IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 23 March 2024, accepted 3 April 2024, date of publication 5 April 2024, date of current version 12 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3385499

RESEARCH ARTICLE

Detecting Top-k Flows Combining Probabilistic Sketch and Sliding Window

KEKE ZHENG[1], WENZHU CHEN[1], BOTAO FENG[2], (Senior Member, IEEE), AND HANXIN ZHANG[3]

[1]College of Information Science and Technology, Jinan University, Guangzhou 510632, China
[2]College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China
[3]School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China

Corresponding author: Keke Zheng (kekezheng_coco@163.com)

ABSTRACT Efficient real-time top-k flows measurement plays a pivotal role in enhancing both network performance and security, including tasks such as timely traffic scheduling, optimizing network latency and identifying potential security threats. However, traditional methods for detecting top-k flows suffer from decreased accuracy and high memory overhead. Furthermore, many existing methods overlook finer-grained measurements, such as the detection within the latest short time intervals. With the increasing expansion scale and link speed of the network, an accurate real-time top-k flows identified method is required. This paper proposes *wSketch*, a novel sketch-based method for real-time top-k flows detection. The innovations of *wSketch* are that it combines with the sliding window model and circular queue model, and introduces a novel probabilistic update solution. The probabilistic update mechanism gives the larger flow a greater chance of retention, the sliding window model focuses on the latest flow in the last $\mathbb{W}$ time units, and the circular queue reduces memory consumption. Therefore, *wSketch* provides insights into the current network situation and does well in anticipating future trends. The experimental results showcase *wSketch*'s superior performance, achieving over 96% accuracy with a small memory size of 20KB.

INDEX TERMS Network measurement, sketch, sliding window, top-k flows detection, approximate measurement, network monitoring.

I. INTRODUCTION

Identifying the top-k flows (i.e., the largest k flows) within network traffic stands as a crucial endeavor in enhancing network performance and security management. Detecting the top-k flows precisely holds paramount importance for various network functionalities, including traffic engineering [1], [2], [3], flow scheduling [4], [5], load balance [6], [7], [8], frequency estimation [9], [10], and abnormal traffic behaviors detection [11], [12] (e.g., heavy hitters [13], [14], [15] and heavy changers [16]). Therefore, detecting top-k flows is highly beneficial for many network applications [17], [18].

With the rapid expansion of network scale and of traffic volume, it is challenge to store all flows within network devices (e.g., switches) due to their limited memory resources [19]. Therefore, it is impractical to assign a counter to each flow because of the large memory consumption. Besides, the increasing network link speeds also make it challenging to accurately detect top-k flows under high-speed network conditions.

In the realm of real network traffic, it's widely known that the flow sizes distribution is highly skewed [20], [21], [22]. In other words, the majority of flows are small in size, which we refer to as "*mice flows*," while a minor portion of flows are large flows, known as "*elephant flows*." Despite the *elephant flows* are fewer in number, it account for a substantial portion of the total traffic volume. whereas the number of *mice flows* is more enormous than *elephant flows*. In the context of top-k flow detection, more attention should be paid to *elephant flows* to improve accuracy, while reducing the storage of *mice flows* to reduce memory consumption. Consequently, approximate detection methods have been proposed in the previous literature and gained wide acceptance [13], [23], [24].

Previous research on approximated top-k flows can be divided into three categories: (1) Methods based on counter: These techniques, such as Space Saving [24], maintain a

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Khalil Afzal.

summary of flows within a table and update it by replacing the smallest flow. However, the accuracy is compromised due to the skewed shape of network traffic distribution [22]. (2) Methods based on sample: This methods [25], [26] record packets at a predefined sampling rate to reduce memory consumption. Nevertheless, achieving high accuracy becomes challenging due to the dilemma of selecting an appropriate sampling rate. A high sampling rate can ensure high precision but entails increased memory overhead. Conversely, a low sampling rate is memory-friendly but will also decrease accuracy. (3) Methods based on sketch: These methods [13], [27], [28], [29] achieve approximated top-k flows measurement with high accuracy by counting flows within a compact data structure (i.e., sketch) shared among different flows. For example, Count Min Sketch [13] utilizes a two-dimensional array with d rows and s columns. It maps each flow to d buckets according to the hash functions of each row and returns the minimum value among d mapped buckets as the measurement result.

The simple structure and low memory overhead of sketch make it efficiently handle network flows and have been widely researched recently. By combining with other structures, sketch show great promise in top-k flows detection [30]. However, the former methods usually detect top-k flows within a long period of time. But the rapid development of the Internet prompts the applications to focus more on the latest flow statistics [19], so the significance of traffic information decreases over time. Consequently, the detection and analysis of top-k flows should focus on the most recent time to reflect the current network situation and future trends. To achieve this purpose, the sliding window model [31] can be applied to the design of the sketch, which is also called sliding window sketch [32].

In this paper, we propose *wSketch*, a novel probabilistic sliding window sketch method to detect real-time top-k flows. Compared with traditional sketch-based methods, *wSketch* combines the sliding window model, circular queue model and probabilistic calculation. Specifically, *wSketch* focus on the top-k flows stored in the current window while ignoring the flow information that has already been moved out of the sliding window to capture the most recent network situation. An important feature of circular queue is the reuse of memory space, which can be used to reduce memory consumption. In addition, *wSketch* updates the sketch in a probabilistic manner and gives the large flows a greater chance of preservation. When the hash collision occurs, *wSketch* calculates the probability based on the recorded flow size and the frequency of hash collision to determine whether to replace the flow in the sketch. By using the dynamic probability mechanism, there is a high chance of retention for top-k flows, and vice versa, and the accuracy of flow size estimation is also be improved. Therefore, *wSketch* can detect the real-time top-k flows with high accuracy and low memory consumption.

In short, this paper makes the following contributions:

(1) A novel sliding window sketch model is proposed to identify the real-time top-k flows. *wSketch* combines the sliding window model and circular queue to focus on the latest top-k flows detection with less memory consumption.
(2) *wSketch* proposes a new dynamic probability calculation mechanism and update strategy, which greatly improves the accuracy of top-k flows detection and flow size estimation with less memory consumption.
(3) A mathematical analysis is developed for *wSketch* to theoretically prove its high precision. In addition, extensive experiments are conducted to demonstrate the high accuracy of *wSketch*. The experimental results show that *wSketch* can achieve an accuracy of more than 96% with only 20KB of memory.

The rest of this paper is organized as follows. Section II outlines the background and related works. Section III introduces the design of *wSketch* in detail. The experiment setup and results are provided in Section IV. Finally, we make a conclusion and discuss the future work in Section V.

II. RELATED WORKS

We compare and summarize some of the existing works on measuring top-k flows as illustrated in Table 1. These works can be classified into three types: methods based on counter, methods based on sample, and methods based on sketch.

A. METHODS BASED ON COUNTER

The core idea underlying counter-based methods [23], [24], [33], [34], [35] is centered around maintaining counters to monitor the most frequent flows within the network. For example, Space Saving (SS) [24] utilizes a min-heap structure to manage each incoming flow, and each node stores a $< key, value >$ pair. When the min-heap is full and a new flow arrives, the update strategy of SS is to replace the smallest flows and increase the smallest counter by 1. However, as k increases, the precision of detection diminishes unless additional memory resources are allocated to accommodate the expanding set of top-k flows. WCSS [23] extends Space Saving to detect heavy hitters over a sliding window that counts the most recent W items.

Lossy Counting [33], employs a flow table supplemented with associated counters to keep the top-k flows. When new flow arrives, Lossy Counting either increments the counter associated with the flow or replaces the smallest flow to maintain the set of top-k flows.

CELL [34] utilizes estimators, which uses fewer bits than the counter, to estimate the frequency of the flow. CELL maps a flow to a specific level according to its frequency. The level is saved in an estimator. The increase in flow size will also increase the level of the flow.

The counter-based methods exhibit relatively good performance under the conditions of sufficient memory or limited network traffic volume. However, the scale of network traffic

TABLE 1. Comparison with existing solutions.

Method	Type	Accuracy	Structure	Core idea
Space Saving [24]	Counter based	Medium	Min-heap, Counter	Maintaining the counter in the form of min-heap, replacing the smallest flow, and adding the smallest counter by 1.
Lossy Counting [33]	Counter based	Medium	Counter	Employing a flow table with associated counters to keep track of large flows.
WCSS [23]	Counter based	Medium	Array, Tiny table	Extending Space Saving in a sliding window.
CELL [34]	Counter based	Medium	Hash table	Mapping a flow to a specific level based on its frequency. The level will increase if the flow size increases.
ActiveGuardian [35]	Counter based	High	Counter Array, Bucket Array	Separating potential elephant flows from mice flows, guarding and reporting active elephant flows in every time window.
NetFlow [26]	Sample based	Medium	\	Sampling the network information and sending it to the collector for analysis and processing.
sFlow [25]	Sample based	Medium	\	Sampling and analyzing packets at a predefined rate and sending to the Collector.
Count Min [13]	Sketch based	Low	Sketch	Increasing the mapped counter in sketch by 1 and returning the minimal counter's value.
Heavy Keeper [22]	Sketch based	High	Sketch, Min-heap	Adopting an exponential decay strategy and min-heap to remove small flows.
HashPipe [41]	Sketch based	Low	Sketch	Obtaining large flows by continuously rolling eviction to small flows.
LUSketch [27]	Sketch based	High	Sketch, Min-heap	Establishing a connection between sketch and min-heap to improve the speed of top-k flows detection.
CountMax [37]	Sketch based	Low	Sketch	Utilizing the majority vote algorithm for identifying elephant flows.
wSketch	Sketch based	High	Sketch, Array	Updating through probabilistic calculation

* The \ indicates that the information is not explicitly mentioned.

is growing exponentially, which poses significant challenges to memory overhead and detection accuracy.

B. METHODS BASED ON SAMPLE

Sampling presents a solution by collecting packets at a predefined rate and transmitting flow statistics to collectors for further analysis. The accuracy of sample-based methods heavily depends on the chosen sampling rate and the characteristics of network traffic. It's worth noting that a higher sampling rate can improve accuracy but results in significant bandwidth overhead and transmission delays. Conversely, lower sampling rates alleviate resources but may compromise the accuracy of detection. Therefore, determining the optimal sampling rate is a significant challenge task, requiring a careful trade-off between accuracy and resource utilization(i.e., SRAM).

The escalating volume of network traffic, combined with limited memory resources, presents a challenge for sample-based methodologies. As network traffic continues to grow exponentially, finding efficient and accurate sampling strategies and striking the right balance is crucial.

C. METHODS BASED ON SKETCH

Sketch-based methodologies, such as Count Min [13], LUSketch [27], sketch-BF [28], Cuckoo Counter [36], CountMax [37], FCM-Sketch [38], ActiveGuardian [35] and

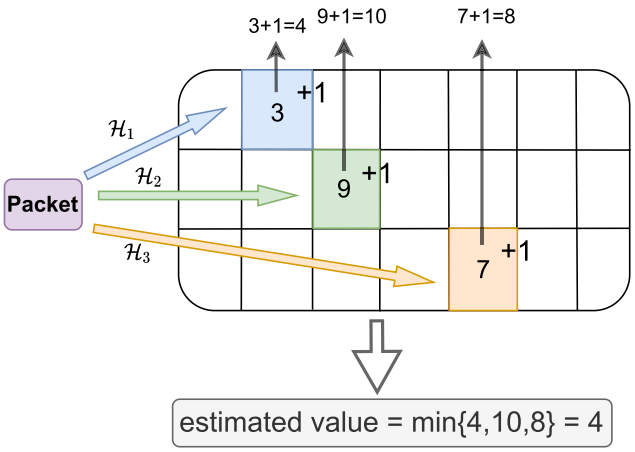


FIGURE 1. Example of count min sketch.

others [39], [40], have been extensively researched. Count Min, as shown in FIGURE 1, maps each incoming flow into d buckets based on respective hash functions and returns the minimum value from these buckets. Despite its simplicity and efficiency, Count Min is susceptible to overestimation issues, because of its continuous increase in the value of the mapped counter without distinguishing different flows. This problem becomes more serious under conditions of limited memory and growing traffic volumes, where hash collisions occur more frequently.

CountMax [37], an improvement over Count Min, mitigates the issue of overestimation by maintaining $< key, value >$ pairs. In cases of hash collisions, it calculates the difference between the current and recorded values, and retains the larger flow with the difference. Ultimately, CountMax derives its estimate by selecting the maximum value across d rows.

Heavy Keeper [22] adopts a min-heap structure to monitor the top-k flows and achieves relatively high accuracy. However, a larger k may strain memory usage, as the min-heap necessitates additional nodes to accommodate the top-k flows.

ActiveGuardian [35] separate potential elephant flows from mice flows. It consists of a filtering module evicting mice flows as well as low-arrival-rate flows and a guarding module identifying active elephant flows.

HashPipe [41] introduces a rolling eviction mechanism to potentially replace small flows. Nonetheless, it faces challenges in scenarios with skewed network traffic [20], where subsequent arrivals of small flows can compromise the accuracy of preserved top-k flows due to the mandatory replacement strategy of the first row.

Although the above methods have achieved relatively good performance, these methods are all aimed at top-k flows detection within a long period, and less consideration is given to the factor that as time migrates, the impact of previous flow information gradually decreases.

In this paper, we propose a novel sliding window sketch method that updates in a probabilistic manner, namely *wSketch*. Specifically, *wSketch* uses the sliding window sketch model to achieve the goal of measuring real-time top-k flows in the most recent period. And the use of the circular queue helps *wSketch* reduce the memory overhead caused by sliding windows through reusing memory. Besides, *wSketch* introduces a probabilistic calculation manner in the sketch, where large flows can be better preserved according to the skewed shape of network traffic. Therefore, *wSketch* can achieve high-precision real-time top-k flows detection and flow size estimation with a small memory overhead, making *wSketch* well suited for different network applications [18], [42].

TABLE 2. Symbols and notations.

Symbol	Description
$\mathbb{W}$	Window unit number
$\mathbb{N}$	Total number of packets
n	Total number of flows
d	*Elephant part* consists of d rows
s	*Elephant part* consists of s buckets in each row
$\mathbb{R}_i$	The i^{th} row of *Elephant part*
$\mathbb{H}_i$	The hash function used in $\mathbb{R}_i$ in *Elephant part*
res	The probability calculation result
random	The generated random number

III. STRUCTURE DESIGN OF WSKETCH

This section describes the structure and processing logic of *wSketch*, and introduces how *wSketch* accurately detects real-time top-k flows with low memory cost through the sliding window model and probabilistic update mechanism. The symbols used frequently in the paper are shown in Table 2.

A. STRUCTURE OVERVIEW

Due to the skewed shape of traffic flow in real networks as discussed above [22] and the inevitable hash collisions in sketch, *wSketch* adopts a probabilistic update strategy in sketch to enhance the accuracy of top-k flows detection. As depicted in FIGURE 2, the structure of *wSketch* comprises two main components: the *Elephant part* (i.e., a sketch) and the *Mice part* (i.e., an array).

The *Elephant part* consists of d rows $(r_1, r_2, \cdots, r_d)$ and s columns, with each row associated with a distinct hash function $(\mathbb{H}_1, \mathbb{H}_2, \cdots, \mathbb{H}_d)$. Different from the former solutions, *wSketch* utilizes a new parameter (i.e., Counter). So each bucket in the *Elephant part* stores three parameters

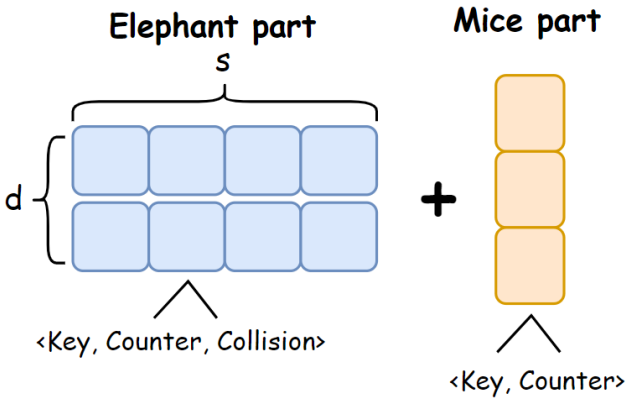


FIGURE 2. Structure design of *wSketch*.

<Key, Counter, and Counter>, where Counter is used to differentiate different flows, Counter is used to record flow size, while Counter is used to tally hash collisions. *wSketch* updates the bucket in *Elephant part* with dynamically probabilities adjusted based on Counter and Counter values. This ensures that larger flows have a higher chance of retention, optimizing the preservation of top-k flows while disregarding smaller ones.

The *Mice part* is an array storing <Counter, Counter> tuples. Initially, large flows may possess small Counter and risk being replaced in the *Elephant part*. In such cases, the *Mice part* serves to prevent inadvertent loss of large flows and further enhancing detection accuracy.

As shown in FIGURE 3(a), traditional sketch-based methods primarily focus on counting all the network flows at each bucket in a long period (from time beginning to time ending). When querying, it summarizes all the flows that have passed and returns the measurement results. Besides, many methods face problem of high memory and low accuracy. To address this issue and better detect the latest traffic flows, *wSketch* combined with a sliding window model for real-time top-k flows detection.

Each bucket of *wSketch* (including *Elephant part* and *Mice part*) is divided into several windows based on the detection time (suppose it is divided into N windows as shown in FIGURE 3 (b)). The measure time of each window is the same, so as to realize time-sharing statistics. *wSketch* only focuses on the flows in the latest $\mathbb{W}$ ($\mathbb{W} \leq N$) windows, as it reflects the latest network conditions.

However, recording all the passing flows is a memory challenge duo to the large amount of traffic transmitted in the network. Since the outdated flows are not important for current statistics and predictions, so we can consider directly ''discarding'' these outdated flows to optimize memory overhead. Therefore, *wSketch* adopts a ***Circular Queue*** model (as shown in FIGURE 4) to optimize memory overhead. Assuming the task is to detect top-k flows from up to the past $\mathbb{W}$ windows, the length of the *Circular Queue* will be set to $\mathbb{W}$. This circular approach ensures that the latest top-k flows can be detected while effectively reducing memory overhead.

To further reduce the memory usage due to the sliding window, the *Elephant part* merges the parameter Counter at each window into one and shares it (as shown in FIGURE 5). Suppose that the Counter occupies 32 bits, the Counter occupies 16 bits, the Counter occupies 16 bits, and the number of windows is 5 (i.e. $\mathbb{W} = 5$). Then, the memory overhead of each bucket before the improvement is $(32 + 16 + 16) \times 5 = 320$ bits, after merging Counter, the memory overhead of each bucket is $(32 + 16) \times 5 + 16 = 256$ bits.

The internal processing logic and step of *wSketch* is illustrated in FIGURE 6. And in the following, we will provide a detailed introduction to the operations of each part

B. OPERATIONS OF WINDOW UPDATE

Regarding the issue of cross-window updates in *wSketch*, let's assume that the measurement task needs to count flows from up to the past 5 time windows (i.e. $\mathbb{W} = 5$), and the current detect window is v. For each window cycle's update operation, there are mainly the following two situations:

(1) When in the window v ($1 \leq v \leq \mathbb{W} - 1$). The flows arriving within this window time are directly inserted into window v until the end of this window time. When it comes to the next cycle, the flows will be inserted into the next window $v + 1$.
(2) When in the window $v (v \geq \mathbb{W} - 1)$. All windows (totally $\mathbb{W}$ windows) have recorded flow information. When the next cycle reaches, there is no empty window in the circular queue to store new flows' information. The strategy *wSketch* adopts is to refresh and cover the information in window $v\%\mathbb{W}$.

For example, if it is currently in period 3, the packet will be inserted into window 4 in the next period. If it is currently in period 5, then the packet will be inserted into window $(5 + 1)\%5 = 1$ in the next period. Besides, whenever the window updates, the shared Collision will refresh to 0.

C. OPERATIONS IN ELEPHANT PART

To differentiate between newly arrived flows and those already stored in the *Elephant part*, we label the new flow as $\mathcal{K}_{new}$ and the recorded flow as $\mathcal{K}_{old}$.

The operations within the *Elephant part* (i.e., the sketch operations) primarily involve insertion, updating, and eviction of flows. Specifically, when a new flow $\mathcal{K}_{new}$ arrives, *wSketch* maps $\mathcal{K}_{new}$ to d buckets based on distinct hash functions $\mathbb{H}_i$ at each row. (1) If the mapped bucket is empty, *wSketch* inserts $\mathcal{K}_{new}$ directly into this bucket. (2) If the mapped bucket is not empty and $\mathcal{K}_{new} = \mathcal{K}_{old}$, *wSketch* increases the Counter by 1. (3) If the mapped bucket is not empty and $\mathcal{K}_{new} \neq \mathcal{K}_{old}$, *wSketch* increases the Counter by 1 and calculates the probability to determine whether to replace $\mathcal{K}_{old}$ or not. If the randomly generated $random$ is less than res (i.e., $random < res$), *wSketch* evicts $\mathcal{K}_{old}$ to the *Mice part* and inserts $\mathcal{K}_{new}$ into this bucket. Otherwise, *wSketch* evicts $\mathcal{K}_{new}$ into the *Mice part* and retains $\mathcal{K}_{old}$ in

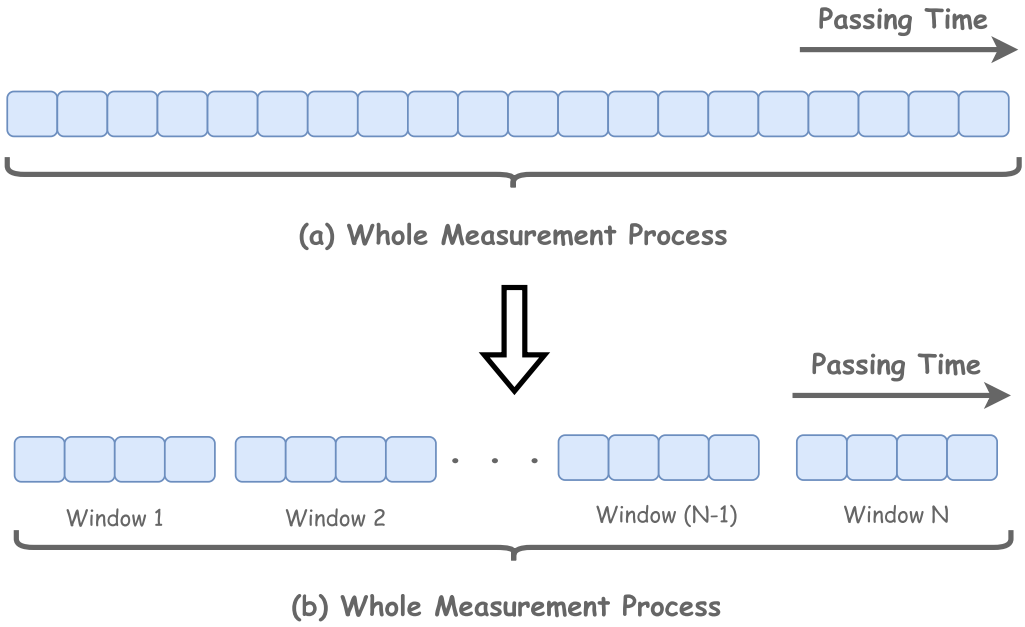


FIGURE 3. Processing Model of sliding window sketch.

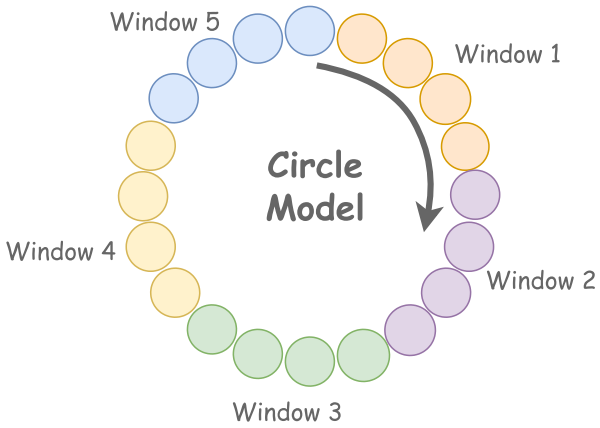


FIGURE 4. Circular queue model of *wSketch*.

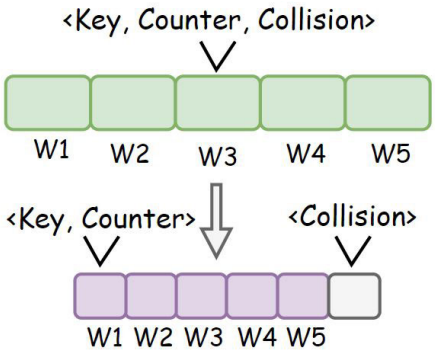


FIGURE 5. Merging counter among multiple windows.

the *Elephant part*. The probability formula is as follows:

$$p = base^{-Counter} * Collision. \quad (1)$$

By using probability, *wSketch* can effectively preserve large flows and improve measurement accuracy.

D. OPERATIONS IN MICE PART

The *Mice part* comprises a standard one-dimensional array primarily performing two operations: insertion and updating. Upon eviction from the *Elephant part*, *wSketch* inserts $\mathcal{K}_{new}$ into the *Mice part*. Three distinct operations in the *Mice part* should be considered. Firstly, if $\mathcal{K}_{new}$ is already stored, *wSketch* increases $\mathcal{V}_{old}$ by $\mathcal{V}_{new}$, updating the bucket to $<\mathcal{K}_{old}, \mathcal{V}_{new} + \mathcal{V}_{old}>$. Secondly, if $\mathcal{K}_{new}$ isn't present and an empty bucket exists, *wSketch* directly inserts $\mathcal{K}_{new}$, updating the empty bucket to $<\mathcal{K}_{new}, \mathcal{V}_{new}>$. Thirdly, if $\mathcal{K}_{new}$ isn't stored and no empty buckets exist, *wSketch* replaces the entry with the smallest Counter with $\mathcal{K}_{new}$. This updates the entry to $<\mathcal{K}_{new}, \mathcal{V}_{new}>$, while $\mathcal{K}_{old}$ is regarded as a small flow, ignored, and removed from the *Mice part*.

E. UPDATE AND QUERY OPERATION OF WSKETCH

In this section, we use a simple example to illustrate the update operations of *wSketch*'s *Elephant part*. For simplicity, we use a single window to illustrate. Assume that the *Elephant part* has 3 rows, and all rows have stored flow information. As shown in FIGURE 7, when flow $\mathcal{K}_1$ arrives, the *Elephant part* has the following update situations:

(1) When the flow stored in the mapped bucket is the same as the arrival flow (i.e., $\mathcal{K}_1 = \mathcal{K}_1$), as depicted in the first row r_1, *wSketch* increases Counter by 1 directly.

(2) When the arrival flow is different from the flow stored in the mapped bucket (i.e., $\mathcal{K}_1 \neq \mathcal{K}_3$), *wSketch* increases Counter by 1 and calculates the probability res to decide whether the stored flow $\mathcal{K}_3$ should be replaced by the arrival flow $\mathcal{K}_1$. Specifically, *wSketch* generates a random number $rand$, and there are two cases here:

(i) If $rand > res$, no replacement will occur. As depicted in the second row r_2, the flow in the

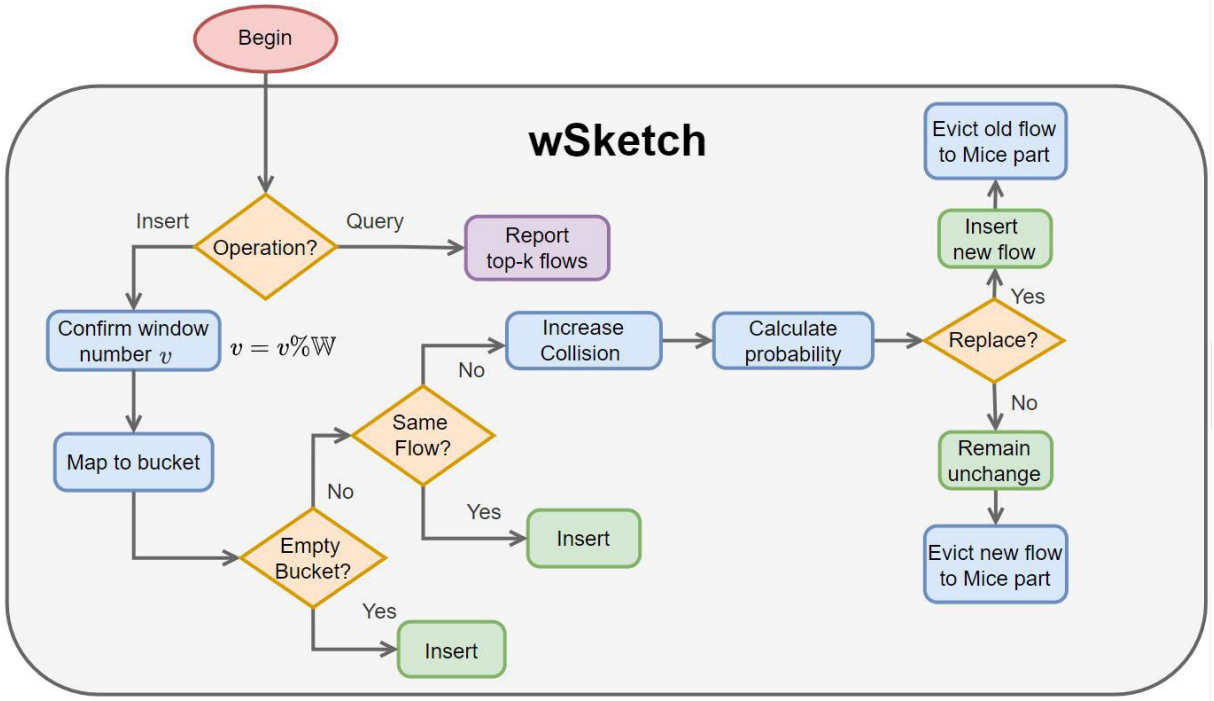


FIGURE 6. Internal processing logic and steps of *wSketch*.

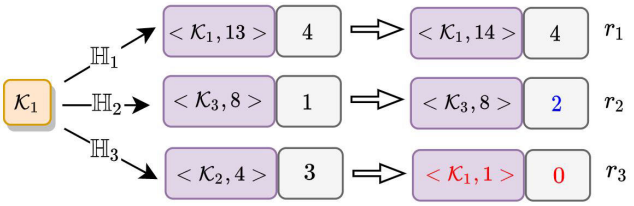


FIGURE 7. Update operations of *wSketch*.

mapped bucket remains unchanged and is still $\mathcal{K}_3$. At this time, *wSketch* increases Counter by 1, which is added from 1 to 2.

(ii) If $rand \leq res$, the arrival flow $\mathcal{K}_1$ will replace the stored flow $\mathcal{K}_2$. As depicted in the third row r_3, the flow in the mapped bucket will be modified from $< \mathcal{K}_2, 4 >$ to $< \mathcal{K}_1, 1 >$, where the Counter is refreshed to 1, and Counter is refreshed to 0.

For query operation, if the flow can be found in the *Elephant part*, *wSketch* returns the maximum value among d rows in the *Elephant part* (whether the flow is stored in the *Mice part*); Conversely, if the flow can only be found in the *Mice part*, *wSketch* returns the value in the *Mice part* directly;

As shown in FIGURE 8, the query result of $\mathcal{K}_1$ is $max(14, 9) = 14$ (the flow information of $\mathcal{K}_1$ in the *Mice part* is ignored), while the query result of $\mathcal{K}_2$ is 10 ($\mathcal{K}_2$ only be found in the *Mice part*). In addition, flow $\mathcal{K}_5$, $\mathcal{K}_6$, and $\mathcal{K}_7$ only stores in the *Mice part*, so the value in the *Mice part* is the query result. Thus, the query result of $\mathcal{K}_7$ is 11, $\mathcal{K}_5$ is 2, and $\mathcal{K}_6$ is 1.

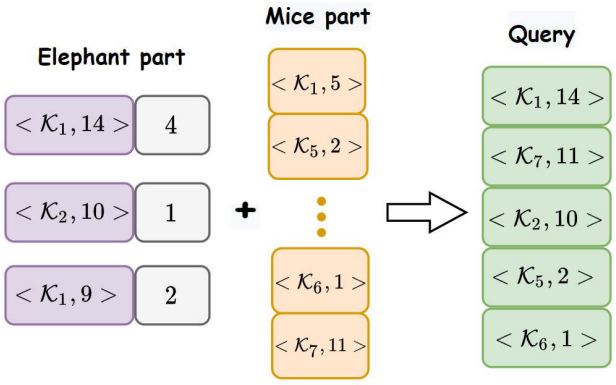


FIGURE 8. Query operation of *wSketch*.

F. MATHEMATICAL ANALYSIS

1) TIME COMPLEXITY OF WSKETCH

The structure of *wSketch* consists of two parts, i.e., the *Elephant part* and the *Mice part*. The sliding window and circular update does not affect the calculation of time complexity. Therefore, the time complexity of *wSketch* is considered from these two parts.

a: INSERTION

For the *Elephant part* (i.e. sketch), the operation is to insert the flows into d rows in the sketch, so the time complexity is $O(d)$. For the *Mice part* (i.e. one-dimensional array), the operation is to traverse the array, so the time complexity is $O(n)$.

Therefore, the time complexity of *wSketch* inserting flow is $O(d) + O(n)$, where d is the number of rows.

b: QUERY

For the *Elephant part*, the query will search from d rows in the sketch, so the time complexity is $O(d)$. For the *Mice part*, the query will traverse the array, so the time complexity is $O(n)$.

Therefore, the time complexity of *wSketch* querying is also $O(d) + O(n)$.

2) ERROR BOUND OF WSKETCH

For clearer expression, we mark $\mathcal{K}_e$ to record the query flow in *wSketch* (assuming that each flow can be queried in *wSketch*). f_e is the real size of the query flow, while $\hat{f}_e$ is the estimated flow size.

The flows in *Mice part* are all evicted from *Elephant part*, thus the *Mice part* will not cause overestimation. Specifically, the chance of estimation error of *wSketch* mainly has the following situations: If one of the mapped buckets of $\mathcal{K}_e$ in *Elephant part* is not replaced, the estimated result is correct (i.e., $\hat{f}_e = f_e$). Otherwise, if the replacement of $\mathcal{K}_e$ occurs at all rows, and $\mathcal{K}_e$ is recorded or not recorded in *Mice part*, then $\hat{f}_e \leq f_e$.

Given a small number ζ, and an top-k flow $\mathcal{K}_e$ is held in Elephant part, wSketch has the error bound as follows:

$$Pr\left\{\hat{f}_e \geq f_e - \zeta\mathbb{N}\right\} \geq 1 - \prod_{r=1}^{d}\left(\frac{1}{s\cdot\zeta}\right), \tag{2}$$

where $\mathbb{N}$ is the total number of packets, f_e is the real size of $\mathcal{K}_e$, and $\hat{f}_e$ is the estimated size of $\mathcal{K}_e$, respectively, ζ is an arbitrarily small positive number, d means the total number of rows in Elephant part, and s is the width of each row.

To simplify the analysis, we focus on a single row of *wSketch* and the flow $\mathcal{K}_e$ is correctly reported in $\mathbb{H}(\mathcal{K}_e)$. Let $T_{e,v}$ be a binary random variable that defined as:

$$T_{e,v} = \begin{cases} 1 & (e \neq v) \ \ and \ \ (\mathbb{H}(\mathcal{K}_e) = \mathbb{H}(\mathcal{K}_v)), \\ 0 & (e = v). \end{cases} \tag{3}$$

$T_{e,v} = 1$ means different flows $\mathcal{K}_e$ and $\mathcal{K}_v$ are held at the same bucket. Thus, the expectations about the hash collision probability is:

$$E(T_{e,v}) = Pr\left\{(\mathbb{H}(\mathcal{K}_e) = \mathbb{H}(\mathcal{K}_v)) \ \ and \ \ (e \neq v)\right\} \leq \frac{1}{s}. \tag{4}$$

We define a random variable Y_e to represent estimation error caused by hash collision in $\mathbb{H}(\mathcal{K}_e)$ as follows:

$$Y_e = \sum_{v=1}^{n} T_{e,v}\cdot f_v, \tag{5}$$

where n is the total number of flows. If the arrival packet belongs to flow $\mathcal{K}_e$, the Counter is incremented by 1. Otherwise, the Counter will be replaced with a certain probability. Thus, we have the range of estimated size as follows:

$$f_e - Y_e \leq \hat{f}_e \leq f_e. \tag{6}$$

If all packets that do not belong to flow $\mathcal{K}_e$ replace this flow, then $\hat{f}_e = f_e - Y_e$. Otherwise, $\hat{f}_e = f_e$. The following shows the expected number of hash collisions:

$$\begin{aligned} E(Y_e) &= E(\sum_{v=1}^{n} T_{e,v}\cdot f_v) \\ &= \sum_{v=1}^{n} f_v \cdot E(T_{e,v}) \\ &\leq \frac{\mathbb{N}}{s}. \end{aligned} \tag{7}$$

Given a random small number ζ, where $\zeta > 0$, we have

$$\begin{aligned} Pr\left\{\hat{f}_e \leq f_e - \zeta\cdot\mathbb{N}\right\} &\leq Pr\left\{f_e - Y_e \leq f_e - \zeta\cdot\mathbb{N}\right\} \\ &\leq Pr\left\{Y_e \geq \zeta\cdot\mathbb{N}\right\} \\ &\leq Pr\left\{\frac{Y_e}{E(Y_e)} \geq \zeta\cdot s\right\}. \end{aligned} \tag{8}$$

Based on *Markov inequality*, we can obtain the formula as follow:

$$\begin{aligned} Pr\left\{\hat{f}_e \leq f_e - \zeta\mathbb{N}\right\} &\leq Pr\left\{\frac{Y_e}{E(Y_e)} \geq \zeta\cdot s\right\} \\ &\leq \frac{E\left(\frac{Y_e}{E(Y_e)}\right)}{\zeta\cdot s} \\ &= \left(\frac{1}{s\cdot\zeta}\right). \end{aligned} \tag{9}$$

Therefore, we have

$$\begin{aligned} Pr\left\{\hat{f}_e \geq f_e - \zeta\mathbb{N}\right\} &= 1 - Pr\left\{\hat{f}_e \leq f_e - \zeta\mathbb{N}\right\} \\ &\geq 1 - \left(\frac{1}{\zeta\cdot s}\right). \end{aligned} \tag{10}$$

In *wSketch*, *Elephant part* has d rows, so the boundary of estimation error is:

$$Pr\left\{\hat{f}_e \geq f_e - \zeta\right\} \geq 1 - \prod_{r=1}^{d}\left(\frac{1}{s\cdot\zeta}\right).$$

IV. EVALUATION

A. EXPERIMENT SETUP

We have implemented *wSketch* on server equipped with 16-core (Intel(R) Core(TM) i7-10700 server @ 2.90GHz 2.90 GHz and 16GB total System memory). All methods used in the comparative experiment are implemented in C++ and executed on the same platform. In addition, all methods are allocated the same memory when conducting experiments, and each method is further subdivided according to its own design. For *wSketch* that deploys on server, the column number w of *Elephant part* is determined by the allocated memory size, while *Mice part* is fixed in a small size.

In the experiment, we compared *wSketch* with Count Min Sketch (CM), CountMax (CMax), HashPipe (HP), and Heavy Keeper (HK) respectively.

B. DATASET

For the experiments, we utilize the publicly accessible University of Wisconsin Data Center Measurement trace dataset. More precisely, we employ the UNI1 dataset [43] for our experimentation.

C. METRICS

1) ***Precision:*** Precision is expressed as $\frac{\eta}{k}$, where η denotes the number of correctly detected top-k flows. *Precision* serves as the accuracy in identifying top-k flows, with higher precision indicating superior performance.
2) ***Average Relative Error (ARE):*** ARE is expressed as $\frac{1}{|\Psi|}\sum_{f_i\in\Psi}\frac{\hat{n_i}-n_i}{n_i}$, where Ψ represents the set of top-k flows, $\hat{n_i}$ denotes the estimated size of flow f_i, and n_i signifies the real size of flow f_i. ARE quantifies the error rate in reported flow size estimates, with lower values indicating superior performance.
3) ***Average Absolute Error (AAE):*** AAE is expressed as $\frac{1}{|\Psi|}\sum_{f_i\in\Psi}|\hat{n_i}-n_i|$, where Ψ, $\hat{n_i}$, and n_i carry the same definitions as in ARE. AAE gauges the accuracy of size estimates, with superior performance indicated by lower AAE values.
4) ***Throughput:*** We perform insertions of the packets at each window and calculate the throughput. The throughput is defined as $\frac{\mathbb{N}}{\mathbb{T}}$, where $\mathbb{N}$ is the total number of packets, and $\mathbb{T}$ is the measured time at each window. We use Million of insertions per second (Mps) to measure the throughput.

D. EXPERIMENT RESULTS

In this section, we perform an experimental comparison between *wSketch* and several existing methods, evaluating their performance across three key metrics: Precision, Average Relative Error (ARE), and Average Absolute Error (AAE).

Throughout all experiments, we conduct 5 sets of experiments. In each set of experiments, we measure 20 results, remove the best and worst results, and average the remaining 18 data as the result of one group. Finally, we derive the average result of the 5 groups as the final measurement result.

1) PERFORMANCE OF WSKETCH AT EACH WINDOW MEASUREMENTS.

We conducted a comparative analysis of accuracy, ARE (Absolute Relative Error), and AAE (Average Absolute Error) measured by various methods in each time window, as depicted in FIGURE 9. It is evident from FIGURE 9a that *wSketch* consistently exhibits the highest accuracy across all window units compared to other methods. For instance, considering the detection performance in window 1, denoted as $w1$ in FIGURE 9a, the accuracy rates for HK, HP, CMax, and CM stand at 82%, 54%, 31%, and 3% respectively, whereas *wSketch* achieves an accuracy of 98%.

The relatively lower accuracy observed with the traditional Count-Min (CM) sketch can be attributed to the allocation of a small and fixed memory space. This results in an increased frequency of hash collisions during the measurement process, leading to overestimation issues. In contrast, *wSketch* employs a probability-based approach within the "*Elephant part*" to adjust the replacement process when hash collisions occur. Moreover, by integrating the "*Mice part*," *wSketch* effectively preserves large flows, thereby yielding more accurate estimation results.

In the experiment concerning ARE, as depicted in FIGURE 9b, the estimated error of *wSketch* is notably lower compared to other methods. Specifically, the ARE of *wSketch* is 2 to 8 times lower than HK, 8 to 20 times lower than HP, and 7 to 15 times lower than CMax. Moreover, as illustrated in FIGURE 9c, itwSketch consistently outperforms all other methods in window detection. The AAE (Average Absolute Error) of *wSketch* is 2 to 15 times lower than HK, 77 to 114 times lower than HP, 36 to 140 times lower than CMax, and more than 200 times lower than CM, respectively.

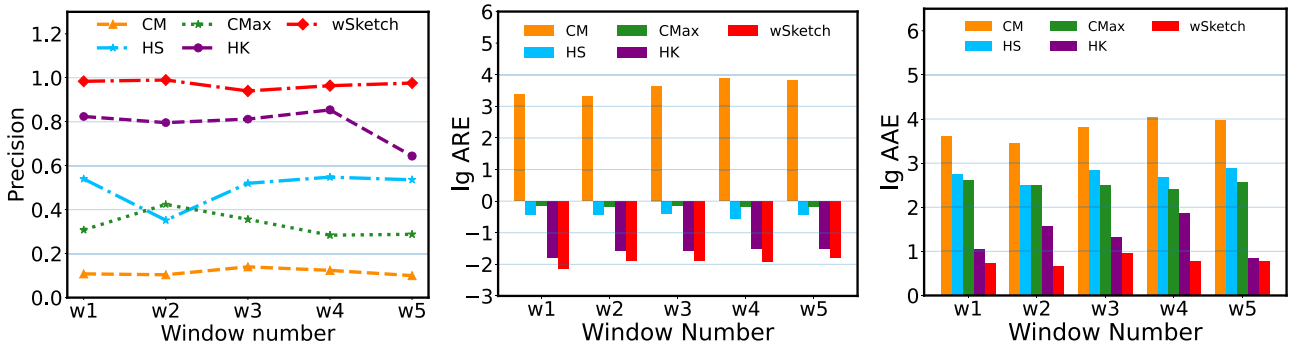
2) PERFORMANCE OF WSKETCH FOR THE PAST FEW WINDOW MEASUREMENTS.

In this section, we evaluated the performance of different methods across the last $\mathbb{W}$ windows, where we set $\mathbb{W} = 5$ for our experiment (this value can be adjusted to suit different scenarios). Specifically, as depicted in FIGURE 10 through FIGURE 12, we assessed the performance over the past one window (referred to as $p1$ in the figures), the past two windows ($p2$), and so forth up to the past five windows ($p5$). Our evaluation encompassed the accuracy and estimation error of various methods under different memory sizes and for different values of k.

In FIGURE 10, with a memory size set to 50KB and a detection threshold of top-50 (i.e., $k = 50$), *wSketch* exhibits the highest accuracy in flow detection and the most precise flow size estimation among all methods. In the past five window unit (i.e., $p2$ in figure), the accuracy of CM is between 6.3% ~ 8%, CMax is 62.5% ~ 64%, HS is 56.3% ~ 64.7%, HK is 87.4% ~ 95%, and *wSketch* reaches 96.7% ~ 87.8%. The ARE of *wSketch* in the past five window units is 84 to 123 times lower than CMax, 80 to 102 times lower than HP, 1.16 to 1.8 times lower than HK, and more than a thousand times lower than CM. In addition, *wSketch* also performs well in the measurement results of AAE. The AAE of *wSketch* is 150 to 302 times lower than CMax, 195 to 381 times lower than HP, 1.2 to 3 times lower than HK, and much lower than CM.

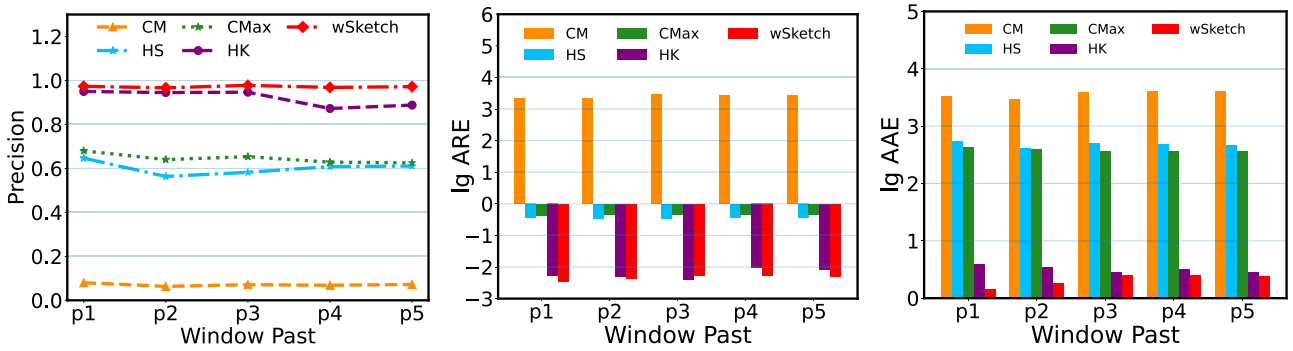
As shown in FIGURE 11, when the memory is 50KB and the top-100 is detected, the accuracy of CM for the past two window units is 7%, CMax is 48%, HP is 57%, HK is 86%, and *wSketch* reaches 95%. As shown in FIGURE 10b, the ARE of *wSketch* is 0.01636, which is 34 times lower than CMax, 20 times lower than HP, 3 times lower than HK, and more than a thousand lower than CM. The AAE of *wSketch* is also much lower than the compared methods.

When k remains unchanged and memory increases to 100KB as illustrated in FIGURE 12, the accuracy of all methods improves, but overall, *wSketch* performs best. The



(a) Precision. (b) ARE. (c) AAE.

FIGURE 9. Performance of *wSketch* for each window when memory is fixed to 20KB and k is fixed to 50.



(a) Precision. (b) ARE. (c) AAE.

FIGURE 10. Performance of *wSketch* on the past few windows when memory is fixed to 50KB and k is fixed to 50.

accuracy of each method improved. In the past two windows, the accuracy of CM is 12%, CMax is 64%, HP is 63%, HK is 96%, and *wSketch* reaches 98.6%. For the estimation error, the ARE of *wSketch* is 64 times lower than CMax, 38 times lower than HP, 1,2 times lower than HK, and much lower than CM. The results of AAE are similar to that of ARE.

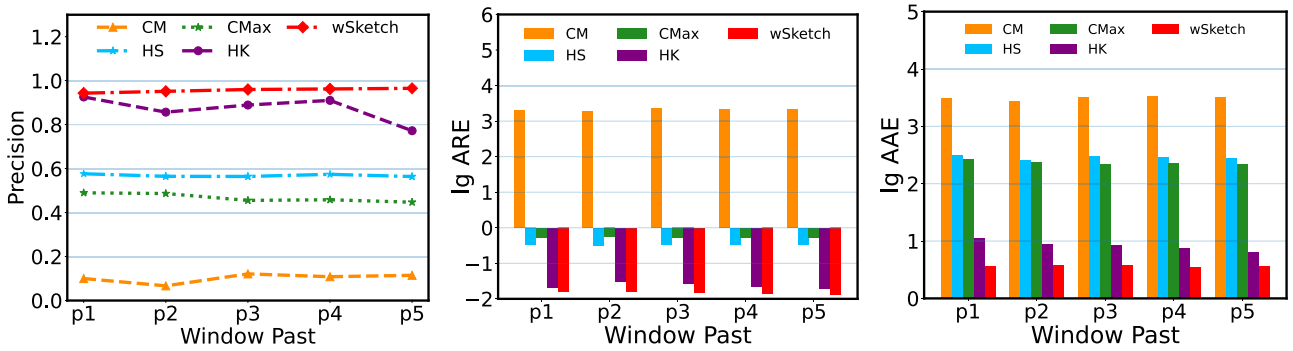
As shown in the comparison of FIGURE 10a and FIGURE 11a, as k increases while memory size remains constant, the accuracy of comparative methods experiences a decline to some extent, whereas *wSketch* maintains consistently high accuracy. This emphasizes *wSketch*'s adaptability to scenarios with limited memory and large traffic compared to other methodologies. This adaptability stems from *wSketch*'s utilization of probabilistic mechanisms for flow replacement, ensuring that *elephant flows* are more likely to be retained, thereby enhancing measurement accuracy. Conversely, *mice flows*, which are easily replaced and overlooked, help mitigate memory overhead. Despite the *mice flows* having a small individual size, the sheer volume necessitates a certain level of memory allocation for accurate preservation and differentiation.

In all, the experimental results demonstrate that *wSketch* maintains high accuracy and low estimation error even with limited SRAM and a large value of k. By employing the probabilistic update sketch approach, *wSketch* effectively prioritizes the retention of large flows over small ones. Additionally, the incorporation of the *Mice part* aids in enhancing the accuracy of top-k flows detection. *wSketch* adopts a Circular Queue detection structure, further minimizing memory overhead for detection purposes. Ultimately, *wSketch* excels in achieving precise measurements with constrained memory resources. In addition, we conduct a brief mathematical analysis of the error boundary of *wSketch*, and in fact, the error boundary of *wSketch* is smaller than the analyzed values.

E. PERFORMANCE OF WSKETCH ON THROUGHPUT

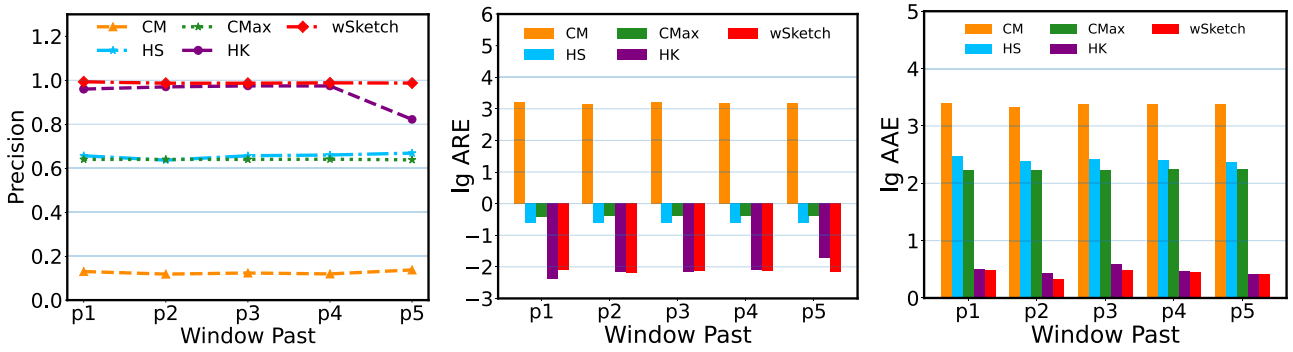
In this section, we measure the performance of the throughput of different methods. Since we focus on the measurement of the past $\mathbb{W}$ windows, where each window has the same time period. Therefore, the throughput is defined as the packet number that can be processed within a single window.

As shown in FIGURE 13, the throughput of *wSketch* performs relatively well among these methods. In fact, the experimental results of throughput are related to the number of packets processed, while the speed of packet processing is closely related to factors such as the server equipment used. If the server performance is good, the throughput will also increase. Therefore, we are more concerned about the throughput performance of *wSketch* compared to other methods, and whether it will lead to a decrease in throughput. Experimental result proves that *wSketch* will not cause the



(a) Precision. (b) ARE. (c) AAE.

FIGURE 11. Performance of *wSketch* on the past few windows when memory is fixed to 50KB and k is fixed to 100.



(a) Precision. (b) ARE. (c) AAE.

FIGURE 12. Performance of *wSketch* on the past few windows when memory is fixed to 100KB and k is fixed to 100.

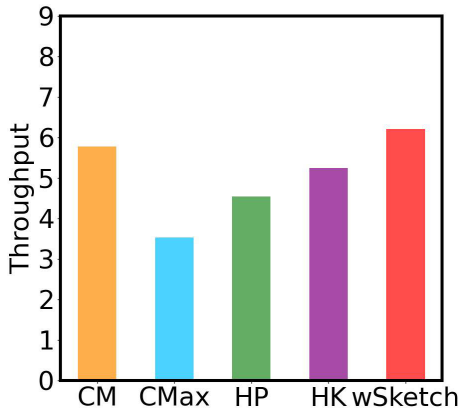


FIGURE 13. Performance of throughput.

processing speed to drop significantly. Some other methods include min-heap adjustment, etc., so the throughput drops obviously. Overall, *wSketch* performs well in terms of processing speed and throughput.

V. CONCLUSION

The identification of top-k flows in high-speed networks is an important yet challenging issue in many security analysis scenarios. In real networks, the latest flows are often more important than historical flows. In this paper, we propose *wSketch*, a sliding window sketch method that combines the sliding window model, circular queue model, and probability calculation to detect real-time top-k flows. *wSketch* combines the advantages of sketch and sliding window, focusing on the latest flow statistic, while the circular queue reduces the cost of memory consumption by reusing memory. Besides, the use of probability calculation assigns different weights for large flows and small flows, resulting in higher retention of large flows, while small flows are ignored and not recorded, ultimately reducing memory overhead further. We conducted different experiments, and the results show that *wSketch* is superior to other methods and can cope with the gradual increase in volume in network traffic with low memory overhead.

There is also an ability *wSketch* needs to improve. For example, due to the use of sliding windows and relatively complex calculations (i.e., probability calculation), *wSketch* has limitations when migrating to different platforms, such as being deployed on a programmable data plane. In future work, we can consider approximating *wSketch* so as to migrate it to the data plane for faster processing speed. Meanwhile, the probabilistic approach adopted by *wSketch*

can be applied to INT measurement. Through probabilistic reverse inference, the packets can reduce the amount of information collected at network nodes, and thus reduce bandwidth overhead.

REFERENCES

[1] D. Yu, "dShark: A general, easy to program and scalable framework for analyzing in-network packet traces," in *Proc. 16th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2019, pp. 207–220.

[2] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 139–152.

[3] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2011, pp. 1–12.

[4] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "HPCC: High precision congestion control," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 44–58.

[5] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 1–16.

[6] G. Dittmann and A. Herkersdorf, "Network processor load balancing for high-speed links," in *Proc. Int. Symp. Perform. Eval. Comput. Telecommun. Syst.*, vol. 735, 2002, pp. 1–9.

[7] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "A fast and reliable software network load balancer," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 523–535.

[8] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown, "Programmable packet scheduling at line rate," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 44–57.

[9] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proc. Int. Conf. Manage. Data*, Jun. 2016, pp. 1449–1463.

[10] L. Jie, C. Hongchang, S. Penghao, H. Tao, and Z. Zhen, "OrderSketch: An unbiased and fast sketch for frequency estimation of data streams," *Comput. Netw.*, vol. 201, Dec. 2021, Art. no. 108563.

[11] A. Lakhina, M. Crovella, and C. Diot, "Characterization of network-wide anomalies in traffic flows," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, Oct. 2004, pp. 201–206.

[12] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *Proc. 6th ACM SIGCOMM Conf. Internet Meas.*, Oct. 2006, pp. 147–152.

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

[14] B. Chen, Z. Lv, X. Yu, and Y. Liu, "Sliding window top-K monitoring over distributed data streams," *Data Sci. Eng.*, vol. 2, no. 4, pp. 289–300, Dec. 2017.

[15] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *Proc. Symp. SDN Res.*, Mar. 2018, pp. 1–7.

[16] R. Schweller, A. Gupta, E. Parsons, and Y. Chen, "Reversible sketches for efficient and accurate change detection over network data streams," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, Oct. 2004, pp. 207–212.

[17] M. Etemadi, M. Ghobaei-Arani, and A. Shahidinejad, "A cost-efficient auto-scaling mechanism for IoT applications in fog computing environment: A deep learning-based approach," *Cluster Comput.*, vol. 24, no. 4, pp. 3277–3292, Dec. 2021.

[18] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, M. Masdari, and H. Shakarami, "Data replication schemes in cloud computing: A survey," *Cluster Comput.*, vol. 24, no. 3, pp. 2545–2579, Sep. 2021.

[19] Z. Zeng, L. Cui, M. Qian, Z. Zhang, and K. Wei, "A survey on sliding window sketch for network measurement," *Comput. Netw.*, vol. 226, May 2023, Art. no. 109696.

[20] M. H. ur Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, and S. U. Khan, "Big data reduction methods: A survey," *Data Sci. Eng.*, vol. 1, no. 4, pp. 265–284, Dec. 2016.

[21] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, Nov. 2010, pp. 267–280.

[22] T. Yang, H. Zhang, J. Li, J. Gong, S. Uhlig, S. Chen, and X. Li, "HeavyKeeper: An accurate algorithm for finding top-k elephant flows," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 1845–1858, Oct. 2019.

[23] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *Proc. IEEE INFOCOM 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[24] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proc. Int. Conf. Database Theory*, 2004, pp. 398–412.

[25] *SFlow*. Accessed: May 15, 2023. [Online]. Available: https://sflow.org/

[26] *Networks, C. NetFlow*. Accessed: May 15, 2023. [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[27] J. Lu, H. Chen, and Z. Zhang, "LUSketch: A fast and precise sketch for top-k finding in data streams," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2022, pp. 1–10.

[28] J. Lu, H. Chen, Z. Zhang, Z. Zhang, and L. Wang, "Sketch-BF: A fast algorithm for finding top-k flows," *Electron. Lett.*, vol. 58, no. 11, pp. 429–431, May 2022.

[29] J. Huang, W. Zhang, Y. Li, L. Li, Z. Li, J. Ye, and J. Wang, "ChainSketch: An efficient and accurate sketch for heavy flow detection," *IEEE/ACM Trans. Netw.*, vol. 31, no. 2, pp. 738–753, Apr. 2023.

[30] H. Huang, J. Yu, Y. Du, J. Liu, H. Dai, and Y.-E. Sun, "Memory-efficient and flexible detection of heavy hitters in high-speed networks," *Proc. ACM Manage. Data*, vol. 1, no. 3, pp. 1–24, Nov. 2023.

[31] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM J. Comput.*, vol. 31, no. 6, pp. 1794–1813, Jan. 2002.

[32] S. Li, L. Luo, D. Guo, Q. Zhang, and P. Fu, "A survey of sketches in traffic measurement: Design, optimization, application and implementation," 2020, *arXiv:2012.07214*.

[33] G. S. Manku and R. Motwani, "Approximate frequency counts over data stream," in *Proc. 28th Int. Conf. Very Large Databases*, 2002, pp. 346–357.

[34] R. Shahout, R. Friedman, and D. Adas, "CELL: Counter estimation for per-flow traffic in streams and sliding windows," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–12.

[35] B. Xiong, Y. Liu, R. Liu, J. Zhao, S. He, B. Zhao, K. Yang, and K. Li, "ActiveGuardian: An accurate and efficient algorithm for identifying active elephant flows in network traffic," *J. Netw. Comput. Appl.*, vol. 224, Apr. 2024, Art. no. 103853.

[36] Q. Shi, Y. Xu, J. Qi, W. Li, T. Yang, Y. Xu, and Y. Wang, "Cuckoo counter: Adaptive structure of counters for accurate frequency and top-k estimation," *IEEE/ACM Trans. Netw.*, vol. 31, no. 4, pp. 1854–1869, Aug. 2023.

[37] X. Yu, H. Xu, D. Yao, H. Wang, and L. Huang, "CountMax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2774–2786, Dec. 2018.

[38] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "FCM-sketch: Generic network measurements with data plane support," in *Proc. 16th Int. Conf. Emerg. Netw. EXperiments Technol.*, Nov. 2020, pp. 78–92.

[39] P. Chen, D. Chen, L. Zheng, J. Li, and T. Yang, "Out of many we are one: Measuring item batch with clock-sketch," in *Proc. Int. Conf. Manage. Data*, Jun. 2021, pp. 261–273.

[40] Z. Sun, Y. E. Sun, and Y. Du, "Persistent sketch: A memory-efficient and robust algorithm for finding top-k persistent flows," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.* Singapore: Springer, 2023, pp. 19–38.

[41] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. Symp. SDN Res.*, Apr. 2017, pp. 164–176.

[42] A. Shahidinejad, M. Ghobaei-Arani, A. Souri, M. Shojafar, and S. Kumari, "Light-edge: A lightweight authentication protocol for IoT devices in an edge-cloud environment," *IEEE Consum. Electron. Mag.*, vol. 11, no. 2, pp. 57–63, Mar. 2022.

[43] *DataSet for IMC 2010 Data Center Measurement*. Accessed: May 15, 2023. [Online]. Available: https://pages.cs.wisc.edu/tbenson/~IMC10_Data.html

KEKE ZHENG received the B.E. degree from Guangzhou University, Guangzhou, China, in 2021. She is currently pursuing the master's degree with the Department of Computer Science, Jinan University. Her current research interests include network measurement, stateful data plane/programmable data plane, and software-defined networking (SDN).

WENZHU CHEN received the B.S. degree from the School of Mathematics and Statistics, Zhaoqing College of China, in 2021. She is currently pursuing the M.S. degree with the College of Information Science and Technology, Jinan University, Guangzhou, China. Her current research interests include machine learning, computer vision, and computer networks.

BOTAO FENG (Senior Member, IEEE) was born in Guangdong, China, in 1980. He received the B.S. and M.S. degrees in communication engineering from Chongqing University of Posts and Telecommunications (CQUPT), Chongqing, China, in 2004 and 2009, respectively, and the Ph.D. degree in communication and information system from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2015.

He joined the Dongguan Branch, Nokia Mobile Phones Ltd., China, as a Senior Communication Engineer, in 2004. From 2009 to 2012, he was a Senior Research Fellow and a Chief Executive with Guangdong Branch, China United Network Communications Company Ltd., where he received the Award of Breakout Star of the Year and the title of a Technical Innovation Expert. He is currently a Postgraduate Advisor and a Postdoctoral Advisor with Shenzhen University, Guangdong, China. He is also the Head of the Laboratory of Wireless Communication, Antennas and Propagation, Shenzhen University; also the Deputy Director of the Department of Electronic Science and Technology; also the Director of the Joint Laboratory of Antenna and Electromagnetic Propagation, Shenzhen University, Rihai Communication Technology Company Ltd.; also the Director of the Joint Laboratory, Shenzhen Zhongke Wireless Technology Company Ltd., and the Group of Wireless Communication, Antennas and Propagation, Shenzhen University; also the Director of the Joint Laboratory of Antenna and Microwave Technology, Shenzhen University, Skywave Communication Technology Company Ltd.; also the Chief Scientist with Shenzhen Nandouxing Technology Company Ltd.; and the President of Shenzhen Taobida Technology Company Ltd. His research team members are currently conducting more than 20 projects on antenna development and design for 5G/THz and future communications, which are supported by natural science research funds and industrial cooperation research and development funds. His several antenna designs for 5G applications have been widely used by Chinese communication operators. He has authored or coauthored more than 50 science citation index (SCI)- and engineering index (EI)-articles and holds more than 20 invention patents. It is estimated that the related total production value is approximately 200 million Ren Min Bi (RMB). His research interests include antennas and mobile communications.

Dr. Feng has obtained the Award of the Outstanding Instructor of the First Prize in National Graduate Electronic Contest and has been the Tencent Outstanding Teacher Award, since 2017. He also serves as a regular peer reviewer, a technical committee member, the section chair, and a Guest Editor for IEEE/IET, Elsevier, Wiley, and Springer journals and conferences on microwave technique and antenna development.

HANXIN ZHANG was born in Guangdong, China, in 1999. He received the B.E.E. degree in automation from the University of Science and Technology Beijing, Beijing, China, in 2017. He is currently pursuing the M.S. degree with the School of Automation Science and Electrical Engineering, Beihang University, Beijing. His research interests include control strategy of PM machines, induction machine, and computer science.