Computer Networks 270 (2025) 111562

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

AF-Detector: An accurate low-overhead method for detecting active flows in network traffic

Bing Xiong [a], Yu Chang [a], Yuhang Zhang [a], Jin Zhang [a,*], Baokang Zhao [b,*], Keqin Li [c,*]

[a] School of Computer Science and Technology, Changsha University of Science and Technology, Changsha 410114, PR China
[b] School of Computer, National University of Defense Technology, Changsha 410073, PR China
[c] Department of Computer Science, State University of New York at New Paltz, New York 12561, USA

ARTICLE INFO

Keywords:
Network traffic measurement
Active flows
Probabilistic replacement strategy
Information cleanup mechanism

ABSTRACT

Active flows refer to packet flows whose frequencies consistently exceed a certain threshold for multiple consecutive time windows. They occupy the majority of network traffic and have a great impact on network performance. Previous work on detecting active flows only recorded the persistence of flows with low accuracy and high memory overhead, unable to report the activity periods and real-time frequencies of active flows. To address these issues, we propose an accurate low-overhead method for detecting active flows called AF-Detector, which separates the tracking and identification functions of active flows, enabling precise detecting and reporting of active flows. In particular, AF-Detector consists of a monitoring part that employs a compact hash table to track active, and an identification part that adopts a probabilistic data structure to estimate flow frequency and find out potentially active flows. As for the monitoring part, we design a probabilistic replacement strategy to accommodate new potentially active flows and clear out low-frequency flows, for accurately detecting active flows. Additionally, we devise an information cleanup mechanism to clear out flows that are no longer active, and outdated information at the end of each time window. Finally, we evaluate the performance of AF-Detector by theoretical analysis and experiments verification with real network traffic traces. Experimental results indicate that AF-Detector performs better than the state-of-the-art methods with the precision 99.59% and the recall rate 99.78%.

1. Introduction

In recent years, network measurement [1–3] has become a research topic in the network field. Previous network measurement efforts mainly focused on frequency estimation [4,5], elephant flows identification [6,7] and persistence estimation [8]. Nowadays, active flows, as key flows in network traffic, have been receiving increasing attention. Active flows refer to flows whose frequencies consistently exceed a certain threshold for multiple consecutive time windows. They constitute the majority of network traffic and have a considerable impact on network performance. Detecting active flows accurately is an essential task for network management and optimization of network performance, especially in areas such as congestion control [9], load balance [10–12], traffic scheduling [13], network security [14–16], and network capacity planning [17]. In network congestion control, real-time detection of active flows allows network systems to quickly detect the flow that causes congestion and take appropriate actions, such as applying rate limiting or adjusting routing, to mitigate the impact on network performance. In Quality of Service (QoS) [18,19] assurance, real-time detection of active flows in network security allows network administrators to prioritize critical business traffic such as video conferencing and VoIP calls when multiple applications concurrently request network resources. This ensures the appropriate allocation of network resources.

Most existing methods for detecting active flows are designed based on probabilistic data structure sketch, such as On-off Sketch [20] and Waving Sketch [21]. On-off Sketch estimates the persistence of flows by compressing increments when multiple flows are mapped to the same counter. Waving Sketch designs an unbiased estimation method to find out active flows. However, On-off Sketch and Waving Sketch only detect the persistence of flows, without considering the real-time frequency of the flow. This results in some persistent but low-frequency flows occupying a large amount of storage space, preventing some persistent and high-frequency flows from being recorded. To overcome this issue, FastKeeper [22] employs a sliding-window-based method to measure the real-time frequencies of flows and detect active flows. However, the highly skewed nature of network traffic results in a

* Corresponding authors.
E-mail addresses: jzhang@csust.edu.cn (J. Zhang), bkzhao@nudt.edu.cn (B. Zhao), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.comnet.2025.111562
Received 8 April 2025; Received in revised form 6 July 2025; Accepted 15 July 2025
Available online 26 July 2025

significant number of inactive flows interfering with the detection of active flows, leading to low accuracy of active-flow detection. To address this issue, Burst Sketch [23] adopts the running track technique to reduce the interference of inactive flows. However, Burst Sketch records the ID and frequency of each flow in detail in the first stage, which consumes excessive memory. In summary, existing methods are unable to accurately detect active flows with low memory overhead.

To address these issues, we propose an accurate low-overhead method of detecting active flows called AF-Detector, which is composed of a monitoring part and an identification part. In the monitoring part, we only record potentially active flows. Inspired by the idea of exponential-weakening decay strategy [24,25], we propose a probabilistic replacement strategy for the monitoring part, which allows new potentially active flows to replace inactive flows in a probabilistic way. To filter inactive flows and find out potentially active flows, we apply a probabilistic data structure sketch to estimate the frequencies of flows in the identification part, helping the monitoring part to more accurately track active flows. Additionally, we devise an information cleanup mechanism for both the monitoring and the identification parts to clear out flows that are no longer active and outdated information at the end of each time window, freeing up memory space for new potentially active flows and enhancing the accuracy of active-flow detection. By separating the functions of identification and monitoring, AF-Detector can effectively filter out inactive flows in the identification part, and prevent them from interfering with the tracking of active flows in the monitoring part. Therefore, the monitoring part can focus on recording the information of active flows, which contributes to accurate detection of active flows. Our main contributions are as follows:

- Proposing a novel method for detecting active flows called AF-Detector, which tracks active flows by the monitoring part and finds out potentially active flows by the identification part, to achieve high accuracy and low overhead of active-flow detection.
- Designing a probabilistic replacement strategy for the monitoring part, which allows new potentially active flows to replace low-frequency flows in a probabilistic way for the case of full mapped buckets, to enhance the accuracy of active-flow detection.
- Devising an information cleanup mechanism for the monitoring and identification parts, which resets their frequency counters at the end of each time window, and clears out flows that are no longer active in the monitoring part to free memory space for new potentially active flows.
- Proving that AF-Detector can provide the precise detection of active flows, by performing theoretical analysis on the error bounds of its flow frequency estimation, false positive error rate and false negative error rate.

The rest of the paper is organized as follows. Section 2 introduces the definition of active flow and related work of active-flow detection. Section 3 introduces the framework, data structures, and running examples of AF-Detector, and presents two optimized versions. Section 4 provides theoretical analysis on the error bounds of its flow frequency estimation, its false positive and false negative rates, and its space and time complexities. Section 5 describes the experimental setup and evaluates the performance of AF-Detector on real network traffic traces. Section 6 concludes the paper and discusses future work.

2. Definition of active flows and related work

2.1. Definition of active flows

Suppose there is a sequential packet traffic $S = \{e_1, e_2, e_3, \dots\}$, divided by multiple fixed-size time windows $w_1, w_2, w_3, \dots$. For a packet flow e, let its packet arrival rates in the time windows be $r_1, r_2, r_3, \dots$. If there are multiple consecutive time windows $w_i, w_{i+1}, \dots, w_{i+k}$ of the flow e, meeting the following condition:

$$k + 1 \geqslant \beta \wedge \forall r_j \geqslant \alpha, j \in \{i, \dots, i + k\}$$

where α indicates the frequency threshold of an active flow in a time window, and β denotes the number threshold of consecutively active time windows of an active flow. Then the flow e is determined as an active flow with the activity periods from w_i to w_{i+k}. Table 1 shows the symbols commonly used in the AF-Detector and their meanings.

2.2. Related work

Frequency estimation is a fundamental function of active-flow detection. Previous frequency estimation methods are mostly designed based on probabilistic data structure sketch, such as Count-Min Sketch [26] and Count Sketch [27]. Count-Min Sketch is composed of a $d \times w$ two-dimensional array and d mutually independent hash functions. When a packet arrives, it maps the packet into d counters by hashing and increments them by 1. As for the query of a flow, the minimum of its mapped counters is reported as its estimated frequency. Count Sketch is similar to Count-Min Sketch, but it incorporates an additional bool function to increment or decrement the mapped counters of a packet. As for the query of a flow, the median of its mapped counters is taken as its estimated frequency.

Pyramid Sketch [28] employed a pyramid-shaped data structure to automatically enlarge the size of counters in accordance with the frequency of an incoming item, enhancing the accuracy of frequency estimation with low memory overhead. Elastic Sketch [29] divided network traffic into elephant and mice flows, and separately stored them into heavy and light parts. It further compressed and merged data structures to reduce bandwidth usage while maintaining the accuracy of frequency estimation. However, these methods can only estimate the frequency of a flow in entire data stream, but cannot detect the real-time status of a flow. Consequently, traditional frequency estimation methods cannot be directly applied to detect active flows.

In recent years, some researchers have proposed new methods to detect active flows in real time. Ada-Sketch [30] proposed a time-adaptive method that dynamically adjusts the size of its data structures to measure recent active flows based on changes in the data stream. However, Ada-Sketch is difficult to apply in scenarios with limited memory, due to that it needs a large number of counters. On-off Sketch [20] took advantage of the characteristic that the persistence of a flow is increased periodically. It compressed increments to accurately estimate the persistence of flows and record active flows when multiple flows are mapped to a same counter. Waving Sketch [21] applied a bloom filter [31] to filter out duplicates in its first stage, and designed an unbiased estimation method to keep track of active flows in its second stage. However, On-off Sketch and Waving Sketch do not consider the real-time frequencies of flows, resulting in the miss of some active flows. FastKeeper [22] adopted a sliding-window-based method to track the real-time frequencies of flows and employed bitmap-voting algorithm to promptly replace inactive flows. However, these methods cannot report active flows, and have low accuracy in detecting active flows due to interference from a large number of inactive flows. Burst Sketch [23] applied the running track technique to effectively filter potential burst flows, and employed Snapshotting technology to capture burst active flows and report these flows at the end of each time window. However, Burst Sketch recorded the ID and frequency of each flow in its first stage, resulting in the waste of memory. In summary, existing detection methods of active flows cannot report their activity periods at high accuracy with low memory overheads.

In summary, existing methods face several challenges to achieve accurate detection of active flows. Specifically, methods like On-off Sketch, and Waving Sketch emphasize flow persistence estimation but overlook real-time frequency. Other methods such as FastKeeper and Burst Sketch introduce time-window mechanisms, but either suffer

Table 1
Symbols commonly used in the AF-Detector.

Symbol	Meaning
S_{win}	The size of each time window, indicated by a fixed number of measured packets
F_{cur}	The frequency of a flow in the current time window
W_{cur}	The sequence number of the current time window
$W_{initial}$	The sequence number of the initial active time window of a flow
N_{active}	The number of consecutively active time windows of a flow
α	The frequency threshold of an active flow in a time window
β	The number threshold of consecutively active time windows of an active flow
k	The number of hash functions in the identification part of AF-Detector
p	The probability of replacement in the monitoring part of AF-Detector
r	The replacement rate in probabilistic replacement strategy

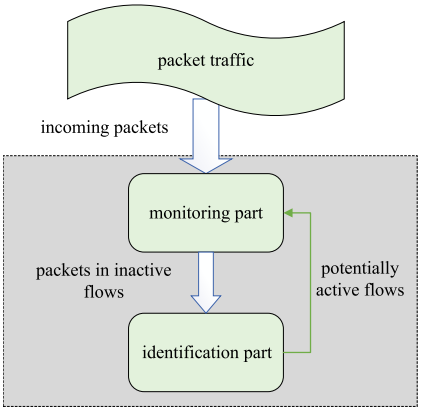

Fig. 1. The framework of AF-Detector.

from low accuracy due to inactive flow interference, or bring about high memory overhead by storing detailed flow records. In addition, these detection methods cannot report the activity periods of active flows. To address these limitations, we propose an accurate low-overhead method of detecting active flows called AF-Detector, which separates the tracking and identification functions of active flows based on time windows, enabling precise detecting and reporting of active flows.

3. The design of AF-Detector

In this section, we first present the framework of AF-Detector. Then, we design the data structure of AF-Detector and demonstrate its running examples. Finally, we propose two optimization strategies for AF-Detector.

3.1. The framework of AF-Detector

Considering that inactive flows account for the majority of packet flows in network traffic, we construct the framework of AF-Detector in Fig. 1, whose key idea is to separate the tracking and identification functions of active flows. The AF-Detector consists of a monitoring part and an identification part. The monitoring part employs a compact hash table to track active flows. We propose a probabilistic replacement strategy for the monitoring part, to accommodate new potentially active flows for the case of full mapped buckets, improving the accuracy of active-flow detection. The identification part adopts a probabilistic data structure to estimate the frequencies of incoming flows and find out potentially active flows. At the end of each time window, we report active flows and clear out outdated information and flows that are no longer active.

As for an arrived packet within a flow f, we first look up the flow f in the monitoring part. If the flow f has been recorded in the monitoring part, we update its frequency in the current time window. Otherwise, we insert the packet into the identification part to further observe whether the flow f is a potentially active flow. If its frequency in the current time window reaches the frequency threshold α, we consider the flow f as a potentially active flow and insert it into the monitoring part. At the end of each time window, we report the information of active flows, and adopt an information cleanup mechanism to clear out flows that are no longer active in the current time window. Finally, we reset all flow frequencies in the monitoring part and all counters in the identification part.

3.2. Monitoring part

The data structure of the monitoring part: As shown in Fig. 2, the monitoring part is designed in the form of a hash table with a hash function $h(\cdot)$. It consists of M buckets $A[1], A[2], A[3], \ldots, A[M]$, each of which contains n cells. Each cell records the information of a flow, including *Flow ID*, F_{cur}, and $W_{initial}$. To save memory, the *Flow ID* only records the fingerprint of a flow identified by 5 tuples: source IP address, destination IP address, source port, destination port, and protocol type. F_{cur} indicates the frequency of a flow in the current time window. $W_{initial}$ indicates the sequence number of the initial active time window of a flow.

Initialization: In initial state, we set all fields to 0 or *NULL*.

Packet processing: As for an arrived packet within a flow f, we first look up the flow f in the monitoring part. If the flow f is recorded in the monitoring part, we increment its frequency in the current time window F_{cur} by 1. Otherwise, we insert the packet into the identification part to further check whether the flow f has become a potentially active flow.

The insertion of a potentially active flow: If the flow f is identified as a potentially active flow, we need to look up the monitoring part for an empty cell in bucket $A[h(f)]$ and place the flow f into it. There are two cases:

Case 1: There is an empty cell in the bucket $A[h(f)]$. In this case, we insert the flow f into the empty cell. Specifically, we set *Flow ID* as f, F_{cur} as its estimated frequency in the current time window, and $W_{initial}$ as the sequence number of the current time window W_{cur}.

Case 2: If there is no empty cell in the bucket $A[h(f)]$, we will find out those flows whose frequencies do not reach the threshold α. Then, we replace a flow with the smallest frequency among these flows by the flow f in a probabilistic way. The replacement probability is designed as $p = e^{-rx}$, where x indicates the frequency of replaced flow and r denotes replacement rate. If the replacement is confirmed to perform, the information of the flow f will replace the information of the flow with the smallest frequency. Otherwise, we discard the flow f.

The report of active flows: At the end of each time window, we traverse each cell in the monitoring part to report current active flows. For each non-empty cell, we check if the flow frequency F_{cur} reaches the frequency threshold α and the number of consecutively active time windows N_{active} ($N_{active} = W_{cur} - W_{initial} + 1$) reaches the number threshold of consecutively active time windows β. There are three cases for the checking result of each non-empty cell:

Case 1: If $F_{cur} < \alpha$, the flow in this cell is no longer active, and we reset all information in this cell.

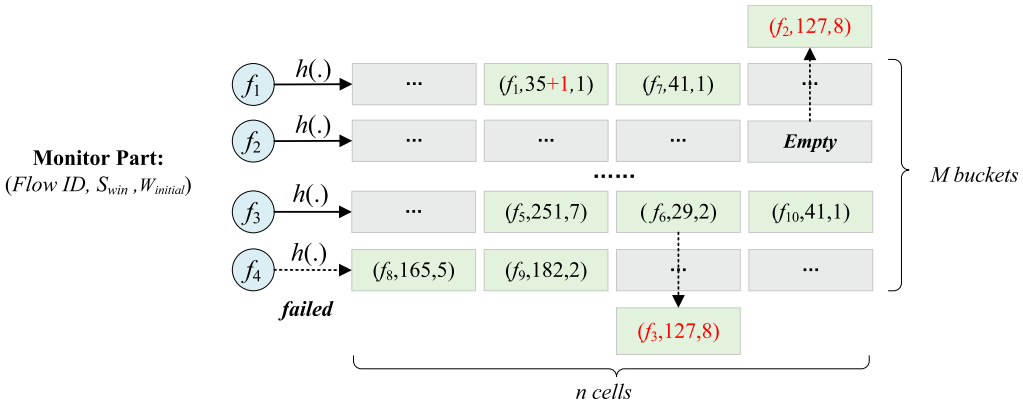

Fig. 2. The data structure and running example of the monitoring part.

Case 2: If $F_{cur} \geq \alpha$ and $N_{active} < \beta$, the flow in this cell still is a potentially active flow, and we do not report the flow. Meanwhile, we reset flow frequency in this cell.

Case 3: If $F_{cur} \geq \alpha$ and $N_{active} \geq \beta$, the flow in this cell is confirmed as an active flow, and we report the flow including its latest activity period and frequency in current time window. Finally, we reset flow frequency in this cell.

A running example: Fig. 2 exhibits a running example of the monitoring part. Assume that the frequency threshold of an active flow in a time window $\alpha = 127$, the number threshold of consecutively active time windows of an active flow $\beta = 4$, and the sequence number of current time window $W_{cur} = 8$. The insertion of f_1, f_2, f_3, and f_4 can be described as follows.

(1) To insert f_1 which is recorded in the monitoring part, we only need to increment its frequency by 1.

(2) To insert f_2 identified as a potentially active flow, we find out an empty cell in the bucket $A[h(f_2)]$ and insert $(f_2, 127, 8)$ into the empty cell.

(3) To insert f_3 identified as a potentially active flow, we choose a flow in the bucket $A[h(f_3)]$ with the smallest frequency and replace it with the probability p, since there is no empty cell in the bucket $A[h(f_3)]$. If the replacement is determined to execute, the cell $(f_6, 29, 2)$ will be changed into $(f_3, 127, 8)$.

(4) The insertion of f_4 is similar to that of f_3, but the flow f_4 does not replace any flow in the bucket $A[h(f_4)]$ and is discarded.

At the end of the time window, we report active flows and clean up flows that are no longer active. For the flows f_1, f_7, and f_{10}, their frequencies in the current time window do not reach the frequency threshold α. Thus, we reset all information in the corresponding cells. For the flows f_2, f_3, and f_5, their frequencies in the current time window reach the frequency threshold α, but their number of consecutively active time windows does not reach the threshold β. Accordingly, we do not report these flows. For the flows f_8 and f_9, they meet the criteria of active flows. So, we report them with their activity periods of $5 \sim 8$ and $2 \sim 8$, and frequencies of 165 and 182, respectively. After reporting, we reset all flow frequencies in the monitoring part to 0.

3.3. Identification part

The data structure of the identification part: As shown in Fig. 3, the identification part is an array B with N counters, associated with k mutually independent hash functions $H_1(.)$, $H_2(.)$, $H_3(.) \ldots H_k(.)$. These counters are used to estimate the frequencies of flows. At the end of each time window, we reset all counters.

Initialization: In initial state, we set all counters as 0.

The insertion of a packet: As for an arrived packet within a flow f, it is first mapped to the k counters $B[H_i(f)]$ (where $1 \leq H_i(f) \leq N$, $1 \leq i \leq k$) in the identification part by hashing. Next, we select the minimal counter among the k counters ($\min\{B[H_i(f)], 1 \leq i \leq k\}$) and increase it by 1. If there are multiple minimal counters, we increment each of them by 1. After the increment, we check if the updated counter reaches the frequency threshold α. There are two cases:

Case 1: $\min\{B[H_i(f)], 1 \leq i \leq k\} < \alpha$. In this case, the flow f is considered as an inactive flow.

Case 2: $\min\{B[H_i(f)], 1 \leq i \leq k\} \geq \alpha$. In this case, the flow f is identified as a potentially active flow, and we insert it into the monitoring part to track it.

A running example: Fig. 3 illustrates a running example of the identification part. Assume that the frequency threshold of an active flow in a time window $\alpha = 127$, and the identification part has 3 hash functions. The insertion of a packet in f_{11}, f_{12}, and f_{13} can be described as follows.

(1) To insert a packet in f_{11}, we obtain its mapped counters in the identification part by hashing, and increase the minimal counter $B[H_1(f_{11})]$ among them by 1. After the increment, we find that $B[H_1(f_{11})]$ goes below the frequency threshold α. Thus, the flow f_{11} is considered as an inactive flow.

(2) To insert a packet in f_{12}, we increase the minimum of its mapped counters $B[H_1(f_{12})]$ and $B[H_2(f_{12})]$ by 1. However, both counters do not reach the frequency threshold α. Hence, the flow f_{12} is also not identified as a potentially active flow.

(3) To insert a packet in f_{13}, the minimum of its mapped counters $B[H_2(f_{13})]$ and $B[H_3(f_{13})]$ reach the frequency threshold α. Therefore, the flow f_{13} is identified as a potentially active flow and inserted into the monitoring part.

3.4. Optimization 1: optimized flow insertion strategy of the monitoring part

In the basic version of the monitoring part, the insertion of a flow will fail if its mapped bucket has no empty cell. However, there may be empty cells in its two adjacent buckets. This provides an opportunity for the flow to be accommodated in the monitoring part. To take advantage of such opportunity, we optimize the flow insertion strategy of the monitoring part, by expanding the candidate insertion range of a flow to its two adjacent buckets. This will greatly improve the memory utilization of the monitoring part along with the detection accuracy of active flows, at the price of slight growth in its lookup overhead. Specifically, there are four cases for the flow insertion of the monitoring part.

Case 1: To insert the flow f, we traverse its directly mapped bucket $A[h(f)]$ and its two adjacent ones. If the flow f has been recorded in these buckets, we only need to increase its frequency by 1.

Case 2: The flow f is not recorded in its directly mapped bucket and its two adjacent ones, and there is an empty cell in these buckets, we insert it into this cell.

Case 3: The flow f is not recorded in its directly mapped bucket and its two adjacent ones, and there is no empty cell in these buckets.

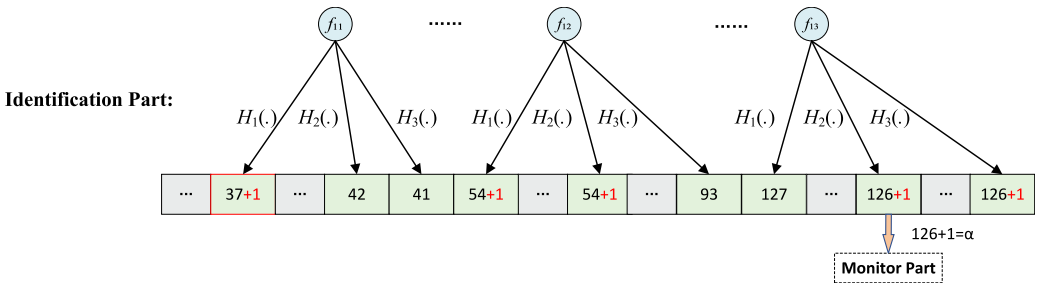

Fig. 3. The data structure and running example of the identification part.

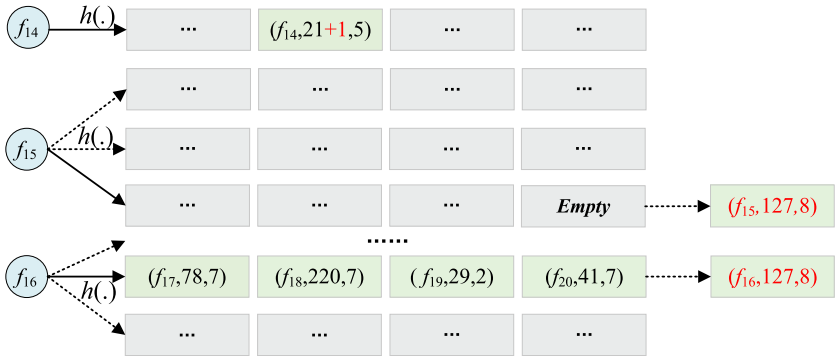

Fig. 4. The running example of the optimized insertion strategy for the monitoring part.

We select a flow with the smallest frequency in the bucket $A[h(f)]$ for probabilistic replacement.

A running example: Fig. 4 depicts a running example of the optimized insertion strategy for the monitoring part. Assume that the frequency threshold of an active flow in a time window $\alpha = 127$ and the sequence number of the current time window $W_{cur} = 8$. The insertion of f_{14}, f_{15}, and f_{16} can be described as follows.

(1) To insert f_{14}, it has been recorded in its directly mapped bucket. Therefore, we simply increment F_{cur} (the frequency of a flow in the current time window) of the corresponding cell by 1.

(2) To insert f_{15}, it is not recorded in its directly mapped bucket or its adjacent buckets. However, there is an empty cell in one of the adjacent buckets, so we insert $(f_{15}, 127, 8)$ into it.

(3) To insert f_{16}, it is not recorded in its directly mapped bucket $A[h(f_{16})]$ or its adjacent buckets, and there is no empty cell in these buckets. Therefore, we choose a flow in bucket $A[h(f_{16})]$ with the smallest frequency in the current time window and replace it with the probability p. If the replacement is successfully performed, the information $(f_{20}, 41, 7)$ will be changed into $(f_{16}, 127, 8)$.

3.5. Optimization 2: applying SIMD parallel technology in the monitoring part

SIMD (Single Instruction Multiple Data) technology achieves data parallelism by vectorization, which can be applied to accelerate sequential access operations [32]. As for each arrived packet, we need to perform sequential lookup on its directly mapped bucket and two adjacent ones in the monitoring part after applying the Optimization 1. Meanwhile, the lookup on each bucket needs to sequentially check multiple cells. These two factors lead to a multiplier effect on the lookup overheads of the monitoring part for packet traffic. Moreover, we still need to look up the monitoring part for an empty cell during the insertion of each potentially active flow. These result in serious problem of poor lookup performance of the monitoring part. Fortunately, this problem can be resolved by applying the SIMD technology to process multiple cells of the monitoring part in parallel with a single command.

We briefly describe the working principle of the monitoring part applying the SIMD technology as follows. As for an arrived packet within a flow f, we first map it to a bucket in the monitoring part by hashing. Specifically, we employ the `MurmurHash3_x64_128` primitive to generate a 128-bit hash value based on the flow identifier, which is then split into three parts corresponding to the directly mapped bucket and its two adjacent buckets. Next, we load the flow identifiers stored in all cells of these three buckets into a 256-bit SIMD register using the `_mm256_set_epi64x` instruction. Thereafter, we match the identifier of the flow f with the loaded identifiers in parallel using the `_mm256_cmpeq_epi64` instruction. If the flow f has been recorded in the monitoring part, we update its frequency in current time window. Otherwise, we insert the packet into the identification part. When a flow is identified as a potentially active one, we similarly apply the SIMD technology to accelerate lookup on its directly mapped bucket and two adjacent ones in the monitoring part for an empty cell. If an empty cell is found, we place the flow into it. In summary, it can significantly enhance the lookup performance and flow insertion efficiency of the monitoring part by applying the SIMD technology.

4. Theoretical analysis

This section provides theoretical analysis on the performance of AF-Detector. We first derive the error bounds of its flow frequency estimation in Section 4.1. Then, we infer its false positive and negative error rates in Section 4.2. Finally, we provide its space and time complexities in Section 4.3.

4.1. The error bounds of flow frequency estimation

Theorem 1. *Suppose the monitoring part consists of M buckets, each of which contains n cells. The monitoring part records F flows, where there are S flows whose frequencies in current time window are lower than the frequency threshold α. There are C potentially active flows arrived at the monitoring part. Let f and $\hat{f}$ respectively be the true frequency and*

estimated one of a flow in the monitoring part. We can infer the error bound of flow frequency estimation in the monitoring part as

$$E\left(\left|\hat{f}-f\right|\right) < \alpha \cdot \frac{e^{-r\alpha}}{M \cdot n \cdot (n!)^3} \tag{1}$$

Proof. We first analyze flow frequency estimation during the insertion process of packets in the monitoring part:

I: As for any flow g, if it has been recorded in the monitoring part, we directly increase its counter value by 1. In this case, the estimated frequency of the flow g is equal to its true frequency.

II: If the flow g is not recorded in the monitoring part and there is an empty cell in its directly mapped bucket, the flow g is inserted into this empty cell. In this case, the estimated frequency of the flow g is also equal to its true frequency.

III: If the flow g is replaced by a new potentially-active flow g', and the estimated frequency of the flow g is lower than its true frequency.

In summary, flow frequency estimation will only result in errors during flow replacement in the monitoring part. Flow replacement only occurs in the case of the following conditions: (1) Condition A: The flow g' is mapped to the same bucket as the flow g. (2) Condition B: There are no empty cell in its directly mapped bucket and its two adjacent buckets. (3) Condition C: The estimated frequency of the flow g is lower than the frequency threshold of active flows α and the flow g has the lowest frequency among all flows in its directly mapped bucket.(4) Condition D: The flow g is successfully replaced by the flow g'.

Since the above conditions are independent of each other, we can express the probability of flow replacement as

$$P = P(A \cap B \cap C \cap D) = P(A) \cdot P(B) \cdot P(C) \cdot P(D). \tag{2}$$

Suppose that each flow is randomly mapped into the monitoring part by hashing, we can consider the insertion of each flow as an independent random event. Consequently, we can infer that the probability of the condition A is

$$P(A) = \frac{1}{M}. \tag{3}$$

Suppose that each flow will be randomly inserted into any bucket, the number of flows recorded in each bucket approximately follows a Poisson distribution with parameter $\lambda = \frac{F}{M}$. Hence, the probability of the condition B as

$$P(B) = \left(\frac{e^{-\lambda}\lambda^n}{n!}\right)^3. \tag{4}$$

Because there are S flows with frequencies lower than the frequency threshold α, the estimated frequency of the flow g is lower than the frequency threshold α with the probability of $\frac{S}{F}$. Due to the full state of the directly mapped bucket of the flow g, its estimated frequency is the smallest in this bucket with the probability of $\frac{1}{n}$. Therefore, we can infer the probability of the condition C as

$$P(C) = \frac{S}{F \cdot n}. \tag{5}$$

According to our designed probabilistic replacement strategy, the probability of the condition D is

$$P(D) = e^{-r\hat{f}}. \tag{6}$$

Based on the above discussion, we can derive the probability that flow g is replaced by a potentially active flow g' in current time window as

$$P = P(A \cap B \cap C \cap D) = \frac{1}{M}\left(\frac{e^{-\lambda}\lambda^n}{n!}\right)^3 \frac{S}{F \cdot n} e^{-r\hat{f}}. \tag{7}$$

Each bucket receives $\frac{C}{M}$ flows. The flow g is replaced only when its estimated frequency goes below the frequency threshold α. Consequently, we can derive the error bound of flow frequency estimation in the monitoring part as

$$\begin{aligned} E\left(\left|\hat{f}-f\right|\right) &< \alpha \cdot \left(1-(1-P)^{\frac{C}{M}}\right) \\ &= \alpha \cdot \left(1-\left(1-\frac{1}{M}\left(\frac{e^{-\lambda}\lambda^n}{n!}\right)^3 \frac{S}{F \cdot n} e^{-\eta\hat{f}}\right)^{\frac{C}{M}}\right) \\ &\le \alpha \cdot \frac{e^{-r\alpha}}{M \cdot n \cdot (n!)^3} \end{aligned} \quad \square \tag{8}$$

Theorem 2. *Suppose the identification part consists of N counters and k independent hash functions. Let S be a sequence of network traffic with W packets. For a flow g in the identification part, let $\hat{f}$ be its estimated frequency, and let f be its true frequency. Given a small variable δ ($\delta > 0$), we can infer the error bound of flow frequency estimation in the identification part as*

$$P(\hat{f} - f > \delta \cdot W) \le \left(\frac{1}{\delta \cdot N}\right)^k. \tag{9}$$

Proof. As for any flow g, let X_j $(1 \le j \le k)$ be its mapped counter in the identification part by the jth hash function. We can formulate X_j as

$$X_j = f + \sum_{g_i \neq g} \mathbb{I}(h_j(g_i) = h_j(g)) \cdot f_{g_i}. \tag{10}$$

where g_i refers to a flow that is different from the flow g in the identification part with the frequency f_{g_i}; $h_j(\cdot)$ represents the jth hash function; $\mathbb{I}(\cdot)$ is a boolean function.

Since there are a total of W packets, other flows contain $W - f$ packets. Suppose that all flows are randomly mapped into N counters, we can obtain the expectation of the frequency estimation error of the flow g caused by other flows as

$$E[\sum_{g_i \neq g} \mathbb{I}(h_j(g_i) = h_j(g)) \cdot f_{g_i}] = \frac{W-f}{N}. \tag{11}$$

Since all hash functions are independent of each other, we can apply the Markov inequality to derive the error bound of flow frequency estimation at the jth mapped counter of flow g as

$$\begin{aligned} &P\left(\sum_{g_i \neq g} \mathbb{I}(h_j(g_i) = h_j(g)) \cdot f_{g_i} > \delta \cdot W\right) \\ &\le \frac{E\left[\sum_{g_i \neq g} \mathbb{I}(h_j(g_i) = h_j(g)) \cdot f_{g_i}\right]}{\delta \cdot W} \\ &= \frac{W-f}{\delta \cdot W \cdot N}. \end{aligned} \tag{12}$$

Since the identification part takes the minimal mapped counter of a flow as its estimated frequency, we can express the estimated frequency of the flow g as

$$\hat{f} = \min_{1 \le j \le k} X_j. \tag{13}$$

Consequently, we can derive the error bound of flow frequency estimation in the identification part as

$$\begin{aligned} &P(\hat{f} - f > \delta \cdot W) \\ &= P(\min_{1 \le j \le k} X_j - f > \delta \cdot W) \\ &= P\left(\bigcap_{j=1}^{k} \{X_j - f > \delta \cdot W\}\right) \\ &= \prod_{j=1}^{k} P\left(X_j - f > \delta \cdot W\right) \\ &\le \left(\frac{W-f}{\delta \cdot W \cdot N}\right)^k \\ &< \left(\frac{1}{\delta \cdot N}\right)^k. \end{aligned} \quad \square \tag{14}$$

4.2. False positive and negative error rates

Theorem 3.

Suppose there are multiple fixed-size time windows w_1, w_2, w_3, …, w_β. There are W packets arrived at the identification part composed of N counters and k independent hash functions in the time window w_1. Let f and $\hat{f}$ respectively be the true frequency of a flow whose frequency less than the frequency threshold α and its estimated frequency in the identification part. The monitoring part consists of M buckets, each of which contains n cells. The monitoring part records F_i flows in the ith time window, where there are S_i flows whose frequencies is lower than the frequency threshold α. Then, we can infer the false positive error rate of AF-Detector as

$$P_{FPR}^{total} \le \left(\frac{W}{N \cdot \alpha}\right)^k. \tag{15}$$

Proof. A flow is only misclassified as an active flow by AF-Detector when it is mistakenly identified as a potentially active flow in the identification part and its frequency exceeds the threshold α in each of the following $\beta - 1$ time windows. Therefore, we can express that the probability of a flow being misclassified as an active flow by AF-Detector as

$$P_{FPR}^{total} = P_{FPR} \cdot P_{active}. \tag{16}$$

where P_{FPR} represents the false positive error rate of the identification part and P_{active} denotes the probability that the flow maintains a frequency above the threshold α in each of the following $\beta - 1$ time windows.

As for any flow g, the occurrence of a false positive error in the identification part implies that its estimated frequency exceeds the threshold α, while its true frequency is lower than α. Therefore, we can obtain the false positive error rate as

$$P_{FPR} = P(\hat{f} \ge \alpha \cap f < \alpha). \tag{17}$$

According to the law of total probability, the false positive error rate can be expressed through the following decomposition

$$P_{FPR} = P(\hat{f} \ge \alpha) - P(\hat{f} \ge \alpha \cap f \ge \alpha). \tag{18}$$

Let X_j $(1 \le j \le k)$ be the mapped counter in the identification part by the jth hash function for flow g. Due to $f < \alpha$, we have

$$P(\hat{f} \ge \alpha \cap f \ge \alpha) = 0. \tag{19}$$

$$P_{FPR} = P(\hat{f} \ge \alpha) = P\left(\bigcap_{j=1}^{k}\{X_j \ge \alpha\}\right) = \prod_{j=1}^{k} P\left(X_j \ge \alpha\right). \tag{20}$$

Suppose that all flows are randomly mapped into N counters, we can conclude the expectation of each counter in the identification part as

$$E[X_j] = \frac{W}{N}. \tag{21}$$

Further, we can use Markov's inequality to infer that the probability of $X_j \ge \alpha$ $(1 \le j \le k)$ as

$$P\left(X_j \ge \alpha\right) \le \frac{E\left[X_j\right]}{\alpha} = \frac{W}{N \cdot \alpha}. \tag{22}$$

Consequently, we can deduce the false positive error rate in the identification part as

$$P_{FPR} = P(\hat{f} \ge \alpha) \le \left(\frac{W}{N \cdot \alpha}\right)^k. \tag{23}$$

Since the monitoring part records F_i flows in the ith time window, where there are S_i flows whose frequencies is lower than the frequency threshold α, we can derive the probability that the flow maintains a frequency above the threshold α for a consecutive sequence of β time windows as

$$P_{active} = \left(\frac{F_i - S_i}{F_i}\right)^{\beta-1}. \tag{24}$$

Consequently, we can deduce the false positive error rate of AF-Detector as

$$FPR^{total} = P_{FPR} \cdot \left(\frac{F_i - S_i}{F_i}\right)^{\beta-1} \le P_{FPR} \le \left(\frac{W}{N \cdot \alpha}\right)^k. \quad \square \tag{25}$$

Theorem 4. *With the same assumption as Theorem 3, we can infer the false negative error rate of AF-Detector as*

$$P_{FNR}^{total} < 1 - \left(1 - \frac{e^{-r\alpha}}{M \cdot n \cdot (n!)^3}\right)^{\beta}. \tag{26}$$

Proof. An active flow will not be detected by AF-Detector only when it is replaced by other potentially active flows. Specifically, the monitoring part will miss an active flow if it is replaced in any of the consecutive β time windows.

Suppose that each flow will be randomly inserted into any bucket, each bucket receives C_i/M flows. With the flow replacement probability in (7), we can derive the probability that the flow is stored in the monitoring part for a consecutive sequence of β time windows as

$$\begin{aligned} P_{FNR}^{total} &= 1 - \prod_{i=1}^{\beta}\left(1 - \frac{1}{M}\left(\frac{e^{-\lambda_i}\lambda_i^n}{n!}\right)^3 \frac{S_i}{F_i \cdot n} e^{-r\hat{f}_i}\right)^{\frac{C_i}{M}} \quad \square \\ &< 1 - \left(1 - \frac{e^{-r\alpha}}{M \cdot n \cdot (n!)^3}\right)^{\beta}. \end{aligned} \tag{27}$$

To demonstrate the expected values of the theoretical frequency estimation error, false positive rate, and false negative rate, we evaluate the corresponding formulas under representative parameter settings. These values are representative of real-world configurations: $N = 4096$, $n = 4$, $M = 2048$, $\alpha = 127$, $\beta = 4$, $r = 0.01$, $k = 3$, $\delta = 0.001$, $W = 5 \times 10^5$. By substituting these values into the respective formulas, we obtain the following results:

- The error bound of flow frequency estimation in the monitoring part is less than approximately 3.15×10^{-7};
- The error bound of flow frequency estimation in the identification part is less than approximately 1.4×10^{-2};
- The total false positive error rate of AF-Detector is less than approximately 7.1×10^{-3};
- The false negative error rate of AF-Detector is less than approximately 9.92×10^{-9}.

4.3. Space and time complexities

Space complexity: AF-Detector consists of a monitoring part and an identification part. The monitoring part contains $M \cdot n$ cells, where M is the number of its buckets and n is the number of cells for its each bucket. Meanwhile, the identification part contains N counters. Suppose the size of each cell in the monitoring part and each counter in the identification part respectively as S_1 and S_2, we can get the memory size of AF-Detector as $M \cdot n \cdot S_1 + N \cdot S_2$. In our AF-Detector, each cell in the monitoring part consists of three fields: *Flow ID*, F_{cur}, and W_{first}. *Flow ID* is usually manifested as a flow fingerprint typically with 32 bits. For F_{cur}, 12 bits are adequate to record the frequency of a flow in a time window with 100 K packets. W_{first} can be configured with 8 bits to record the sequence number of the current time window. In summary, S_1 is 52 bits. S_2 is suitable to be set as 7 bits, due to the frequency threshold of active flows $\alpha = 127$. According to the state-of-the-art method [23], N, M, and n are typically set to 2048, 2048, and 4, respectively. Consequently, we can calculate the memory size of AF-Detector approximately as 53 KB. In conclusion, AF-Detector is a lightweight method with low space complexity.

Time Complexity: For an arrived packet within a flow f, we first look up the flow f in its directly mapped bucket and two adjacent ones in the monitoring part by hashing. This lookup process involves $3n$

cells, where n is the number of cells in each bucket of the monitoring part. If the flow f has been recorded in the monitoring part, we only need to update its frequency in the current time window. This case is executed with the time complexity $O(3n)$. Otherwise, we need to map the flow into the identification part with its k hash functions and update the minimum of k mapped counters. If the updated counter does not reach the frequency threshold α, the flow is not yet a potentially active flow, and packet processing ends. This case performs with the time complexity $O(3n + k)$. If the flow is identified as potentially active, we insert the flow f into the monitoring part, which requires checking $3n$ cells. Hence, the execution of this case needs to take the time complexity $O(6n + k)$. In summary, AF-Detector processes each packet with the time complexity $O(6n + k)$ in the worst case. Since n and k are generally configured as very small values, AF-Detector is a lightweight method with very low time complexity.

5. Experiments

In this section, we first introduce experimental setup. Subsequently, we optimize the parameter settings of AF-Detector. Finally, we evaluate the performance of AF-Detector and compare it with prevalent active-flow detection methods.

5.1. Experimental setup

(1) Dataset

MAWI Dataset: The MAWI dataset [33] was collected from daily traces at the transit link of WIDE to the upstream ISP. We select 20M packets, which contains 6692 active flows for the time window size set as 100 K.

Campus Dataset: The campus dataset [34] consist of campus network traffic collected over 10 days in 2016. We select 20M packets, which contains 5899 active flows for the time window size set as 100 K.

(2) Implementation We implement the state-of-the-art methods including AF-Detector, Steady Sketch [1], MV Sketch [7], Waving Sketch [21], Burst Sketch [23], and Elastic Sketch [29], whose parameters are configured as follows.

Steady Sketch: We set the ratio of the memory usage of its Steady-Filter to its total memory size as 0.2. Meanwhile, its RollingSketch contains two arrays, where the number of slots is determined by given memory size.

MV Sketch: We set the number of its rows to 4, where the number of its columns is adjusted in accordance with the size of its provided memory.

Waving Sketch: We set the number of cells in its each bucket to 16, where the number of its buckets is determined by the size of its allocated memory.

Burst Sketch: We set the number of hash functions to 1 and the ratio of the memory usage of its Stage 1 to its total memory size as 0.5. Meanwhile, each bucket in its Stage 2 contains 4 cells.

Elastic Sketch: Its light part consists of an array of counters, while each bucket in its heavy part stores 7 flows and a negative vote counter. Additionally for the heavy part, we set its replacement threshold to 8, where the number of its storage buckets is determined by configured memory size.

AF-Detector: We set the number of counters in its identification part to 3072 and 4 cells for each bucket in its monitoring part, where the number of its buckets is determined by given memory size. AF-Detector.Opt is the version of AF-Detector that adopts optimization 1 and optimization 2. AF-Detector.Cbf is the version of AF-Detector that adopts a Counting Bloom Filter in its identification part. We set the frequency threshold of an active flow in a time window as 127, the number threshold of consecutively active time windows of an active flow as 4, and the time window size as 100 K. Subsequently, we run the program on a server with dual 6-core CPUs (24 threads, Intel Xeon Silver 4214R @2.4 GHz) and 32 GB DRAM memory, to evaluate their detection performance of active flows.

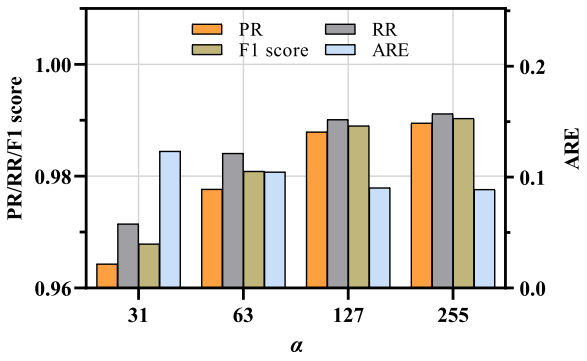

Fig. 5. Effects of α.

(3) Metrics

Precision Rate (PR): The ratio of the number of correctly reported active flows to the total number of reported active flows.

Recall Rate (RR): The ratio of the number of correctly reported active flows to the true number of active flows.

F1 score: $\frac{2 \cdot RR \cdot PR}{RR+PR}$. The F1 score is the harmonic average of the recall rate and precision, used to measure the accuracy of a method in monitoring active flows.

ARE (Average Relative Error): $\frac{1}{|\psi|}\sum_{n_i \in \psi}\frac{|\hat{f}_i - f_i|}{f_i}$, ψ is the total number of active flows reported by AF-Detector, f_i is the true frequency of the ith flow, and $\hat{f}_i$ is the estimated frequency of the i th flow.

Throughput: $\frac{N}{T}$, where N is the total number of packets, and T is the total measurement time. Throughput represents million insertions per second (MIPS).

5.2. Parameter settings

We conduct a comprehensive evaluation of the key parameters of AF-Detector, including the frequency threshold of an active flow in a time window α, the number threshold of consecutively active time windows of an active flow β, the number of hash functions in the identification part k, the replacement rate r, and the time window size S_{win}. In the experiments, we allocate 100 KB of memory. We conduct experiments on the MAWI dataset and evaluate the effects of parameters on performance of AF-Detector in terms of RR, PR, F1 score, and ARE.

Effects of α: Fig. 5 exhibits the performance metrics of AF-Detector by varying the value of α from 31 to 255, while keeping other parameters constant. As shown in Fig. 5, it is suitable to configure the frequency threshold α as 127. As the frequency threshold α increases, the PR, RR, and F1 score improve continuously and gradually stabilize after $\alpha = 127$. Meanwhile, the ARE decreases continuously. This is because as α increases, fewer flows meet the criteria for active flows. AF-Detector is able to track almost all active flows when $\alpha = 127$. Therefore, we default the frequency threshold α to 127.

Effects of β: Fig. 6 displays the performance metrics of AF-Detector by varying the value of β from 2 to 6, while keeping other parameters constant. As shown in Fig. 6, it is suitable to configure the number threshold of consecutively active time windows β as 4. With the increasing number threshold of consecutively active time windows β, the PR, RR, and F1 score continuously improve and gradually stabilize after $\beta = 4$. Meanwhile, the ARE continuously decreases and gradually stabilizes after $\beta = 4$. It is not advisable to set a high threshold for the number of consecutively active time windows, since our goal is to detect flows that may affect network performance. Therefore, we

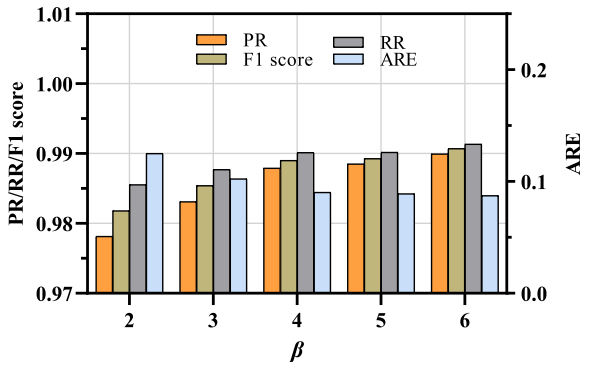

Fig. 6. Effects of β.

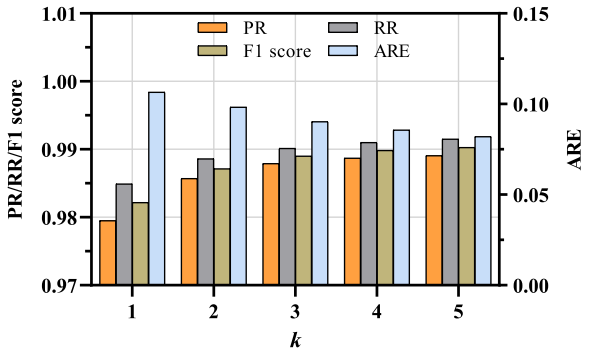

Fig. 7. Effects of k.

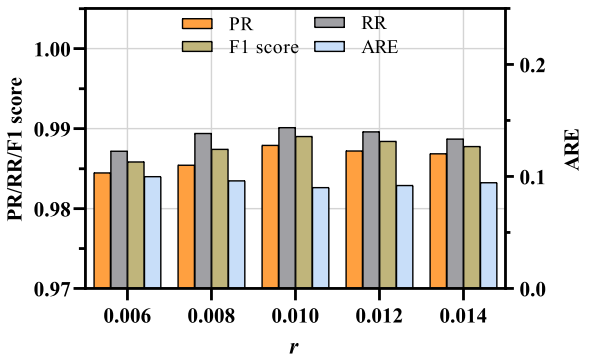

Fig. 8. Effects of r.

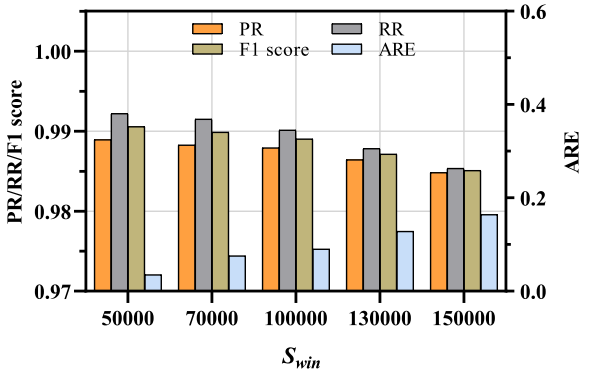

Fig. 9. Effects of S_{win}.

default the number threshold of consecutively active time windows β to 4.

Effects of k: Fig. 7 shows the performance metrics of AF-Detector by varying the value of k from 1 to 5, while keeping other parameters constant. As shown in Fig. 7, it is suitable to configure the number of hash functions in the identification part k as 3. With the increasing number of hash functions in the identification part, the PR, RR, and F1 score initially improve quickly and the improvement slows down after $k = 3$. Meanwhile, the ARE continuously decreases. Considering that more hash functions will increase packet processing time, we default the number of hash functions in the identification part to 3.

Effects of r: Fig. 8 depicts the performance metrics of AF-Detector by varying the value of r from 0.006 to 0.014, while keeping other parameters constant. As seen from Fig. 8, it is suitable to configure the replacement rate r as 0.01. With the increase of the replacement rate r, the PR, RR, and F1 score continuously improve at the beginning and gradually decrease after $r = 0.01$. Meanwhile, the ARE initially decreases and then gradually increases after $r = 0.01$. This is due to that a low replacement rate will lead potentially active flows to be easily replaced, while a high replacement rate can make it difficult to replace low-frequency flows. Therefore, we default the replacement rate r to 0.01.

Effects of S_{win}: Fig. 9 illustrates the performance metrics of AF-Detector by varying the value of Swin from 50 K to 150 K, while keeping other parameters constant. The experimental results in Fig. 9 demonstrate the optimal value of the time window size Swin as 100 K. As the time window size Swin increases, the PR, RR, and F1 score slightly decrease at the initial stage, but the decrease accelerates after Swin = 100 K, while the ARE continuously improves. As for a small size of time window, there will be fewer packets within a time window, which increases the difficulty for a flow to reach the frequency threshold of active flows. This results in fewer active flows which can be almost tracked by AF-Detector with higher detection accuracy. However, it will overlook some important flows which cannot meet the criteria of active flows. As for a big size of time window, there will be more packets within a time window, and more flows with their frequencies meeting the criteria of active flows. This results in a higher hash collision rate of the monitoring part, and the omission of some active flows, reducing detection accuracy. Meanwhile, a big size of time window also means a longer update cycle of AF-Detector, weakening the real-time nature of active-flow detection. In summary, we default the time window size Swin as 100 K.

5.3. Precision, recall and F1 score

Precision vs. Memory Size: Fig. 10 shows the precision of prevalent active-flow detection methods under different memory sizes. As depicted in Fig. 10, the precision of AF-Detector.Opt is always higher than that of other methods under the same memory size. Moreover, AF-Detector.Opt demonstrates superior precision in detecting active flows, even under low memory sizes. As seen from Fig. 10(a), AF-Detector.Opt and AF-Detector respectively achieve precision of 98.64% and 97.06%, surpassing that of Steady Sketch (89.29%), Burst Sketch (95.75%), Waving Sketch (85.63%), MV Sketch (79.17%), AF-Detector.Cbf (83.53%), and Elastic Sketch (82.73%) when the memory size is set to 60 KB. With the increasing memory size, the precision of all methods continuously improves. When the memory size is set to 100 KB, AF-Detector.Opt and AF-Detector respectively achieve precision of 99.59% and 98.79%, outperforming Steady Sketch (95.73%), Burst Sketch (98.21%), Waving Sketch (98.26%), MV Sketch (94.13%), AF-Detector.Cbf (86.45%), and Elastic Sketch (93.24%). In summary, both AF-Detector.Opt and AF-Detector achieve high accuracy in active-flow detection due to their separation of the tracking and identification functions of active flows, which reduces interference from inactive flows. Additionally, AF-Detector.Opt adopts a better insertion strategy, further enhancing its performance.

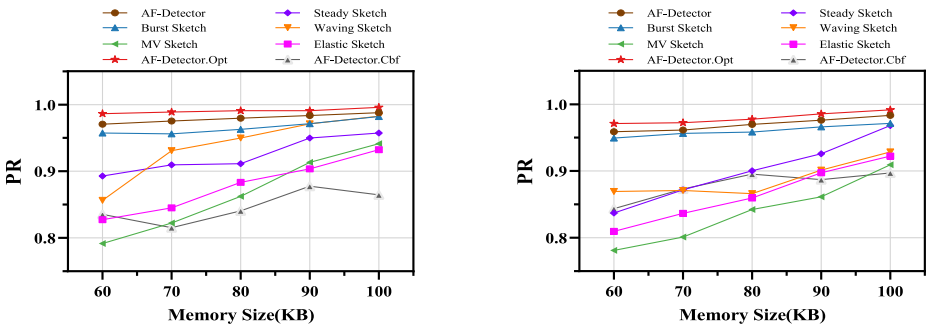

Fig. 10. Precision vs. memory size. .

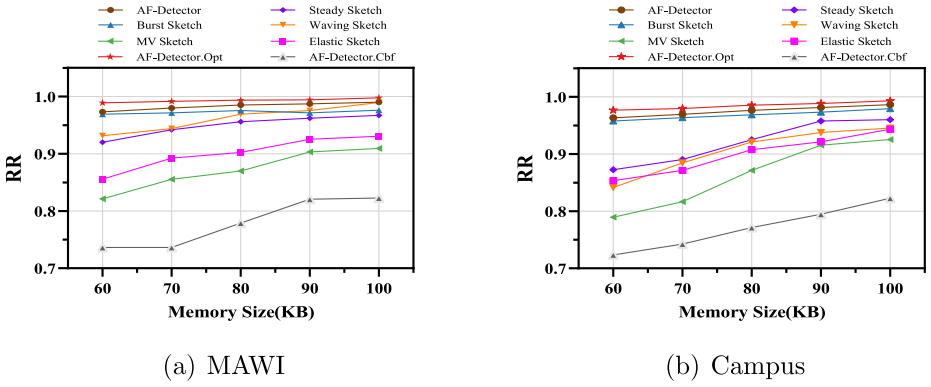

Fig. 11. Recall vs. memory size.

Recall vs. Memory Size: Fig. 11 shows the recall of prevalent active-flow detection methods under different memory sizes. As shown in Fig. 11, the recall rate of AF-Detector.Opt is higher than that of other methods under the same memory size. As seen from Fig. 11(a), AF-Detector.Opt and AF-Detector respectively achieve recall rates of 98.89% and 97.31%, surpassing that of Steady Sketch (92.00%), Burst Sketch (96.89%), Waving Sketch (93.14%), MV Sketch (82.12%), AF-Detector.Cbf (73.64%), and Elastic Sketch (85.57%) when the memory size is set to 60 KB. With the increasing memory size, the recall rates of all methods continuously improve. When the memory size is set to 100 KB, AF-Detector.Opt and AF-Detector respectively achieve recall rates of 99.78% and 99.01%, outperforming Steady Sketch (96.69%), Burst Sketch (97.59%), Waving Sketch (98.95%), MV Sketch (90.93%), AF-Detector.Cbf (82.26%), and Elastic Sketch (93.05%). In summary, both AF-Detector.Opt and AF-Detector consistently achieve high recall rates. This is due to their adoption of a probabilistic replacement strategy, which helps prevent the incorrect discarding of potentially active flows in the case of full buckets. Additionally, they employ an information cleanup mechanism to remove outdated or inactive flows at the end of each time window, allowing them to accurately detect active flows even under limited resource conditions. Moreover, AF-Detector.Opt adopts an optimized insertion strategy, providing a further boost to its performance.

F1 Score vs. Memory Size: Fig. 12 shows the F1 score of prevalent active-flow detection methods under different memory sizes. As shown in Fig. 12, AF-Detector.Opt consistently demonstrates superior F1 score compared to other methods under the same memory size. As seen from Fig. 12(a), AF-Detector.Opt and AF-Detector respectively achieve F1 scores of 0.9878 and 0.9718, surpassing that of Steady Sketch (0.9062), Burst Sketch (0.9632), Waving Sketch (0.8922), MV Sketch (0.8062), AF-Detector.Cbf (0.7876), and Elastic Sketch (0.8412) when the memory size is set to 60 KB. With the increasing memory size, the F1 score of all methods continuously improves. When the memory size is set to 100 KB, AF-Detector.Opt and AF-Detector respectively achieve F1 scores of 0.9969 and 0.9890, surpassing Steady Sketch (0.9621), Burst Sketch (0.9790), Waving Sketch (0.9860), MV Sketch (0.9251), AF-Detector.Cbf (0.8430), and Elastic Sketch (0.9315). In summary, both AF-Detector.Opt and AF-Detector consistently maintain higher F1 scores than other methods.

5.4. ARE

ARE vs. Memory Size: Fig. 13 shows the ARE of prevalent active-flow detection methods under different memory sizes. As shown in Fig. 13, it is evident that AF-Detector.Opt consistently maintains lower ARE than other methods under the same memory size. As seen from Fig. 13(a), AF-Detector.Opt and AF-Detector respectively achieve $\log_{10}$ ARE of −1.01 and −0.93, lower than that of Steady Sketch (−0.81), Burst Sketch (−0.90), Waving Sketch (−0.91), MV Sketch (−0.30), AF-Detector.Cbf (−0.01), and Elastic Sketch (0.34) when the memory size is set to 60 KB. With the increasing memory size, the ARE of all methods continuously decreases. When the memory size is set to 100 KB, AF-Detector.Opt and AF-Detector respectively achieve $\log_{10}$ ARE of −1.13 and −1.04, outperforming Steady Sketch (−0.98),

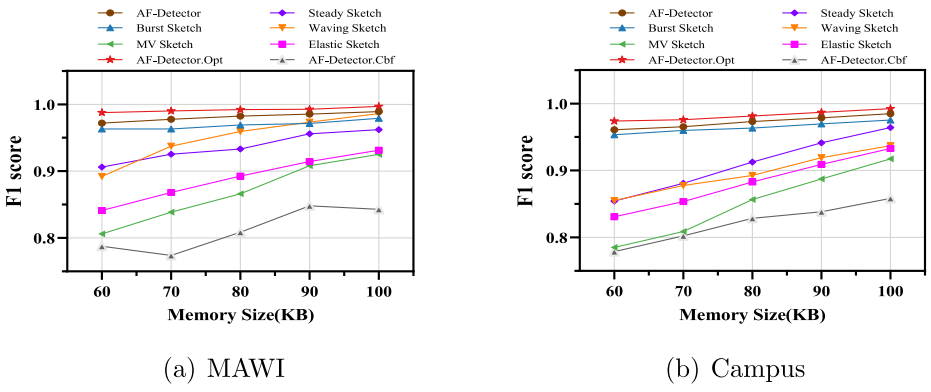

Fig. 12. F1 score vs. memory size.

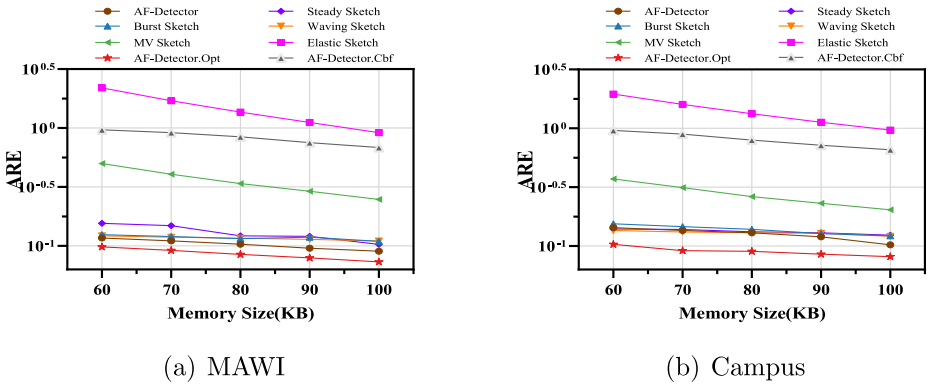

Fig. 13. ARE vs. memory size..

Burst Sketch (−0.96), Waving Sketch (−0.95), MV Sketch (−0.30), AF-Detector.Cbf (−0.16), and Elastic Sketch (−0.03). In summary, both AF-Detector.Opt and AF-Detector always achieve lower ARE compared to other methods. This is due to their prompt removal of inactive flows and periodic reporting of active flows, which improve the memory utilization efficiency of the monitoring part and reduce the hash conflicts between active flows.

5.5. Throughput

Throughput: Fig. 14 illustrates the throughput of prevalent active-flow detection methods on different datasets. As shown in Fig. 14, AF-Detector.Opt consistently maintains the highest throughput, no matter which dataset. For the MAWI dataset, the throughput of AF-Detector.Opt reaches 30.32 Mips, approximately 3.39, 1.15, 6.77, 1.72, 1.10, and 1.52 times that of Steady Sketch, Burst Sketch, Waving Sketch, MV Sketch, AF-Detector.Cbf, and Elastic Sketch, respectively. For the Campus dataset, the throughput of AF-Detector.Opt reaches 30.99 Mips, approximately 3.23, 1.21, 6.78, 1.82, 1.08, and 1.53 times that of Steady Sketch, Burst Sketch, Waving Sketch, MV Sketch, AF-Detector.Cbf, and Elastic Sketch, respectively. Compared to AF-Detector, AF-Detector.Opt increases throughput by 28.52% in average. This is due to that AF-Detector.Opt applies SIMD technology to enhance the lookup performance of the monitoring part and its flow insertion efficiency. In conclusion, AF-Detector.Opt can real-timely detect active flows in high-speed network environments.

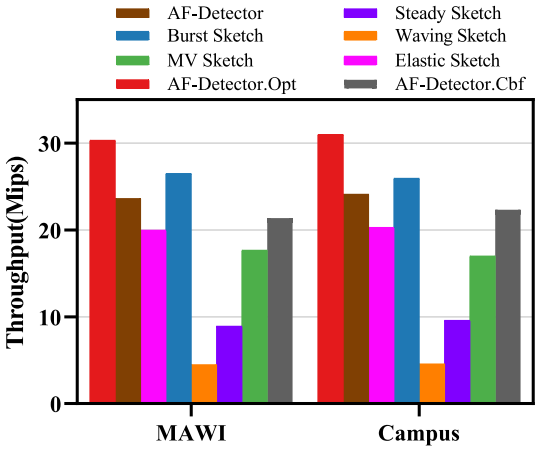

Fig. 14. The throughput of different active-flow detection methods.

6. Conclusion and discussion

Real-time detection of active flows plays a crucial role in network measurement. To address the issues with existing methods, we propose AF-Detector to detect active flows with high precision and low memory overhead. AF-Detector separates the tracking and identification functions of active flows, to precisely find out active flows and report their activity periods. Specifically, we estimate the frequencies of incoming flows and find out potentially active flows in the identification part. Subsequently, we track active flows in the monitoring part with a compact hash table. Meanwhile, we design a probabilistic replacement strategy to accommodate new potentially active flows for the case of full mapped buckets, and an information cleanup mechanism to promptly remove flows that are no longer active and outdated information, improving the accuracy of active-flow detection.

The experimental results show that AF-Detector achieves up to 99.59% precision and 99.78% recall rate for detecting active flows.

Even when the memory size is set to 60 KB, the precision and recall rate can achieve up to 98.64% and 98.89%, respectively. At the same time, the average relative error of AF-Detector is lower than that of the state-of-the-art methods. These results demonstrate that AF-Detector can ensure high precision, high recall rates, and low error even under low memory constraints, showing significant advantages over state-of-the-art methods. Moreover, AF-Detector achieves higher throughput of 30.99 Mips compared to the state-of-the-art methods. This indicates that AF-Detector is capable of real-time detection of active flows in high-speed network environments.

In our future work, we will further optimize the AF-Detector to enhance its robustness for detecting active flows in larger-scale and more complex network environments. For this end, we will apply our proposed techniques published in the paper [35] to enhance the performance of AF-Detector under highly bursty traffic conditions. Specifically, we will employ a stretchable identification part based on cyclic sketch chain, which dynamically adjusts the number of sketches in terms of real-time packet arrival rates, to ensure accurate filtering of inactive flows during network traffic fluctuations. Meanwhile, we will adopt a scalable monitoring part based on dynamic segmented hashing, which adaptively adjusts segments based on the number of potentially active flows, to fully record all active flows while maintaining high memory space utilization. Eventually, we plan to deploy AF-Detector in various network management systems to assist network administrators in improving network performance.

CRediT authorship contribution statement

Bing Xiong: Writing – review & editing, Methodology, Funding acquisition. **Yu Chang:** Writing – original draft, Validation, Data curation. **Yuhang Zhang:** Visualization, Software. **Jin Zhang:** Project administration, Investigation. **Baokang Zhao:** Funding acquisition, Formal analysis. **Keqin Li:** Supervision, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Baokang Zhao reports financial support was provided by National Natural Science Foundation of China. Bing Xiong reports financial support was provided by Hunan Provincial Natural Science Foundation of China. Bing Xiong reports financial support was provided by Scientific Research Foundation of Hunan Provincial Education Department. Bing Xiong has patent pending to Changsha University of Science and Technology. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (U22B2005, 61972412), Hunan Provincial Natural Science Foundation of China (2023JJ30053), Scientific Research Fund of Hunan Provincial Education Department (22A0232).

Data availability

The authors do not have permission to share data.

References

[1] Xiaodong Li, Zhuochen Fan, Haoyu Li, et al., SteadySketch: Finding steady flows in data streams, in: 31st IEEE/ACM International Symposium on Quality of Service, IWQoS, Orlando, USA, 2023, pp. 1–9.
[2] Hao Zheng, Chen Tian, Tong Yang, et al., Flymon: enabling on-the-fly task reconfiguration for network measurement, in: ACM Special Interest Group on Data Communication, SIGCOMM, Amsterdam, Netherlands, 2022, pp. 486–502.
[3] Peiqing Chen, Yuhan Wu, Tong Yang, et al., Precise error estimation for sketch-based flow measurement, in: 21st ACM Internet Measurement Conference, IMC, Online, 2021, pp. 113–121.
[4] Yuchen Xu, Wenfei Wu, Bohan Zhao, et al., MimoSketch: A framework to mine item frequency on multiple nodes with sketches, in: 29th ACM Special Interest Group on Knowledge Discovery and Data Mining, SIGKDD, Long Beach, USA, 2023, pp. 2838–2849.
[5] Lingtong Liu, Yulong Shen, Yibo Yan, et al., SF-sketch: A two-stage sketch for data streams, IEEE Trans. Parallel Distrib. Syst. 31 (10) (2020) 2263–2276.
[6] Ran Ben Basat, Gil Einziger, Roy Friedman, et al., Optimal elephant flow detection, in: IEEE Conference on Computer Communications, INFOCOM, Atlanta, USA, 2017, pp. 1–9.
[7] Lu Tang, Qun Huang, Patrick P.C. Lee, Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams, in: IEEE Conference on Computer Communications, INFOCOM, Paris, France, 2019, pp. 2026–2034.
[8] Qingjun Xiao, Yifei Li, Yeke Wu, Finding recently persistent flows in high-speed packet streams based on cuckoo filter, Comput. Netw. 237 (110097) (2023) 1–16.
[9] Jiahua Zhu, Xianliang Jiang, Yan Yu, et al., An efficient priority-driven congestion control algorithm for data center networks, China Commun. 17 (6) (2020) 37–50.
[10] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, et al., Maglev: A fast and reliable software network load balancer, in: Symposium on Network System Design and Implementation, NSDI, Santa Clara, USA, 2016, pp. 523–535.
[11] Xin Jin, Xiaozhou Li, Haoyu Zhang, et al., Netcache: Balancing key-value stores with fast in-network caching, in: 26th Symposium on Operating Systems Principles, SOSP, Shanghai, China, 2017, pp. 121–136.
[12] Lei Ying, Rayadurgam Srikant, Xiaohan Kang, The power of slightly more than one sample in randomized load balancing, Math. Oper. Res. 42 (3) (2017) 692–722.
[13] Wei Bai, Li Chen, Kai Chen, et al., Information-agnostic flow scheduling for commodity data centers, in: Symposium on Network System Design and Implementation, NSDI, Oakland, USA, 2015, pp. 455–468.
[14] Abdul Samad Bin Shibghatullah, Mitigating developed persistent threats (APTs) through machine learning-based intrusion detection systems: a comprehensive analysis, SHIFRA 2023, 17–25.
[15] L. Hussain, Fortifying AI against cyber threats advancing resilient systems to combat adversarial attacks, EDRAAK 2024, 26–31.
[16] S. Zhang, R. Shi, J. Zhao, A visualization system for multiple heterogeneous network security data and fusion analysis, KSII Trans. Internet Inf. Syst. (TIIS) 10 (6) (2016) 2801–2816.
[17] Anja Feldmann, Albert Greenberg, Carsten Lund, et al., Deriving traffic demands for operational IP networks: Methodology and experience, IEEE/ACM Trans. Netw. 9 (3) (2001) 265–279.
[18] W. Dou, X. Xu, S. Meng, et al., An energy-aware virtual machine scheduling method for service QoS enhancement in clouds over big data, Concurr. Comput.: Pr. Exp. 29 (14) (2017) e3909.
[19] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, et al., Programmable packet scheduling at line rate, in: ACM Special Interest Group on Data Communication, SIGCOMM, Florianopolis,Brazil, 2016, pp. 44–57.
[20] Yinda Zhang, Jinyang Li, Yutian Lei, et al., On-off sketch: A fast and accurate sketch on persistence, Proc. VLDB Endow. 14 (2) (2020) 128–140.
[21] Jizhou Li, Zikun Li, Yifei Xu, et al., Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams, in: 26th ACM Special Interest Group on Knowledge Discovery and Data Mining, SIGKDD, Long Beach, USA, 2020, pp. 1574–1584.
[22] Yanshu Wang, Dan Li, Jianping Wu, FastKeeper: A fast algorithm for identifying top-k real-time large flows, in: IEEE Global Communications Conference, GLOBECOM, Madrid, Spain, 2021, pp. 01–07.
[23] Zheng Zhong, Shen Yan, Zikun Li, et al., Burstsketch: Finding bursts in data streams, in: ACM Special Interest Group on Management of Data, SIGMOD, Online, 2021, pp. 2375–2383.
[24] Tong Yang, Haowei Zhang, Jinyang Li, et al., HeavyKeeper: an accurate algorithm for finding top-k elephant flows, IEEE/ACM Trans. Netw. 27 (5) (2019) 1845–1858.
[25] Tong Yang, Junzhi Gong, Haowei Zhang, et al., Heavyguardian: Separate and guard hot items in data streams, in: 24th ACM Special Interest Group on Knowledge Discovery and Data Mining, SIGKDD, London, United Kingdom, 2018, pp. 2584–2593.

[26] Graham Cormode, Shan Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, J. Algorithms 55 (1) (2005) 58–75.
[27] Moses Charikar, Kevin Chen, Martin Farach-Colton, Finding frequent items in data streams, in: International Colloquium on Automata, Languages, and Programming, ICALP, Malaga, Spain, 2002, pp. 693–703.
[28] Tong Yang, Yang Zhou, Hao Jin, et al., Pyramid sketch: A sketch framework for frequency estimation of data streams, VLDB Endow. 10 (11) (2017) 1442–1453.
[29] Tong Yang, Jie Jiang, Peng Liu, et al., Elastic sketch: Adaptive and fast network-wide measurements, in: ACM Special Interest Group on Data Communication, Budapest, Hungary, 2018, pp. 561–575.
[30] Anshumali Shrivastava, Arnd Christian Konig, Mikhail Bilenko, Time adaptive sketches (ada-sketches) for summarizing data streams, in: ACM Special Interest Group on Management of Data, SIGMOD, Florianópolis, Brazil, 2016, pp. 1417–1432.
[31] Burton H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[32] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, et al., SIMD-scan: ultra fast in-memory table scan using on-chip vector processing units, VLDB Endow. 2 (1) (2009) 385–394.
[33] MAWILab, Mawi working group traffic archive, http://mawi.wide.ad.jp/mawi/.
[34] Manmeet Singh, Maninder Singh, Sanmeet Kaur, 10 days DNS network traffic from April-May, 2016, Mendeley Data 2 (2019).
[35] Bing Xiong, Yong Qing Liu, Zhuo Qun Xia, et al., RobustSketch: Elastic method for elephant flow identification supporting network traffic jitters, J. Softw. 36 (2) (2025) 660–679.

Bing Xiong received the Ph.D. degree in Computer Science by master-doctorate program from Huazhong University of Science and Technology (HUST), China, in 2009, and the B.S. degree from Hubei Normal University, China, in 2004. He has been working in Changsha University of Science and Technology (CSUST), China, since 2010. He worked as a visiting scholar in the Department of Computer and Information Science, Temple University, USA, from 2018 to 2019. He is currently an associate professor in the School of Computer Science and Technology, CSUST, China. His main research interests include future network architecture, network traffic measurements, and artificial intelligence applications.

Yu Chang received the B.S. degree in Software Engineering from North China University of Water Resources and Electric Power, China, in 2023. He is currently pursuing the M.S. degree in the School of Computer Science and Technology, Changsha University of Science and Technology, China. His main research interests include network traffic measurements and software-defined networking.

Yuhang Zhang is currently an undergraduate majoring in computer science, and pursuing the B.S. degree in the School of Computer Science and Technology, Changsha University of Science and Technology, China. His main research interests include network traffic measurements and artificial intelligence applications.

Jin Zhang received the B.S. degree in communication engineering and the M.S. degree in computer application from Hunan University, Changsha, China, in 2002 and 2004, respectively, and the Ph.D. degree in biomedical engineering from Zhejiang University, Hangzhou, China, in 2007. He has been a Professor with Changsha University of Science and Technology since 2021. From 2008 to 2009, he worked as an Associate Professor with the Hunan University, Changsha, China. From 2009 to 2011, he worked as a Postdoctoral Fellow with the Beijing Normal University, Beijing, China. From 2012 to 2013, he worked as a Post-doctoral Fellow with the University of Chicago, Chicago, IL, USA. From 2014 to 2021, he has been a Professor with Hunan Normal University, Changsha, China. His research interests include computer network, software engineering, and artificial intelligence.

Baokang Zhao received the B.S., M.S., and Ph.D. degrees from National University of Defense Technology, all in computer science. He is currently an Associate Professor in the School of Computer Science, NUDT. His research interests include system design, protocols, algorithms, and security issues in computer networks.

Keqin Li received a B.S. degree in computer science from Tsinghua University in 1985 and a Ph.D. degree in computer science from the University of Houston in 1990. He is a SUNY Distinguished Professor at the State University of New York and a National Distinguished Professor at Hunan University (China). He has authored or co-authored more than 1130 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked #2) and career-long impact (ranked #4) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars (2023–2024) and is among the top 0.02% out of over 20 million scholars worldwide based on top-cited publications. He is listed in ScholarGPS Highly Ranked Scholars (2022–2024) and is among the top 0.002 scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He was a recipient of the 2022–2023 International Science and Technology Cooperation Award and the 2023 Xiaoxiang Friendship Award of Hunan Province, China. He is a Member of the SUNY Distinguished Academy. He is an AAAS Fellow, an IEEE Fellow, an AAIA Fellow, an ACIS Fellow, and an AIIA Fellow. He is a Member of the European Academy of Sciences and Arts. He is a Member of Academia Europaea (Academician of the Academy of Europe).