study varies σ from 1 to 9 and measures the F1-score for each σ on CAIDA19 with the total DMS memory fixed at 1 MB.

Figure 3a shows that the F1-score increases over σ values from 1 to 7, attaining its maximum at σ = 7. Further growth in σ causes the F1-score to decline. Thus, σ = 7 is optimal for super-spreader detection.

Figure 3b shows how σ affects heavy hitter detection. The F1-score increases as σ changes from 1 to 6, remains at its maximum as σ varies from 6 to 7, and then drops sharply as σ increases from 7 to 9. A larger σ raises the barrier for heavy hitters to replace small items, thereby reducing F1-score. Based on these results, the replacement threshold σ is set to 7.



(**a**) Super-spreader Detection (**b**) Heavy Hitter Detection

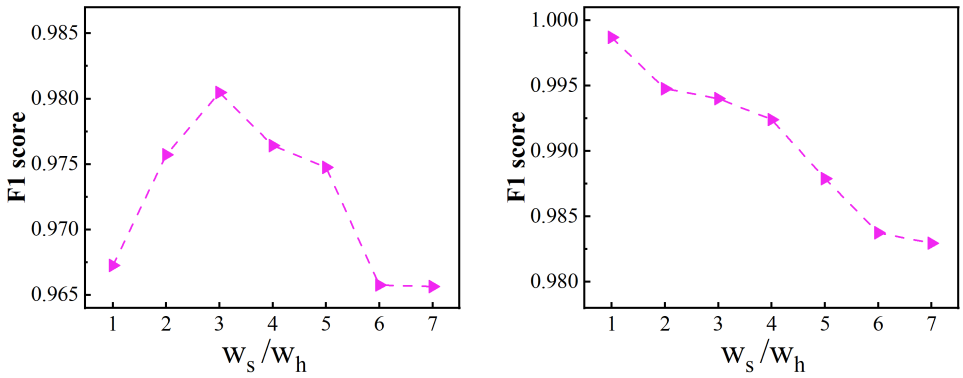
Figure 3. F1-score at different thresholds σ.

Impact of w_s/w_h: The quantities w_s and w_h denote the buckets per spreader-layer row and the hitter-layer bucket count, respectively. The ratio w_s/w_h controls the bucket allocation between the two layers and therefore affects both detection tasks. Selecting an appropriate w_s/w_h is therefore important. Its sensitivity is evaluated by varying w_s/w_h from 1 to 7 on CAIDA19, measuring the DMS F1 score for each w_s/w_h, and fixing the total memory at 1 MB.

Figure 4a indicates a rising F1-score while w_s/w_h moves from 1 to 3, with the peak occurring at w_s/w_h = 3. Further increases in w_s/w_h produce a continuing decline in F1-score. Hence, 3 is the optimal w_s/w_h value for super-spreader detection.

The heavy hitter F1 score for w_s/w_h is shown in Figure 4b. As w_s/w_h increases, the F1 score decreases overall. When w_s/w_h is 1, the F1 score reaches its maximum; for w_s/w_h values of 2 or 3, it remains high. Therefore, the suitable range of w_s/w_h is 1-3. Considering both detection tasks, w_s/w_h is set to 3, and the observed robustness supports using this ratio across different datasets.

Super-spreader Detection: Across all datasets in Figure 5, OPEN attains the highest precision, whereas its recall stays below 70%. Its bitmap improves precision by filtering low-cardinality flow labels and thereby constraining overestimation. However, probabilistic item sampling prevents complete capture and recording of super-spreader information, thereby lowering recall. The DMS replacement strategy instead yields a recall rate near 100% on CAIDA16 with 0.5 MB memory; Figure 5b shows that this value is 29% above OPEN.



(**a**) Super-spreader Detection (**b**) Heavy Hitter Detection

Figure 4. F1-score at different w_s/w_h.

VBF, CDS, and FAST keep flow labels regardless of whether their element cardinalities are low or high; collisions between low-cardinality labels and super-spreaders therefore become more frequent. Precision falls when these labels are incorrectly reported as super-spreaders in Figure 5a,e,i. Through dynamic separation of super-spreaders and heavy hitters, DMS suppresses the collisions and produces the much lower ARE observed in Figure 5d,h,l. At 1 MB memory, Figure 5l specifically gives a 50.15% ARE reduction relative to OPEN.

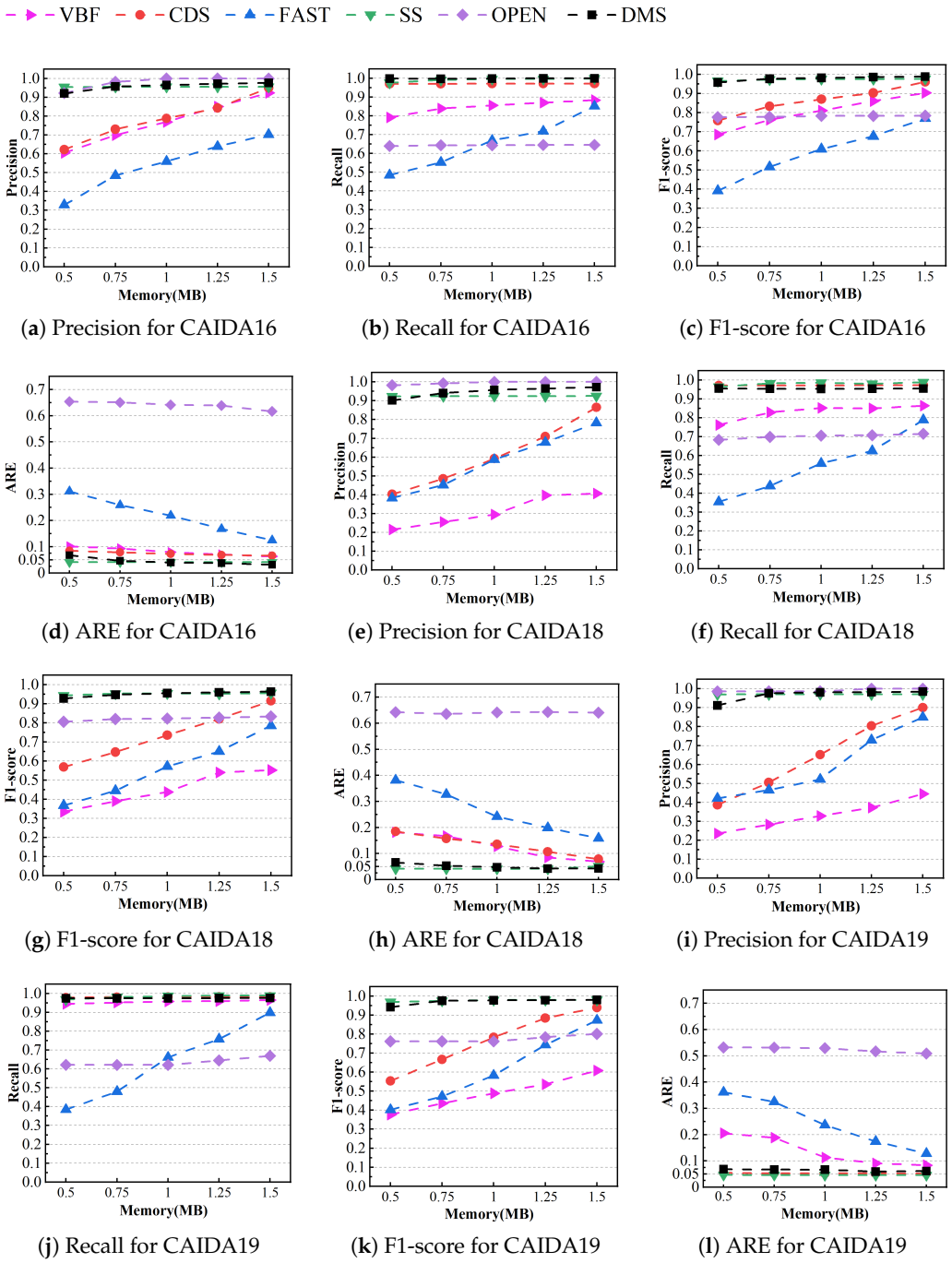
VBF and CDS attain substantially lower precision on CAIDA18 and CAIDA19 than on CAIDA16. The larger number of distinct items in CAIDA18 and CAIDA19 produces more hash collisions under the same memory budget, thereby reducing precision. On CAIDA19, DMS precision is 25% higher than that of CDS.

Heavy hitter Detection: The highest precision on every dataset belongs to DMS in Figure 6. Figure 6a shows that, with 1 MB memory, this precision is close to 100% and higher than UM. Recall also approaches 100% for all datasets, confirming robustness across different stream distributions. In Figure 6d,h,l, ARE stays under 1%; the CAIDA19 value is 4.6% lower than UM.

This heavy hitter detection accuracy results from separating the two traffic classes dynamically and storing them independently, thereby reducing mutual interference. By contrast, the multi-stream sampling mechanism of UM overestimates flow sizes and lowers precision, as seen in Figure 6a,e,i,k shows that DMS exceeds the UM F1-score by 23%, 6%, and 3% when memory is 0.5 MB, 0.75 MB and 1 MB, respectively.

Update Throughput: Figure 7 shows the update throughput of DMS on CAIDA19 dataset. The update throughput of DMS has surpassed that of the vast majority of sketches. Particularly, the throughput of DMS is 4.92 and 11.71 times higher than that of FAST and UM, respectively. The reason is that the update operation of DMS only involves a few simple operations, such as shifting, addition and hash operations. VBF, MV Sketch and Spacesaving have higher update throughput. This is because when an item arrives, VBF divides the flow label into eight parts during the update and finds the corresponding bucket through bit extraction of the hash function, thus avoiding spending a lot of time on hashing. Spacesaving maintains a fixed-length array and only needs to traverse the array to find the

corresponding bucket and update the counter therein to complete the update operation. In contrast, the throughput of FAST and UM is relatively low, the reason is that FAST requires multiple hashing during updates, and for each hash result, multiple bits need to be checked, causing a large number of judgment branches. UM adopts a hierarchical recursive structure, which leads to multiple layers of hashing during updates, thereby reducing the throughput. DMS preserves throughput comparable to VBF, MV and Spacesaving while performing super-spreader and heavy hitter detection simultaneously.



(**a**) Precision for CAIDA16 (**b**) Recall for CAIDA16 (**c**) F1-score for CAIDA16

(**d**) ARE for CAIDA16 (**e**) Precision for CAIDA18 (**f**) Recall for CAIDA18

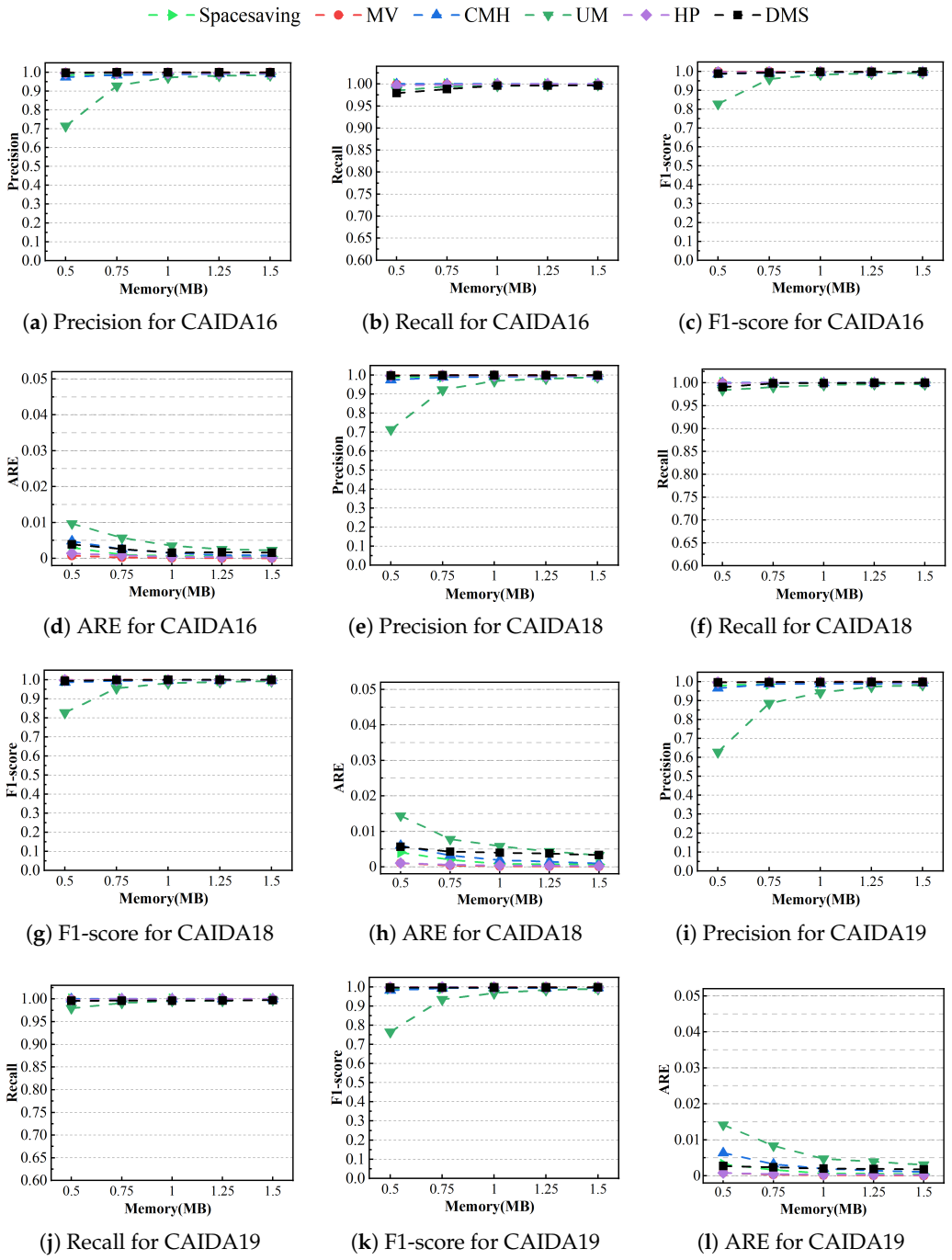
(**g**) F1-score for CAIDA18 (**h**) ARE for CAIDA18 (**i**) Precision for CAIDA19

(**j**) Recall for CAIDA19 (**k**) F1-score for CAIDA19 (**l**) ARE for CAIDA19

Figure 5. The accuracy of super-spreader detection on CAIDA datasets.

Performance Comparison of Joint Deployment: To more clearly contrast the performance differences between DMS and the deployment of multiple independent sketch schemes simultaneously, this section compares DMS with a joint detection scheme composed of two specialized sketches. This joint detection scheme is made up of the sketches

with the best performance in each of the two measurement tasks, namely Spread Sketch for super-spreader detection and Spacesaving for heavy flow detection. For ease of comparison in the following text, this joint detection scheme is named Combine Sketch in this section.



(**a**) Precision for CAIDA16 (**b**) Recall for CAIDA16 (**c**) F1-score for CAIDA16

(**d**) ARE for CAIDA16 (**e**) Precision for CAIDA18 (**f**) Recall for CAIDA18

(**g**) F1-score for CAIDA18 (**h**) ARE for CAIDA18 (**i**) Precision for CAIDA19

(**j**) Recall for CAIDA19 (**k**) F1-score for CAIDA19 (**l**) ARE for CAIDA19

Figure 6. The accuracy of heavy hitter detection on CAIDA datasets.

For a fair comparison, both DMS and Combine Sketch receive a total memory budget of 1 MB. Combine Sketch assigns 0.5 MB to Spread Sketch and 0.5 MB to Spacesaving. This equal split is used because the two tasks are equally important and no prior priority is assumed. It provides a fair baseline without deliberately weakening either component. Other memory splits may improve one task at the expense of the other.

Combine Sketch is composed of two independent sketches, Spread Sketch for super-spreader detection and Spacesaving for heavy-hitter detection. In the throughput evaluation, each sketch is allocated with equal and separate processor and memory resources. Under this execution architecture, the two sketches run in parallel, processing every incoming packet to perform both measurement tasks simultaneously. Under this parallel execution model, the combined throughput is determined by the slower of the two sketches.

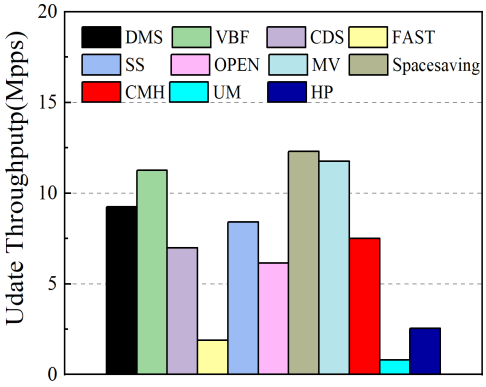


Figure 7. Update throughput.

The experimental results are shown in Figures 8–10. DMS outperforms Combine Sketch on most metrics: in super-spreader detection, DMS achieves an F1-score as high as 0.98, surpassing Combine Sketch's 0.95; meanwhile, DMS achieves a very low average relative error of only 0.066. The high F1-score is consistent with the spreader layer's bitmap representation, which records connection diversity without increasing bitmap occupancy for repeated items. Taking the minimum estimate across the mapped rows also helps limit overestimation caused by collisions with other flow labels. In heavy flow detection, DMS achieves an F1 score as high as 0.997, outperforming Combine Sketch's 0.99; at the same time, its average relative error is also controlled at an extremely low 0.0019, far lower than the 0.003 of Combine Sketch. DMS updates retained items directly in the hitter layer and uses a counter-dependent replacement threshold to favor the retention of frequent items. Its auxiliary counters also retain frequency information when candidates are displaced, providing a mechanism consistent with the high F1-score and low counting error observed. Additionally, the update throughput of Combine Sketch is determined by the sketch with the lowest throughput among them. As shown in Figure 10, DMS not only demonstrates superior performance in super-spreader detection and heavy flow detection but also achieves a higher update throughput, approximately 12% higher than that of Combine Sketch, showcasing a clear advantage.

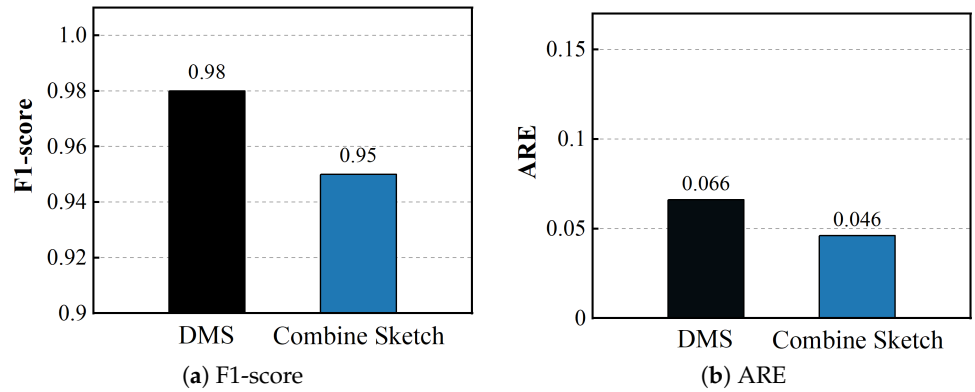
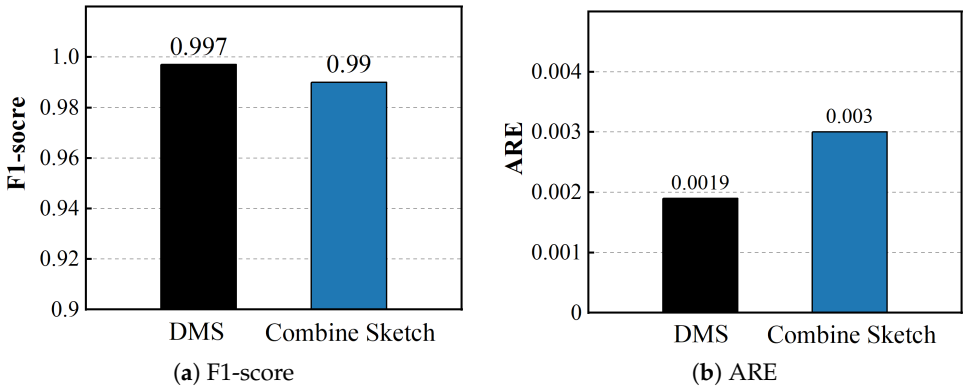


Figure 8. Comparison of super-spreader detection.



(**a**) F1-score (**b**) ARE

Figure 9. Comparison of heavy hitter detection.

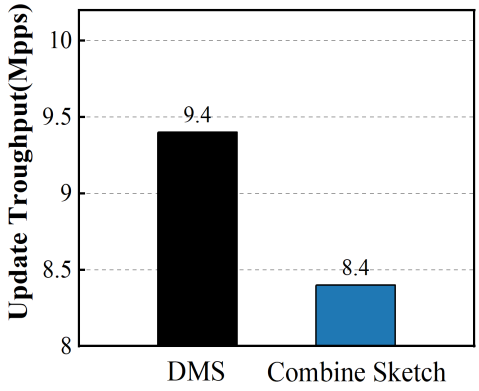


Figure 10. Comparison of throughput.

5.3. Discussion

The three datasets differ in traffic volume and distribution. By distinguishing super-spreaders from heavy hitters and storing them separately during updates, DMS detects both classes robustly under variable and skewed traffic. On CAIDA19 with 1 MB memory, DMS improves super-spreader F1-score by up to 40% and reduces heavy-hitter error by up to 80% compared with dedicated state-of-the-art sketches. These results demonstrate accurate joint detection without sacrificing throughput. In the microgrid network, the hash conflicts, highly skewed traffic and the memory cost of separated sketches are the challenges of sketch-based schemes. DMS addresses them jointly. It separates super-spreaders and heavy hitters dynamically through ignoring duplicate flows for cardinality estimation and keeping them for flow size estimation. This reduces influence from low-cardinality sources and mouse flows, making higher F1-score and lower errors. DMS also achieves super-spreader and heavy hitter detection in one insertion, avoiding the memory and time consumption of two dedicated sketches.

6. OVS Implementation

This section measures DMS packet-processing performance on the OVS platform. Experiments integrate OVS 2.12.1 and DPDK 18.11.10 in the platform configuration. The OVS-DPDK is configured to process 64-byte packets and is allocated of one RxQ and one TxQ for each port, running on a 12-core Intel i9-10920X platform. Fast user-space packet handling is supplied by the dpif-netdev module; consequently, the CPU can combine data packet forwarding with flow size measurement. In microgrid communication, DMS is evaluated as a traffic measurement component in the data-plane, whose stored traffic statistics can further support control-plane functions. This experiment focuses on the evaluation of the processing rate in the data-plane.

Figure 11 reports packet processing rate. DMS processes packets quickly and passes them to the control plane for congestion control, scheduling, and load balancing. Latency, jitter, and loss are mainly control-plane concerns, not sketch concerns. DMS reaches 15.71 Mpps, comparable to SS and higher than MV and UM. VBF and Spacesaving are faster but do not support both tasks. This shows DMS can provide joint measurement with practical data-plane throughput.

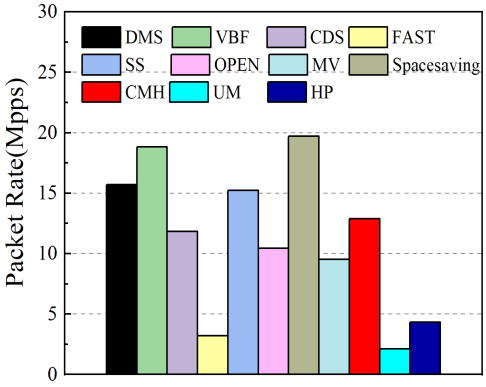


Figure 11. Packet rate on OVS platform.

7. Conclusions

DualMeasure Sketch provides a single sketch structure for concurrent heavy-hitter and super-spreader measurement. Its update process dynamically separates the two traffic classes and stores them independently, reducing interference under skewed traffic. On CAIDA with 0.5–1.5 MB memory, compared with state-of-the-art sketches, DMS improves about 10% in heavy hitter detection F1-score, 22% in super-spreader detection F1-score, and more than 50% in update throughput. In a joint deployment on CAIDA19 with 1 MB total memory, compared with Combine Sketch, DMS improves about 12% in throughput. These results address the challenges raised in the Introduction: conflicting measurement requirements, skewed traffic, and the overhead of separate sketches. These results support DMS as an effective sketch-based solution for detecting heavy hitters and super-spreaders simultaneously in microgrid network traffic.

Author Contributions: Conceptualization, R.W., W.L., S.X., and Z.L.; methodology, R.W., W.L., and R.M.; software, R.W., W.L., and R.M.; validation, R.W. and W.L.; formal analysis, R.W., W.L., and S.X.; investigation, R.W., W.L., X.H., S.X., R.M., H.X., and Z.L.; resources, X.H. and R.M.; data curation, R.W., W.L., S.X., and R.M.; writing—original draft preparation, R.W., W.L., X.H., S.X., R.M., H.X., and Z.L.; writing—review and editing, R.W., W.L., X.H., S.X., R.M., H.X., and Z.L.; visualization, R.W., W.L., and H.X.; supervision, X.H. and Z.L.; project administration, X.H. and Z.L.; funding acquisition, X.H. All authors agree to be accountable for all aspects of the work and to ensure that questions about its accuracy or integrity are investigated and resolved. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Science and Technology Project of State Grid Corporation of China under Contract kj2025-090.

Data Availability Statement: The datasets presented in this article are not readily available because the data are part of an ongoing study. Requests to access the datasets should be directed to the corresponding author (Zhuo Li). The raw data supporting the conclusions of this article will be made available by the corresponding author (Zhuo Li) on request.

Conflicts of Interest: The funding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results. Authors Ruofei Wang, Xuekai Hu, Shiwei Xue, and Rui Ma are employed by the company

State Grid Hebei Electric Power Research Institute. The research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The authors declare no conflicts of interest.

References

1. Hirsch, A.; Parag, Y.; Guerrero, J. Microgrids: A review of technologies, key drivers, and outstanding issues. *Renew. Sustain. Energy Rev.* **2018**, *90*, 402–411. [CrossRef]
2. Nejabatkhah, F.; Li, Y.W.; Liang, H.; Ahrabi, R.R. Cyber-Security of Smart Microgrids: A Survey. *Energies* **2021**, *14*, 27. [CrossRef]
3. Gu, S. Identifying network connection: Benign vs. DoS attack-A comparative analysis of binary classifiers. *Appl. Comput. Eng.* **2024**, *34*, 147–152. [CrossRef]
4. Adedeji, K.B.; Abu-Mahfouz, A.M.; Kurien, A.M. DDoS attack and detection methods in internet-enabled networks: Concept, research perspectives, and challenges. *J. Sens. Actuator Netw.* **2023**, *12*, 51. [CrossRef]
5. Zhao, Z.; Li, Z.; Zhou, Z.; Yu, J.; Song, Z.; Xie, X.; Zhang, F.; Zhang, R. DDoS family: A novel perspective for massive types of DDoS attacks. *Comput. Secur.* **2024**, *138*, 103663. [CrossRef]
6. Kandula, S.; Sengupta, S.; Greenberg, A.; Patel, P.; Chaiken, R. The nature of data center traffic: Measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, IMC '09*; ACM: New York, NY, USA, 2009; pp. 202–208.
7. Li, W.; Patras, P. Stable-Sketch: A Versatile Sketch for Accurate, Fast, Web-Scale Data Stream Processing. In *Proceedings of the ACM Web Conference 2024, WWW '24*; ACM: New York, NY, USA, 2024; pp. 4227–4238.
8. Qi, Y.; Li, R.; Wang, P.; Sun, Y.; Xing, R. QSketch: An Efficient Sketch for Weighted Cardinality Estimation in Streams. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '24*; ACM: New York, NY, USA, 2024; pp. 2432–2443.
9. Tang, L.; Huang, Q.; Lee, P.P.C. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In *Proceedings of the IEEE INFOCOM 2019–IEEE Conference on Computer Communications*; IEEE: Piscataway, NJ, USA, 2019; pp. 2026–2034.
10. Yang, T.; Jiang, J.; Liu, P.; Huang, Q.; Gong, J.; Zhou, Y.; Miao, R.; Li, X.; Uhlig, S. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18*; ACM: New York, NY, USA, 2018; pp. 561–575.
11. Metwally, A.; Agrawal, D.; El Abbadi, A. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the International Conference on Database Theory*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 398–412.
12. Tang, L.; Xiao, Y.; Huang, Q.; Lee, P.P.C. A High-Performance Invertible Sketch for Network-Wide Superspreader Detection. *IEEE/ACM Trans. Netw.* **2023**, *31*, 724–737. [CrossRef]
13. Liu, W.; Qu, W.; Gong, J.; Li, K. Detection of Superpoints Using a Vector Bloom Filter. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 514–527. [CrossRef]
14. Liu, Y.; Chen, W.; Guan, Y. A fast sketch for aggregate queries over high-speed network traffic. In *Proceedings of the 2012 Proceedings IEEE INFOCOM*; IEEE: Piscataway, NJ, USA, 2012; pp. 2741–2745.
15. Peng, T.; Leckie, C.; Ramamohanarao, K. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Comput. Surv.* **2007**, *39*, 3–es. [CrossRef]
16. Jing, X.; Yan, Z.; Han, H.; Pedrycz, W. ExtendedSketch: Fusing Network Traffic for Super Host Identification With a Memory Efficient Sketch. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 3913–3924. [CrossRef]
17. Yu, M.; Jose, L.; Miao, R. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*; USENIX Association: Berkeley, CA, USA, 2013; pp. 29–42.
18. Wang, P.; Guan, X.; Qin, T.; Huang, Q. A Data Streaming Method for Monitoring Host Connection Degrees of High-Speed Links. *IEEE Trans. Inf. Forensics Secur.* **2011**, *6*, 1086–1098. [CrossRef]
19. Yoon, M.; Chen, S. Detecting stealthy spreaders by random aging streaming filters. *IEICE Trans. Commun.* **2011**, *94*, 2274–2281. [CrossRef]
20. Ma, C.; Odegbile, O.O.; Melissourgos, D.; Wang, H.; Chen, S. From countmin to super kjoin sketches for flow spread estimation. *IEEE Trans. Netw. Sci. Eng.* **2023**, *11*, 2353–2370. [CrossRef]
21. Zhou, Y.; Zhang, Y.; Ma, C.; Chen, S.; Odegbile, O.O. Generalized sketch families for network traffic measurement. *Proc. ACM Meas. Anal. Comput. Syst.* **2019**, *3*, 51. [CrossRef]
22. Cormode, G.; Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms* **2005**, *55*, 58–75. [CrossRef]
23. Liu, Z.; Manousis, A.; Vorsanger, G.; Sekar, V.; Braverman, V. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16*; ACM: New York, NY, USA, 2016; pp. 101–114.

24. Sivaraman, V.; Narayana, S.; Rottenstreich, O.; Muthukrishnan, S.; Rexford, J. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research, SOSR '17*; ACM: New York, NY, USA, 2017; pp. 164–176.
25. Li, Y.; Yu, X.; Yang, Y.; Zhou, Y.; Yang, T.; Ma, Z.; Chen, S. Pyramid family: Generic frameworks for accurate and fast flow size measurement. *IEEE/ACM Trans. Netw.* **2021**, *30*, 586–600. [CrossRef]
26. Zhou, Y.; Jin, H.; Liu, P.; Zhang, H.; Yang, T.; Li, X. Accurate per-flow measurement with bloom sketch. In *Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*; IEEE: Piscataway, NJ, USA, 2018; pp. 1–2.
27. Yang, T.; Gao, S.; Sun, Z.; Wang, Y.; Shen, Y.; Li, X. Diamond sketch: Accurate per-flow measurement for big streaming data. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2650–2662. [CrossRef]
28. Fan, Z.; Wang, R.; Cai, Y.; Zhang, R.; Yang, T.; Wu, Y.; Cui, B.; Uhlig, S. Onesketch: A generic and accurate sketch for data streams. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 12887–12901. [CrossRef]
29. Liu, J.; Li, Z.; Du, H.; Zhou, H.; Li, L.; An, Y.; Zhang, Y.; Liu, K.; Li, Q. An effective and accurate flow size measurement using funnel-shaped sketch. *Comput. Netw.* **2024**, *247*, 110467. [CrossRef]
30. Landau-Feibish, S.; Liu, Z.; Rexford, J. Compact Data Structures for Network Telemetry. *ACM Comput. Surv.* **2025**, *57*, 191. [CrossRef]
31. CAIDA. The CAIDA UCSD Anonymized Internet Traces—2016, 2018, and 2019. Available online: https://www.caida.org/catalog/datasets/passive_dataset/ (accessed on 14 October 2026).
32. Shi, Q.; Jia, C.; Li, W.; Liu, Z.; Yang, T.; Ji, J.; Xie, G.; Zhang, W.; Yu, M. BitMatcher: Bit-level Counter Adjustment for Sketches. In *Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE)*; IEEE: Piscataway, NJ, USA, 2024; pp. 4815–4827.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.