SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2025, Vol. 68, Iss. 3, 132103:1–132103:15
https://doi.org/10.1007/s11432-024-4244-3

FEA-Sketch: flow entries assisted sketch for heavy flow detection in software-defined networking

Xiaocan WU[1], He HUANG[1*], Yang DU[1], Yu-E SUN[2] & Yifan HAN[1]

[1]*School of Computer Science and Technology, Soochow University, Suzhou 215008, China*
[2]*School of Rail Transportation, Soochow University, Suzhou 215131, China*

Received 7 June 2024/Revised 31 August 2024/Accepted 4 December 2024/Published online 10 February 2025

Abstract Software-defined networking decouples the control plane from the data plane to enable centralized flow-level network management, while requiring periodically collecting traffic statistics from the data plane to enforce optimal management. As one of the most important traffic measurement tasks, heavy flow detection has received wide attention for its providing fundamental statistics in various practical applications. Existing studies have proposed sketch-based detection solutions to address the mismatch problem between massive traffic and limited high-speed memory resources for measurement in the data plane. However, they overlook the potential of integrating the flow table, where each entry simultaneously enforces forwarding rules for specific flows and records flow statistics into the sketch design, leading to redundant measurement between the flow table and sketch and being unable to utilize their statistics to jointly enhance estimation accuracy. We propose flow entries assisted sketch (FEA-Sketch) in this work, which employs a differentiated flow recording strategy to record flow statistics jointly using the flow table and sketch for memory-efficient and computationally efficient heavy flow detection. We also propose an optimization-based estimation algorithm to accurately recover per-flow sizes for the flows that only have aggregated statistics due to the sharing of entries in the table (or counters in the sketch). We extend the FEA-Sketch to the distributed measurement setting with a hop-based collaborative measurement strategy, which reduces the measurement workload on switches across the network by avoiding redundant measurements. The experimental results on real Internet traces show that the accuracy of heavy flow detection is improved up to 1.95 times, and the bias of flow size estimation is improved up to 2.99 times, demonstrating that integrating flow entries can significantly improve the performance of heavy flow detection.

Keywords software-defined networking, traffic measurement, heavy flow, flow table, sketch

Citation Wu X C, Huang H, Du Y, et al. FEA-Sketch: flow entries assisted sketch for heavy flow detection in software-defined networking. Sci China Inf Sci, 2025, 68(3): 132103, https://doi.org/10.1007/s11432-024-4244-3

1 Introduction

Software-defined networking (SDN) enables centralized network control by introducing a centralized controller that decouples the control plane from the data plane [1]. To facilitate optimal network management, the controller regards the packets carrying the same flow label (e.g., TCP/IP 5-tuples) as a flow, periodically collects flow traffic statistics from the data plane, determines the optimal forwarding paths for each flow, and finally deploys rules at switches for flow management. In the data plane, switches use ternary content addressable memory (TCAM) to implement flow tables, where each entry in the table includes a match field and an action field to enforce forwarding rules for flows, and has a counter field recording the number of packets matching that entry. However, due to the high cost and significant power consumption of TCAM, flow tables are generally limited to about 1000 entries [2], which means only a few flows can be allocated to individual entries (i.e., exact matching), while the other flows will be aggregated to the shared entries using wildcard matching, and even some flows will have no entries and need to await path assignments from the control plane. Therefore, the flow traffic measurement in the flow table is incomplete since the traffic statistics of many unmatched flows are not recorded, and the flows sharing the same wildcard entry only receive aggregated statistics. Because the controller requires the flow statistics of the last period for computing optimal forwarding rules, there is a critical need to develop efficient SDN traffic measurement mechanisms, particularly for flows not assigned individual entries in the flow table, to ensure fine-grained and robust flow management.

* Corresponding author (email: huangh@suda.edu.cn)

It is important to note that network traffic exhibits a highly skewed distribution: most flows are mouse flows, while a small number of heavy flows account for a significant portion of network traffic. For instance, in one minute of CAIDA data, there are 585680 flows, with the top 1% heavy flows constituting 89% of the total traffic. Given the significant impact of these heavy flows on network performance, current research often emphasizes the effective management of heavy flows within the flow table as a key aspect of optimal network management. For example, hybrid switching, which combines traditional and SDN switching, manages most network traffic (mainly contributed by heavy flows) using the flow table and forwards other unmatched flows via traditional default paths to enhance system scalability. Moreover, applications like network capacity planning [3] and anomaly detection [4] also depend on the accurate detection of heavy flows for effective decision-making. Therefore, we focus on heavy flow detection in this work, as it is one of the most important traffic measurement tasks in SDN.

Nevertheless, it is challenging to implement an efficient mechanism for heavy flow detection. Due to the limited flow table capacity and dynamic nature of networks, heavy flows might be categorized under wildcard matches in the flow table or forwarded by default paths, where they cannot be accurately measured by the flow table. Although traffic measurement can be deployed in the on-chip memory (SRAM) that supports high-speed memory access, the on-chip memory is typically just a few megabytes in size and must be shared with essential network functions like routing and firewall management, severely constraining the memory resource available to heavy flow detection. Another challenge is the redundant measurement of flows in a distributed measurement scenario. Because each flow follows a specified path through the data plane, it will be measured at every switch along its path if this does not exist an effective collaborative measurement strategy to distribute the measurement workload among the switches. Such redundant measurement can result in a significant waste of computational and storage resources.

Existing studies primarily focus on deploying compact data structures (i.e., sketches) in on-chip memory for heavy flow detection, which ensures low memory consumption while maintaining accuracy. Notable examples deployed on the switch-side include the solutions based on the counter-sharing strategy like CountSketch [5] and CountMin [6], which record statistics for all passing flows in a few counter arrays and employ a min-heap for tracking heavy flows. Furthermore, some studies explore collaborative measurement by deploying sketches on multiple switches and distributing measurement workload across them, such as CountMax [7] and Distributed Sketch [8]. While these sketch-focused solutions partially address certain challenges in developing mechanisms for heavy flow detection, they fail to fully exploit flow table resources to enhance their performance. Recall that flow table entries automatically update their counter fields when forwarding packets; integrating such statistics into the sketch design can simultaneously reduce the measurement overhead and enhance the performance of heavy flow detection. For instance, the sizes of exactly matched flows are precisely recorded in flow entries, allowing for direct querying from the flow table without additional recordings in the sketch. Meanwhile, by excluding the exact-matched flows from being recorded in the deployed sketch, the sketch records fewer flows (wildcard and unmatched flows) and yields fewer recording conflicts, thereby enhancing performance in detecting heavy flows.

However, it is non-trivial to jointly utilize the flow table and the sketch to improve the performance of heavy flow detection. Intuitively, we will assign flows different storage locations for recording, e.g., in the flow table, in the deployed sketch, or both, based on their flow table matching results. Exact-matched flows and unmatched flows will be recorded in the matched flow entries or the deployed sketch without overlap. Hence, the estimation process of these flows is straightforward. However, it is much more difficult to measure a wildcard-matched flow that shares its flow entry with several other matched flows. For such flows, we will additionally record them in the deployed sketch so that we can combine the statistics recorded in the flow table and sketch for accurate estimation. However, both the wildcard matching feature of the flow table and the counter-sharing nature of the sketch make it hard to fully exploit the recorded statistics to enhance the estimation performance for these flows.

In this paper, we introduce flow entries assisted sketch (FEA-Sketch), a novel heavy flow detection algorithm that offers high memory efficiency, high detection performance, and measurement workload balancing by jointly utilizing the flow table and on-chip memory resource. The key idea of FEA-Sketch is to collaboratively utilize the flow table and sketch to record flow statistics while reducing measurement redundancy for exact-matched and unmatched flows and enhancing estimation accuracy for wildcard-matched flows. We also utilize the network-wide collaborative measurement among switches to reduce the measurement workload of each switch. To achieve these goals, we first design a flow-level differentiated recording strategy to jointly utilize the flow table and sketch, which determines the storage of flow statistics recording (the flow table, the sketch, or both) for the incoming flow based on its matching

Table 1 Table of notations.

Notation	Description
S, $\langle f, e\rangle$	A traffic stream, a network packet with a flow label f and an element identifier e
V, v_i	All the switches in the data plane, switch i
f, n_f, $\hat{n}_f$	Flow f, the actual size of flow f, the estimated size of flow f
$\mathcal{T}$	Threshold of the heavy flow detection task
$\mathcal{E}_e$, $\mathcal{E}_w$	Exact-matching entries, wildcard-matching entries in the flow table
$\mathcal{C}$, $d_{\mathcal{C}}$, $m_{\mathcal{C}}$	A counter array $\mathcal{C}$ with $d_{\mathcal{C}}$ rows and $m_{\mathcal{C}}$ columns
r, $H_{\mathcal{C},r}(\cdot)$, $S_{\mathcal{C},r}(\cdot)$	Row indexing variable, the counter-mapping function, and the direction-mapping function in the i-th row
$\mathcal{F}$, $m_{\mathcal{F}}$	A bit array with $m_{\mathcal{F}}$ bits
e', $V_{\mathcal{F}}(\cdot)$, $H_{\mathcal{F}}(\cdot)$	Virtual element, the virtual element generator, and the bit-mapping function
$F_{\mathcal{E},i}$, $F_{\mathcal{C},i,j}$	All the flows matched to the wildcard-matching entry $\mathcal{E}_{w,i}$ and all the flows mapped to the counter $\mathcal{C}[i][j]$

results in the flow table. Then, we design a memory-efficient sketch with high detection performance by adopting two design features, which are utilizing a randomized bidirectional packet counting technique to randomly assign each flow a counting direction ($+1$ or -1) such that alleviates the noises from small flows, and employing a separated flow recording technique to evenly distribute arriving packets across sketch rows such that further reduce measurement overhead. After that, we design an optimization-based estimation algorithm to recover individual flow sizes accurately from the aggregated counts in both the flow table and the sketch, further refining the estimation accuracy for tracked heavy flows. Finally, FEA-Sketch adopts a hop-based collaborative measurement strategy to achieve network-wide heavy flow detection, which minimizes the measurement workload on switches while avoiding redundant packet measurements. The primary contributions of this paper are outlined as follows.

- In this work, we present a heavy flow detection solution FEA-sketch that simultaneously achieves high memory efficiency and high computational efficiency by integrating flow table resources into the sketch, leveraging the recorded traffic statistics in both the flow table and the sketch to achieve accurate heavy flow detection.
- FEA-Sketch is further extended to a distributed measurement setting with a hop-based collaborative measurement strategy, reducing the measurement workload on switches across the network by avoiding redundant measurements. Additionally, we have implemented an FEA-Sketch prototype on P4 bmv2 [9] software switches.
- We have conducted extensive experiments based on real Internet traces under Fat-Tree and Spine-Leaf network topologies. Experimental results demonstrate that FEA-Sketch surpasses existing solutions in detection accuracy and measurement throughput.

2 Background and motivation

2.1 Problem statement

In Table 1, we list the notations used in this paper. We consider a software-defined network consisting of a centralized controller and a set of switches $V = \{v_1, v_2, \ldots, v_n\}$. The controller adopts hybrid switching [10], integrating both traditional switching and SDN switching, to reduce its workload in determining paths for unmatched flows and address the scalability problem. Meanwhile, switches match the headers of incoming packets against the flow table/switching table and forward packets according to their matched rules, ensuring prompt reactions for the passing packets. To provide fine-grained and efficient forwarding control at the flow level, the controller periodically gathers traffic statistics from the data plane to update per-flow paths and deploy them to flow tables. Moreover, switches are embedded with sketch-based measurement modules to record per-flow sizes and detect heavy flows not specified in the flow table.

Specifically, we consider detecting heavy flows in the packet stream of N packets during a measurement epoch. Among the traffic stream $S = \{\langle f_1, e_1\rangle, \langle f_2, e_2\rangle, \langle f_3, e_3\rangle, \ldots, \langle f_N, e_N\rangle\}$, each forwarded packet carries a flow label f and an element identifier e in its header. A flow is defined as the collection of all packets within S that share the same flow label f, and the selection of flow label is consistent with the specified matching field of the flow table (e.g., TCP/IP 5-tuples). The flow table is too limited to specify paths for all passing flows; hence, only flows with flow sizes exceeding a certain threshold $\mathcal{T}$, identified

as a potential heavy flow, may be allocated a flow entry in the next cycle. This manuscript concentrates on detecting potential heavy flows and estimating their flow sizes for the next-epoch flow table update. The set F represents the collection of flows exceeding the threshold $\mathcal{T}$, where n_f is the actual size of a potential heavy flow f, and $\hat{n}_f$ is its estimated size.

2.2 Prior art and limitations

Sketches have been widely adopted to detect heavy flows with high memory efficiency and processing throughput. In this section, we first provide a comprehensive review of existing solutions for heavy flow detection on switch-side deployment and categorize them into two types, i.e., full-recording solutions and partial-recording solutions. And then, we further discuss the prior work on the collaborative measurement problem among multiple switches.

Full-recording solutions. The full-recording solutions usually employ synopsis data structures to record statistics of all incoming flows while keeping a min-heap for heavy flow tracking. These algorithms similarly adopt a counter array and utilize a counter-sharing strategy to share the counters among massive flows for approximate measurement. For instance, Count-Min [6] builds several counter arrays with the same size and then randomly maps a flow into a counter in each counter array to record the frequency of each flow. Counter Braids [11] and PR-Sketch [12] establish linear equations relating the size of each flow to the overall counting results of counters, subsequently estimating flow sizes by solving these equations. Pyramid Sketch [13] introduces a multi-layer counter array architecture and provides counters with appropriate scale for flows during the recording phase, enhancing estimation accuracy by reducing noise interference between large and small flows. Stingy Sketch [14] organizes the counting space into tree-structured counting nodes using a BC tree and incorporates prefetching techniques based on pipeline scheduling to reduce memory access overhead while maintaining estimation accuracy.

Partial-recording solutions. The partial-recording solutions utilize the replacing technique to keep heavy flows recorded in the sketch, sacrificing the statistics of small flows. For example, the Space Saving algorithm [15] maintains the stream-summary structure with a specific size for flow recording, and continuously evicts the smallest recorded flow to keep heavy flows. HeavyKeeper [16] uses an exponential decay strategy, assigning different decay probabilities based on the current frequency of the flow (larger flows have lower decay probabilities) to retain large flows' labels and statistics within limited memory space. Zhao et al. [17] found that existing compact data structures lack flexibility in configuring counters with different scales and proposed the DHS algorithm, which dynamically adjusts the counter size based on the current flow distribution. SwitchSketch [18] jointly utilizes the flexible bucket structure and the encoding-based switching scheme to support fine-grained memory allocation for varied-sized flows, achieving memory efficiency and flexibility.

Most of these prior solutions focus on the switch-side deployment, trying to resolve the mismatch between the limited on-chip memory of the deployed switch and the great volume of forwarded traffic. Although these solutions have achieved significant progress in implementing a memory efficient heavy flow detection mechanism, they overlook that deploying the detection module on a single switch is hard to accurately capture the global traffic information, and the deployed switch tends to become the bottleneck node affecting the networking performance. Hence, some studies utilize the fact that packets traverse multiple switches in the network, and propose methods for distributing measurement workload among these switches and collecting flow statistics to recover the measurement results.

Collaborative measurement. Some of the existing work attempts to solve the flow distribution issue by formulating optimization problems with targets (e.g., minimizing the maximum number of measured flows in any switch in [19] or maximizing the total number of monitored flows [20]). However, they overlook that the deployed sketch is a critical factor in influencing the performance of collaborative measurement, and fail to utilize the collected data from the data plane to improve the estimated flow statistics. Hence, CountMax [7] adopts key-value pairs to store heavy flows and collaboratively detects network-wide traffic by deploying sketches on edge switches to achieve low processing overhead. NWPS [21] extends priority sampling at switches for sampling, and develops merge algorithms for flow size estimating. Besides, Distributed Sketch [8] introduces the logical path sketch and the physical switch sketch as building blocks to fairly distribute the measurement workload among the switches in a network-wide range.

2.3 Our goal

Existing research on heavy flow detection mainly focuses on utilizing on-chip memory for sketch deployment, aiming to detect heavy flows and achieve precise flow size estimation. So far, these studies have not introduced flow entries into sketch design, leveraging the recorded traffic statistics in both flow entries and the sketch to fully enhance the performance of detecting heavy flows. Additionally, these studies often assign a fraction of switches for network-wide measurement, neglecting the workload imbalances between sketch-deployed and non-deployed switches. Consequently, this paper aims to develop a distributed mechanism for detecting heavy flows exceeding a predefined threshold, leveraging both flow entries and the sketch for performance enhancement while maintaining the collaboration of switches across the data plane to reduce measurement workloads and avoid redundant measurements. The specific goals of the designed mechanism are as follows.

- The proposed sketch should work with a differentiated flow-level statistics recording strategy to locate appropriate storage for passing flows in the recording phase. Also, considering the high line rate of massive network traffic, the implemented sketch must keep its per-packet processing overhead and on-chip memory consumption low enough to adapt to real-world operating conditions.
- The mechanism should leverage the recorded traffic statistics in both flow entries and the sketch to fully enhance the performance of the flow size estimation for heavy flows. Especially when the estimation phase utilizes the statistics in wildcard entries, these aggregated flow sizes must be effectively exploited.
- The proposed sketch should be strategically deployed to appropriately allocate the detection workload among switches while avoiding redundant measurements of flows.

3 FEA-Sketch design

3.1 Basic idea

Notice that, unlike existing solutions that primarily focus on implementing sketches for heavy flow detection, FEA-Sketch integrates the flow table into the heavy flow detection process, fully utilizing both the flow entries and on-chip memory resources to improve the detection performance. In sketch design, a differentiated flow-level statistics recording strategy is proposed to work with the implemented sketch, considering the possible matching cases of the flow table and locating the recording storage for passing flows based on their matching results. Subsequently, the random bidirectional packet counting technique and the separated flow recording technique are combined in the sketch design to eliminate hash conflicts of small flows and detect unmatched heavy flows with higher efficiency. The former randomly assigns each flow a counting direction ($+1$ or -1), while the latter evenly distributes arriving packets across sketch rows. In the estimation phase, the recorded traffic statistics in both flow entries and the sketch are utilized for comprehensive optimization to fully enhance the accuracy of flow size estimation for heavy flows. Specifically, the conflicted recording relations of the arrived flows in the counters are restored to formulate an optimization problem. By adopting the stochastic gradient descent method to iteratively update the estimated flow sizes, the accuracy of flow size estimation for tracked heavy flows is eventually refined.

3.2 Differentiated flow recording strategy

As illustrated in Figure 1, a differentiated flow recording strategy, driven by the flow table's matching results, is proposed to utilize both the flow entries and on-chip memory resources fully. The strategy comprehensively considers the possible matching conditions of the flow table, such as whether the flow is matched to a flow entry and if the matched flow entry supports wildcard matching, and differently locates the recording storage for passing flows based on their matching results. When a flow f is matched to the i-th one of exact-matching entries $\mathcal{E}_e$, the matched flow entry $\mathcal{E}_{e,i}$ can precisely update the flow size in its counter region, so the sketch part does not record the flow statistic again. When a flow is matched to the i-th one of wildcard-matching entries $\mathcal{E}_w$, the matched flow entry $\mathcal{E}_{w,i}$ is shared by several other matched flows for recording in its counter region, the sketch part should also record the flow statistic for a refined estimated result in the subsequent estimation phase. When a flow fails to match against the flow table, the sketch part has to record the flow statistic and determine if the flow is over-thresholded.

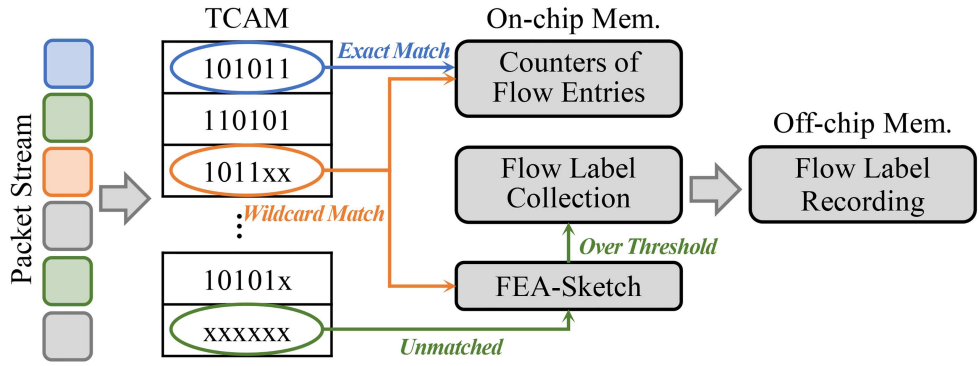

Figure 1 (Color online) Differentiated flow recording strategy driven by flow table matching results.

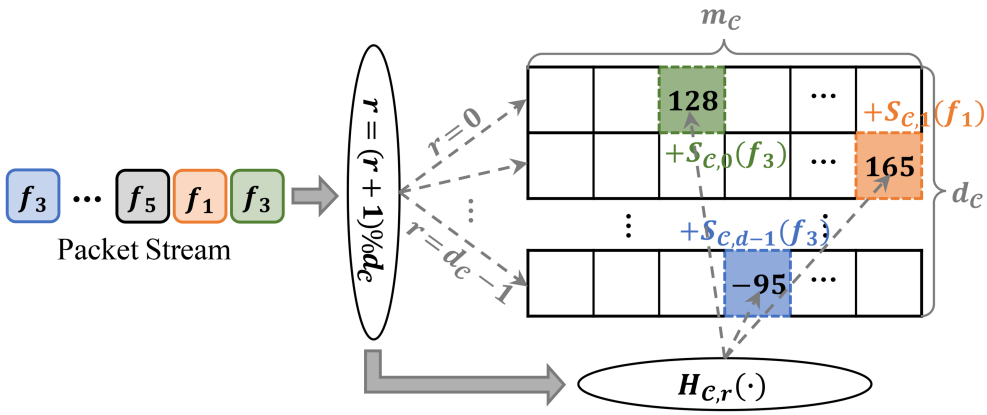

Figure 2 (Color online) Data structure of flow entries assisted sketch.

3.3 Recording phase of FEA-Sketch

With the target of recording flows that fail to exactly match against the flow table in the forwarding process, especially tracking potential heavy flows in the unmatched traffic, several techniques have been adopted in the sketch design. First, a counter-sharing strategy is utilized to share the counters among flows and achieve high memory efficiency. However, considering that most flows in network traffic have small flow sizes, the recorded statistics of a heavy flow are likely affected by other small flows in the sketch. Hence, the bidirectional packet counting technique makes flows have the same probability of increasing/decreasing their mapped counters. The recorded value in a counter mapped by several small flows is 0 in expectation, thereby eliminating hash conflicts of small flows and keeping statistics of heavy flows with higher accuracy. After that, the separated flow recording technique randomly selects one row to record the incoming packet, making a small flow divided into several sub-flows. Half of them will increase the counters, and the other half will decrease the counters, which further reduces the effects of small flows on heavy flow recording.

The data structure is as depicted in Figure 2, which contains a two-dimensional counter array $\mathcal{C}$ with $d_{\mathcal{C}}$ rows and $m_{\mathcal{C}}$ columns. At the very start of each measurement epoch, all the values of the counters are initialized to zero. The counter-sharing strategy is implemented by a map function $H_{\mathcal{C},i}(\cdot)$ that randomly maps each flow to an arbitrary counter in the i-th row of the counter array. The counter-mapping function takes the flow label f of an incoming packet as input and is computed by $H_{\mathcal{C},i}(f) = H'_i(f)\%m_{\mathcal{C}}$, where $H'_i(\cdot)$ is an independent hash function. It ensures that the mapped counter index of a flow is within the width of the counter array, i.e., $H_{\mathcal{C},i}(f) \in \{0, 1, \ldots, m_{\mathcal{C}} - 1\}$, and all packets with the flow label f will be mapped to the same counter $\mathcal{C}[i][H_{\mathcal{C},i}(f)]$. Meanwhile, each row of the counter array also maintains another map function $S_{\mathcal{C},i}(\cdot)$ to randomly decide the counting direction (positive or negative) of an arbitrary flow f in the i-th row. The direction-mapping function takes the flow label f of an incoming packet as input and is computed by $S_{\mathcal{C},i}(f) = (-1)^{H''_i(f)\%2}$, where $H''_i(\cdot)$ is an independent hash function. It ensures that the mapped counting direction of an arbitrary flow is uniformly distributed within the range $\{-1, 1\}$, and all packets with the flow label f will have the same counting direction in the mapped

Algorithm 1 Recording process of FEA-Sketch.

Input: an arriving packet $\langle f, e\rangle$;
$r \Leftarrow (r+1)\%d_{\mathcal{C}}$;
$\mathcal{C}[r][H_{\mathcal{C},r}(f)] \Leftarrow \mathcal{C}[r][H_{\mathcal{C},r}(f)] + S_{\mathcal{C},r}(f)$;
if $\mathcal{C}[r][H_{\mathcal{C},r}(f)] \times S_{\mathcal{C},r}(f) \geqslant \lceil \delta/d_{\mathcal{C}} \rceil$ **then**
    Transmit the flow label f to flow label collection module;
end if

Algorithm 2 Flow label collection process.

Input: a packet $\langle f, e\rangle$ transmitted from the recording phase;
$e' = V_{\mathcal{F}}(f)$;
if $e' > s$ **then**
    if $\mathcal{F}[H_{\mathcal{F}}(f, e')] = 0$ **then**
        $\mathcal{F}[H_{\mathcal{F}}(f, e')] \Leftarrow 1$;
        Download flow label f and virtual element identifier e' to off-chip memory;
    end if
end if

counter. After that, we use the row indexing parameter r to determine the selected row for the comping packet. Since network packets arrive randomly in the traffic, the probability of the selected row index for an arbitrary packet is uniformly distributed from 0 to $d_{\mathcal{C}} - 1$, making a flow evenly separated for recording throughout the counter array.

The recording procedure of an arbitrary arriving packet in the FEA-Sketch is shown in Algorithm 1. For an arbitrary arriving packet of the flow f, the sketch first updates the row indexing variable $r \Leftarrow (r+1)\%d_{\mathcal{C}}$ to determine the row index of the counter array for packet recording. Then, the flow label f of the packet is used as the input of the counter-mapping function $H_{\mathcal{C},r}(\cdot)$ to get the index of the counter that records the packet in r-th row, and the counter direction of the packet is determined according to the direction-mapping function $S_{\mathcal{C},r}(\cdot)$. Specifically, the mapped counter is updated as follows, $\mathcal{C}[r][H_{\mathcal{C},r}(f)] \Leftarrow \mathcal{C}[r][H_{\mathcal{C},r}(f)] + S_{\mathcal{C},r}(f)$. After the recording process for the arriving packet, we have to determine if the flow size of its related flow f is above the flow label collection threshold δ. If the updated value of the mapped counter is above the separated threshold $\delta/d_{\mathcal{C}}$, the packet should be passed to the flow label collection module, which decides if the flow label of the packet is supposed to be transmitted to off-chip part. Apparently, all of the operations in the recording process can be performed in constant time, and the time complexity of recording an arbitrary incoming packet in FEA-Sketch is $O(1)$. And, the space complexity of the recording process is $O(d_{\mathcal{C}} \times m_{\mathcal{C}})$.

3.4 Flow label collection

The flow label collection module consists of a bit array $\mathcal{F}$ with $m_{\mathcal{F}}$ bits, a virtual element generator $V_{\mathcal{F}}(\cdot)$, and a bit-mapping function $H_{\mathcal{F}}(\cdot)$. At the beginning of each measurement epoch, all the bits of the bit array are initialized to zero. The virtual element generator transforms an arriving packet into an arbitrary virtual element $e' \in \{0, 1, \ldots, d_v - 1\}$ with logarithmic probability. To this end, it first takes the flow label of an incoming packet as input and gets a d_v-bit hash value $v = H'(f \oplus a)\%2^{d_v}$, where $H'(\cdot)$ is an independent hash function, a is a random number, and $\oplus$ is the XOR operator. Then, it returns the virtual element identifier e' by computing the function $\rho(\cdot)$ with the input of the hash value v. $\rho(v)$ is a leftmost-one indexing function, which transforms the input value v to a binary form (i.e., $\langle v_0 v_1 \ldots v_{d_v-1}\rangle_2$) and returns the order of the leftmost 1 in the binary form of the value v. Also, the function starts from 0 and returns $d_v - 1$ when all bits of the value v are zeros. In this way, the virtual element generator, i.e., $V_{\mathcal{F}}(f) = \rho(H'(f \oplus a)\%2^{d_v})$, transforms each incoming packet to an arbitrary virtual element identifier $e' \in \{0, 1, \ldots, d_v - 1\}$ with a logarithmic probability $p_{e'} = 0.5^{\min(e'+1, d_v-1)}$. Meanwhile, the bit-mapping function $H_{\mathcal{F}}(\cdot)$ is maintained to randomly map each virtual element to a bit in the bit array $\mathcal{F}$, which represents if the generated virtual element identifier e' of a flow f has been transferred to the off-chip part. It takes the flow label f and the generated virtual element identifier e' of an incoming packet as input, and is computed by $H_{\mathcal{F}}(f, e') = H''(f \oplus e')\%m_{\mathcal{F}}$, where $H''(\cdot)$ is an independent hash function. Also, the mapped bit index of a virtual element is ensured within the length of the counter array, i.e., $H_{\mathcal{F}}(f, e') \in \{0, 1, \ldots, m_{\mathcal{F}} - 1\}$.

The flow label collection procedure of a packet transmitted from the recording phase is shown in Algorithm 2. For the transmitted packet $\langle f, e\rangle$, we first use its flow label f as input and compute with the virtual element generator $V_{\mathcal{F}}(\cdot)$ to obtain a virtual element identifier e' for the flow f. And then, we

Algorithm 3 Estimation process.

1: **for** $f \in F'$ **do**
2: **if** flow f matches the flow entry $\mathcal{E}_{w,i}$ **then**
3: $F_{\mathcal{E},i} \Leftarrow F_{\mathcal{E},i} \cup \{f\}$;
4: **end if**
5: **for** $i \in [0, d_{\mathcal{C}} - 1]$ **do**
6: $F_{\mathcal{C},i,H_{\mathcal{C},i}(f)} \Leftarrow F_{\mathcal{C},i,H_{\mathcal{C},i}(f)} \cup \{f\}$;
7: **end for**
8: $\hat{n}_f \Leftarrow d_{\mathcal{C}} \times \text{median}_{i=0}^{d_{\mathcal{C}}-1}\{\max\{1, \mathcal{C}[i][H_{\mathcal{C},i}(f)] \times S_{\mathcal{C},i}(f)\}\}$;
9: **end for**
10: **for** $t \in [1, \text{steps}]$ **do**
11: update the gradient based on equation (2);
12: utilize the adaptive learning rate algorithm for flow size updates;
13: correct estimated results according to the constraints of problem (1);
14: **end for**

prevent virtual element identifiers with values less than s from being collected. After that, we use the allowed virtual element e' and its flow label f as input and employ the bit-mapping function $H_{\mathcal{F}}(\cdot)$ to get the mapped index in the bit array $\mathcal{F}$. And, if the mapped bit has not been set to 1, which means the module has not collected the virtual element, the virtual element e' should be sent to the off-chip storage, and the mapped bit needs to be set to 1.

Since incoming packets are transformed with a logarithmic probability based on the virtual element generator, the statistic of a downloaded flow can be estimated based on its downloaded virtual elements (mainly the maximum virtual element identifier). Meanwhile, most statistics of small flows are eliminated by the random bidirectional packet counting strategy in the recording phase. Based on these facts, we can have that preventing virtual elements with small values from being collected, whether they belong to heavy flows or small flows, does not have a significant influence on the estimation performance of FEA-Sketch. Moreover, this strategy can greatly prioritize the flow label collection for heavy flows and reduce the communication cost of flow label collection. Besides, since all of the operations in the flow label collection process are performed in constant time, the time complexity of collecting a transmitted packet is $O(1)$. And, the space complexity of the flow label collection process is $O(m_{\mathcal{F}})$.

3.5 Estimation phase of FEA-Sketch

Since passing flows are recorded at different memory resources based on their matching results, the statistics are expected to be fully utilized to refine the measurement results in the estimation phase.

Algorithm 3 details how to leverage the recorded flow statistics in both the flow table and the deployed sketch to refine the measurement results with an iterative optimization process. Both the wildcard matching feature of the flow table and the counter-sharing characteristic of the sketch bring about plenty of conflict relations among the arrived flows; the algorithm fully utilizes these relations and converts the flow size estimation problem into an optimization problem that seeks to minimize the discrepancies between the estimated and actual recording values of counters in both flow entries and the sketch. The algorithm reconstructs the relationships among flows based on the collected flow labels, including the shared counters in wildcard entries and the hash conflicts in the sketch. Then, the optimization goal turns to reduce the total error between the aggregated sizes of flows affected by hash conflicts and the corresponding counter values in the sketch or flow entries. After that, the algorithm adopts the gradient descent method to iteratively enhance the accuracy of the flow size estimation for tracked heavy flows.

Before detailing the algorithm, some more notations are introduced. Let $F' = \{f_1, f_2, f_3, \dots\}$ denote the flow labels gathered by the flow label collection module in the current measurement epoch. Here, n_f and $\hat{n}_f$ represent the actual flow size and estimated flow size of an arbitrary flow f, respectively. All the collected virtual element identifiers of the flow are defined as $E_f = \{e'_0, e'_1, \dots\}$. According to the rules of the differentiated recording mechanism, the collected flows are categorized into the flows matching wildcard flow entries and the unmatched flows exceeding the collection threshold. Hence, let $F_{\mathcal{E},i}$ denote all the matched flows of a wildcard entry $\mathcal{E}_{w,i}$, and let $F_{\mathcal{C},i,j}$ denote flows mapped to counter $\mathcal{C}[i][j]$ in the sketch. As outlined in lines 2–6 of Algorithm 3, all the parameters $F_{\mathcal{E},i}$ and $F_{\mathcal{C},i,j}$ are updated by traversing the collected flows in the off-chip memory and utilizing the matching fields of the flow table (counter-mapping functions) of the sketch to recover conflict relations among the arrived flows. Subsequently, counters in wildcard flow entries can obtain estimated recording values by retrieving the recording process, i.e., $\sum_{f \in F_{\mathcal{E},i}} \hat{n}_f$, and counters in the sketch can similarly obtain estimated recording

values by computing $\sum_{f\in F_{\mathcal{C},i,j}} \hat{n}_f/d_{\mathcal{C}} \times S_{\mathcal{C},i}(f)$.

Given that the estimated recording value of an arbitrary counter in flow entries or the sketch corresponds to an actual recording value, it is evident that optimizing the biases between the estimated and actual recording values of counters can significantly enhance the performance of flow size estimation. Initially, we aim to utilize the recorded flow statistics to refine the measurement results in mechanism design. Based on the conversed optimization objective, the conceptual design goal can be specifically translated into an optimization problem, which targets minimizing the biases of estimated and actual recording values in both flow entries and the sketch. The problem is formally formulated in the following equation:

$$\begin{aligned} \min \quad & J\big(\{\hat{n}_f\}_{f\in F'}\big) = \sum_{i=0}^{|\mathcal{E}_w|-1}\left(\mathcal{E}_{w,i} - \sum_{f\in F_{\mathcal{E},i}} \hat{n}_f\right)^2 + \sum_{i=0}^{d_{\mathcal{C}}-1}\sum_{j=0}^{m_{\mathcal{C}}-1}\left(\mathcal{C}[i][j] - \sum_{f\in F_{\mathcal{C},i,j}} \frac{\hat{n}_f}{d_{\mathcal{C}}} \times S_{\mathcal{C},i}(f)\right)^2, \\ \text{s.t.} \quad & 2^{\max E_f - 1} < \hat{n}_f < 2^{\max E_f + 1}, \quad \forall f \in F'. \end{aligned} \tag{1}$$

To address this optimization problem, we employ the gradient descent method to iteratively update the estimated flow sizes of collected heavy flows and minimize the biases of estimated recording values. Specifically, the optimization objective $J(\{\hat{n}_f\}_{f\in F'})$ in (1) is adopted as the loss function, and its gradient with respect to an arbitrary flow f is derived as

$$\begin{aligned} \frac{\partial J(\{\hat{n}_f\}_{f\in F'})}{\partial \hat{n}_f} = {} & \mathbb{1}(f \in F_{\mathcal{E}}) \times 2\left(\sum_{f'\in F_{\mathcal{E},i}} \hat{n}_{f'} - \mathcal{E}_{w,i}\right) \\ & + \frac{2}{d_{\mathcal{C}}} \times \sum_{i=0}^{d_{\mathcal{C}}-1}\left(\sum_{f'\in F_{\mathcal{C},i,H_{\mathcal{C},i}(f)}} \frac{\hat{n}_{f'}}{d_{\mathcal{C}}} \times S_{\mathcal{C},i}(f') - \mathcal{C}[i][H_{\mathcal{C},i}(f)]\right) \times S_{\mathcal{C},i}(f). \end{aligned} \tag{2}$$

Subsequently, we use the stochastic gradient descent method to iteratively update the estimated sizes of all collected flows until the loss function converges. To avoid getting trapped in local optima as much as possible during the parameter learning process, we utilize the adaptive learning rate algorithm (Adam Optimizer) introduced in [22] for optimization during each iteration of stochastic gradient descent updates. The time complexity of the estimation phase is mainly affected by the number of collected flows and the iterations need to converge. Specifically, the time complexity of each iteration is $O(|F'|)$, where $|F'|$ is the number of collected flows. Since only heavy flows are collected for the estimation phase, which occupies a tiny fraction of the entire arriving flows, the time complexity of the gradient calculation is significantly reduced in each iteration. Another factor is the number of iterations for the loss function to converge. To accelerate the convergence process, the adaptive learning rate algorithm is adopted for optimization. And, the estimation phase generally converges after around 40 iterations in our experiments.

4 Network-wide heavy flow detection

4.1 Hop-based collaborative measurement strategy

We further explore the deployment of the designed mechanism on a network-wide scale to facilitate collaborative heavy flow detection among the switches in the data plane. A distributed heavy flow detection framework is developed, wherein each switch is embedded with the FEA-Sketch in its on-chip memory, and the employed hash functions have random initial seeds. During each measurement epoch, switches across the data plane record statistics of passing flows and detect potential heavy flows in the extensive network traffic. At the end of the epoch, each switch leverages the recorded flow statistics to refine the estimated results and identifies heavy flows for reporting. The centralized controller gathers the reported heavy flows from the switches and provides a comprehensive summary of the heavy flow activity across the network. Meanwhile, to achieve efficient packet-level load distribution among the switches, each switch should allocate computational resources efficiently to balance the measurement workload with other switches while avoiding redundant measurements of the flow.

Existing commodity switches, such as Barefoot Tofino and HP 5406zl, provide flexible packet processing by supporting multiple customizable match-action tables and enabling modifications to packet headers

Algorithm 4 Hop-based collaborative measurement strategy.

Input: arrived packet pkt $= \langle f, e\rangle$ on an arbitrary switch v_i;
Output: the sampling rate p_{f,v_i} of the arrived packet pkt on the switch v_i;
if switch v_i is ingress switch **then**
  add pkt.hop field in the header of the packet;
  pkt.hop $\Leftarrow 1$;
  $p_{f,v_i} \Leftarrow \frac{1}{2}$;
else if switch v_i is egress switch **then**
  pkt.hop $\Leftarrow$ pkt.hop $+ 1$;
  $p_{f,v_i} \Leftarrow \frac{1}{2^{\text{pkt.hop}+1}}$;
  remove pkt.hop field in the header of the packet;
else
  pkt.hop $\Leftarrow$ pkt.hop $+ 1$;
  $p_{f,v_i} \Leftarrow \frac{1}{2^{\text{pkt.hop}}}$;
end if
return p_{f,v_i};

through domain-specific languages. Given that custom packet header functionalities are widely employed in studies like NetCache [23] and DC.p4 [24], our designed framework expands a custom field in incoming packets to record the traversed hop count of each packet. Then, we adopt a packet sampling strategy based on the recorded hop count to efficiently distribute the measurement workloads across switches in the data plane. Algorithm 4 details the calculation of the sampling probability for an arriving packet pkt to be recorded in the sketch part of an arbitrary switch v_i. For a packet pkt $= \langle f, e\rangle$ arriving at switch v_i, the sampling strategy first determines whether switch v_i is the ingress switch. If it is, the packet's sampling probability is set to $\frac{1}{2}$, and a custom field is adjusted to 1 and added in the header of the incoming packet; otherwise, the sampling strategy has to determine whether switch v_i is the egress switch. If it is, the packet's sampling probability is set to $\frac{1}{2^{\text{pkt.hop}+1}}$, and the custom field is removed in its header; otherwise, the packet is processed at the non-edge switch, and the packet's sampling probability is set to $\frac{1}{2^{\text{pkt.hop}}}$. Without loss of generality, suppose that flow f with the flow size n_f is forwarded along the path $P_f = \{v_1, v_2, \dots\}$ in the data plane. The i-th switch in the path P_f processes the traffic of flow f with the proportion of p_{f,v_i}. Apparently, $\sum_{v_i \in P_f} p_{f,v_i} = 1$. Thus, this strategy can achieve complete flow coverage for flows along their forwarding paths.

4.2 Flow statistics collection in data plane

In the collaborative detection framework, switches combine the packet sampling strategy with the deployed heavy flow detection mechanism to achieve network-wide detection and reduce their measurement workloads. An arriving packet at an arbitrary switch is first processed at the hybrid switching module and may get a recording in the matched flow entry ($\mathcal{E}_{e,i}$ or $\mathcal{E}_{w,i}$). Then, suppose the packet needs to be recorded in the sketch according to the rules of the differentiated recording strategy. In that case, the sampling probability p_f is computed based on the hop-based collaborative measurement strategy, and a random value with a range of $[0, 1)$ is generated to decide if the packet will be recorded in the sketch.

Based on the sampling results of the collaborative measurement strategy, FEA-Sketch receives sampled packets for statistic recording, whose processing details have remained the same in its adaptation to the collaborative detection scenario. However, since packets of an arbitrary flow are sampled with probability p_f at switch v_i, there is a modification about the estimated flow size $\hat{n}_f$ in the flow size estimation part. The original optimization formula (1) and the gradient formula (2) undergo the following changes:

$$\begin{aligned} \min \quad & J\big(\{\hat{n}_f\}_{f\in F'}\big) = \sum_{i=0}^{|\mathcal{E}_w|-1}\left(\mathcal{E}_{w,i} - \sum_{f\in F_{\mathcal{E},i}} \hat{n}_f\right)^2 + \sum_{i=0}^{d_{\mathcal{C}}-1}\sum_{j=0}^{m_{\mathcal{C}}-1}\left(\mathcal{C}[i][j] - \sum_{f\in F_{\mathcal{C},i,j}} \frac{\hat{n}_f \times p_f}{d_{\mathcal{C}}} \times S_{\mathcal{C},i}(f)\right)^2, \\ \text{s.t.} \quad & 2^{\max E_f - 1} < \hat{n}_f \times p_f < 2^{\max E_f + 1}, \quad \forall f \in F', \end{aligned} \tag{3}$$

$$\begin{aligned} \frac{\partial J(\{\hat{n}_f\}_{f\in F'})}{\partial \hat{n}_f} = {} & \mathbb{1}(f \in F_{\mathcal{E}}) \times 2\left(\sum_{f'\in F_{\mathcal{E},i}} \hat{n}_{f'} - \mathcal{E}_{w,i}\right) \\ & + \frac{2\times p_f}{d_{\mathcal{C}}} \times \sum_{i=0}^{d_{\mathcal{C}}-1}\left(\sum_{f'\in F_{\mathcal{C},i,H_{\mathcal{C},i}(f)}} \frac{\hat{n}_{f'}\times p_{f'}}{d_{\mathcal{C}}} \times S_{\mathcal{C},i}(f') - \mathcal{C}[i][H_{\mathcal{C},i}(f)]\right) \times S_{\mathcal{C},i}(f). \end{aligned} \tag{4}$$

Subsequently, we still use the stochastic gradient descent method to iteratively update the estimated sizes of all collected flows until the loss function converges.

4.3 Flow statistics collection in control plane

After completing the recording phase of the heavy flow detection and getting the optimized results with the gradient descent method in the estimation phase, each switch v_i checks the flow sizes of the collected flows and reports the statistics of the flows, whose flow sizes exceed the threshold $\mathcal{T}$, i.e., $F'_{>\mathcal{T},v_i}$, to the centralized controller. In the control plane, the controller has gathered all the detected heavy flows whose flow sizes exceed the threshold $\mathcal{T}$, i.e., $F'_{>\mathcal{T},v_i v_i \in V}$. Given the forwarding path $P_f = v_1, v_2, \ldots$ of an arbitrary flow f, the controller can integrate the reported flow sizes from multiple switches and get the revised flow size as follows:

$$\hat{n}_f = \frac{\sum_{v_i \in P_f} \mathbb{1}(f \in F_{>\mathcal{T},v_i}) \times \hat{n}_{f,v_i} \times p_{f,v_i}}{\sum_{v_i \in P_f} \mathbb{1}(f \in F_{>\mathcal{T},v_i}) \times p_{f,v_i}}.$$

5 Performance evaluation

5.1 Metrics and benchmarks

Sketches are deployed on switches to facilitate collaborative heavy flow detection across the network, tracking potential heavy flows with limited computation resources and uploading them to the control plane for aggregation. Hence, we mainly focus on evaluating the performance of the proposed mechanism in terms of detection accuracy, estimation accuracy, and computing overhead. We adopt the following performance metrics.

• **Recall rate in heavy flow detection.** $F_{>\mathcal{T}}$ represents all the flows in the network traffic with their actual flow sizes greater than the threshold $\mathcal{T}$, and $F'_{>\mathcal{T}}$ represents all the tracked flows whose flow sizes exceed the threshold $\mathcal{T}$ by a detection mechanism. The recall rate is defined as $\frac{|F_{>\mathcal{T}} \cap F'_{>\mathcal{T}}|}{|F_{>\mathcal{T}}|}$, indicating the performance of the solution to identify relevant flows.

• **Average relative error (ARE) in estimation tasks for heavy flows.** For each flow f whose actual flow size exceeds the threshold $\mathcal{T}$, we calculate its relative error as $\frac{|n_f - \hat{n}_f|}{n_f}$, where n_f and $\hat{n}_f$ represent the actual and estimated flow sizes of the flow f. The average relative error is defined as $\sum_{f \in F_{>\mathcal{T}}} \frac{|n_f - \hat{n}_f|}{n_f} / |F_{>\mathcal{T}}|$ to reflect the performance of the solution for flow size estimation.

• **Average running time of packet processing in network.** Given a set of n packets, we run different sketches for network-wide heavy flow detection and compare the average running times of the packets, assessing the computational efficiency of our mechanism.

To verify the performance of our mechanism and highlight its advantages in incurring low measurement workload with accuracy guarantees, we compare FEA-Sketch with other existing solutions like CountMax [7], Distributed Sketch [8] (abbreviated as DS in the following content), and PR-Sketch [12]. For FEA-Sketch, we configure the counter array $\mathcal{C}$ to have $d_{\mathcal{C}} = 2$ rows and set the flow label collection threshold δ to 128. For CountMax, DS, and PR-Sketch, we follow the recommendation of parameters in their original papers. In the configuration of CountMax and DS, we set the number of rows to 2. As for PR-Sketch, we set the number of hash functions in both the filter part and the count part to 1.

5.2 Dataset and settings

The experiment employs two prominent network topologies, i.e., Fat-Tree [25] and Spine-Leaf [26], as running examples. The Fat-Tree topology includes 4 core, 8 aggregation, and 8 edge switches. The Spine-Leaf topology consists of 6 spine switches and 12 leaf switches, with each spine switch linked to every leaf switch. We use 5 min of real-world network traffic traces downloaded from CAIDA [27] as the experimental dataset, consisting of 1652015 flows and 160932385 packets. To simulate the forwarded flows on switches, we assign edge switches in the running example that have the same probability of being the source or destination of a flow, and adopt the OSPF algorithm to select the shortest path for the flows. For load balancing, we use 50 min of CAIDA network traffic traces as the dataset and compute with the ARIMA algorithm [28] to forecast the subsequent 5 min of traffic. Then, we identify the maximum 8000

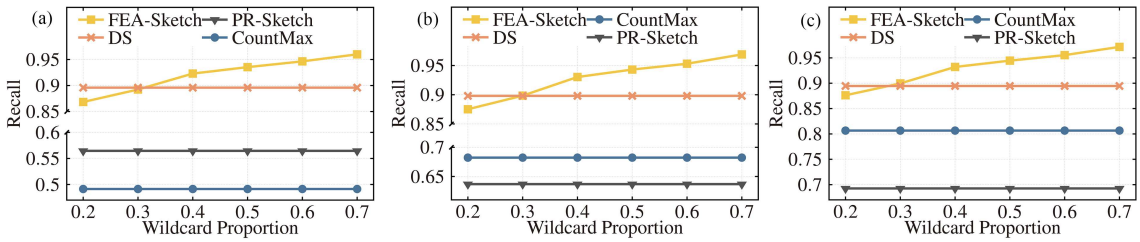

Figure 3 (Color online) Detection performance of solutions for flows whose statistics are above 500 under Fat-Tree topology. (a) $M = 30$ kB; (b) $M = 50$ kB; (c) $M = 70$ kB.

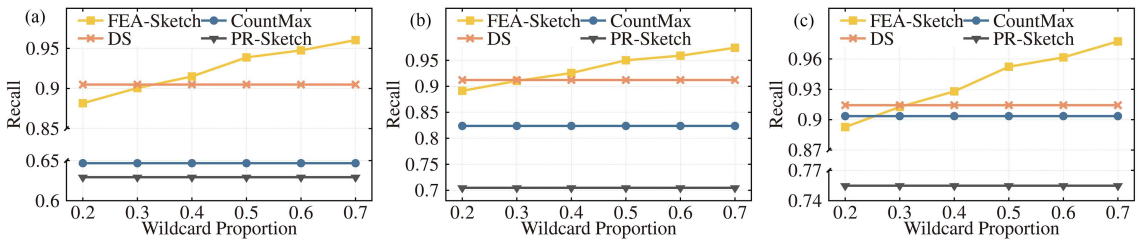

Figure 4 (Color online) Detection performance of solutions for flows whose statistics are above 500 under Spine-Leaf topology. (a) $M = 30$ kB; (b) $M = 50$ kB; (c) $M = 70$ kB.

flows in the predicted traffic to simulate the prediction of the most likely heavy flows in the upcoming measurement period, and reroute these flows by greedily choosing the path with minimum congestion (or maximum link load) in all the possible forwarding paths of a rerouted flow. After that, flow entries are established along the forwarding paths, and the ORTC algorithm [29] is adopted to compress the rules in a flow table, whose size is around 1500 after the compression.

We also implement a prototype based on P4 virtual switches and evaluate the proposed mechanism. Specifically, both Fat-Tree 4 topology and Spine-Leaf 6 topology are constructed in performance evaluation. We use the virtual machines as hosts, and each host is linked to a virtual switch at the edge. We deploy the p4-implemented FEA-Sketch, CountMax, DS, and PR-Sketch on virtual switches, and run these software switches in bmv2 [9]. Moreover, communications between the control plane and the data plane (e.g., flow table update, sketch query, and etc.) are achieved via gRPC connections. The prototype, including the p4 implementations of FEA-Sketch, CountMax, DS, and PR-Sketch, has been published at Github [30].

5.3 Evaluation on heavy flow detection

Impact of wildcard entry proportion on accuracy. In Figures 3(a) and 4(a), we fix the available memory usage of the adopted sketches and vary the wildcard entry proportion to show the recall rates of their detected heavy flows under Fat-Tree topology and Spine-Leaf topology. From the experimental results, we can notice that FEA-Sketch consistently maintains its recall rate over 85%, and the recall rate of its heavy flow detection increases with the growing proportion of wildcard entries in the flow table. Especially with 0.7 wildcard entry proportion, the recall rate of FEA-Sketch is up to 95%, 8%, and 70% higher than CountMax, DS, and PR-Sketch in the Fat-Tree topology, and 48%, 7%, and 52% higher in the Spine-Leaf topology. These results prove the effectiveness of incorporating the flow table into heavy flow detection, which can significantly enhance the accuracy of detection results.

Impact of memory size M on accuracy. In Figures 3 and 4, we vary the available memory usage of the adopted sketches to show the recall rates of their detected heavy flows under different topologies. The experimental results show that FEA-Sketch significantly excels under limited memory consumption, and the recall rate increases with the growing memory usage. Specifically, with 30 kB of memory space for sketches, the recall rate of FEA-Sketch is up to 95%, 7%, and 70% higher than CountMax, DS, and PR-Sketch in the Fat-Tree topology, and 48%, 5%, and 52% higher in the Spine-Leaf topology. These findings confirm that FEA-Sketch sufficiently utilizes the flow entries and has a better memory efficiency than existing solutions.

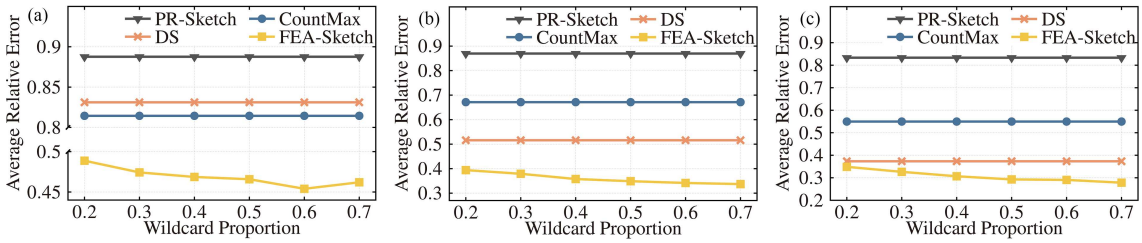

Figure 5 (Color online) Estimation performance of solutions for flows whose statistics are above 500 under Fat-Tree topology. (a) $M = 30$ kB; (b) $M = 50$ kB; (c) $M = 70$ kB.

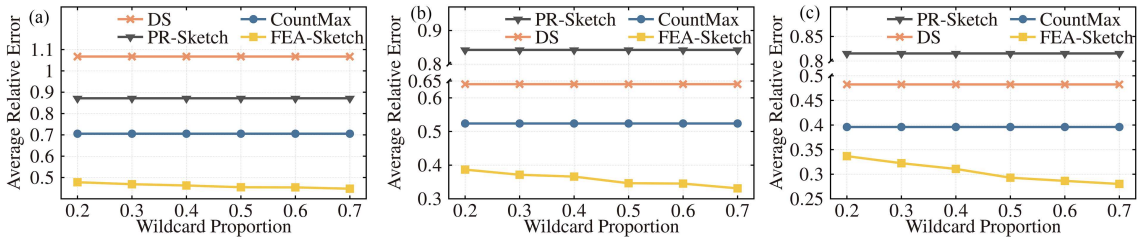

Figure 6 (Color online) Estimation performance of solutions for flows whose statistics are above 500 under Spine-Leaf topology. (a) $M = 30$ kB; (b) $M = 50$ kB; (c) $M = 70$ kB.

Table 2 Estimation performance when the wildcard entry proportion is set to 0.7 and the detection threshold $\mathcal{T}$ is set to 500.

	Algorithm	Fat-Tree topology						Spine-Leaf topology					
		Overall	$\mathcal{T}$–10^3	10^3–5000	5000–10^4	10^4–10^5	10^5–	Overall	$\mathcal{T}$–10^3	10^3–5000	5000–10^4	10^4–10^5	10^5–
30 kB	PR-Sketch	0.888	0.963	0.848	0.832	0.823	0.795	0.872	0.906	0.860	0.820	0.823	0.781
	DS	0.831	1.446	0.551	0.225	0.132	0.055	1.067	1.882	0.693	0.294	0.132	0.072
	CountMax	0.814	0.956	0.843	0.479	0.189	0.030	0.705	0.909	0.690	0.301	0.122	0.018
	FEA-Sketch	0.462	0.707	0.357	0.215	0.139	0.125	0.447	0.663	0.362	0.211	0.137	0.115
50 kB	PR-Sketch	0.870	0.935	0.834	0.824	0.823	0.785	0.842	0.870	0.832	0.806	0.809	0.782
	DS	0.516	0.856	0.364	0.163	0.125	0.133	0.641	1.083	0.443	0.188	0.123	0.074
	CountMax	0.671	0.891	0.641	0.266	0.105	0.015	0.524	0.767	0.456	0.163	0.064	0.010
	FEA-Sketch	0.337	0.509	0.261	0.170	0.118	0.121	0.331	0.488	0.268	0.158	0.117	0.114
70 kB	PR-Sketch	0.833	0.862	0.816	0.823	0.817	0.779	0.816	0.846	0.802	0.788	0.778	0.752
	DS	0.373	0.554	0.304	0.146	0.127	0.081	0.482	0.797	0.346	0.143	0.099	0.104
	CountMax	0.550	0.791	0.487	0.177	0.068	0.011	0.396	0.612	0.323	0.109	0.042	0.007
	FEA-Sketch	0.278	0.419	0.214	0.146	0.109	0.120	0.280	0.416	0.222	0.137	0.106	0.112

5.4 Evaluation on flow size estimation

Impact of wildcard entry proportion on accuracy. In Figures 5(a) and 6(a), we fix the available memory usage of the adopted sketches to 30 kB and vary the wildcard entry proportion to show the AREs of their estimated flow sizes under Fat-Tree topology and Spine-Leaf topology. From the experimental results, we can notice that FEA-Sketch consistently maintains better flow size estimation performance than existing solutions at different wildcard entry proportions, and the bias of its estimation result reduces with the growing proportion of wildcard entries in the flow table. The improvement occurs because the growing proportion of wildcard entries makes more flows redundantly recorded in both flow entries and the sketch, enriching the available data for the estimation phase and enhancing the estimation performance. Especially, as noted in Table 2, when the wildcard entry proportion is set to 0.7, the ARE of FEA-Sketch is up to 50%, 45%, and 67% lower than CountMax, DS, and PR-Sketch in the Fat-Tree topology, and up to 37%, 58%, and 66% lower in the Spine-Leaf topology. Therefore, we can reach a summary that incorporating flow entries into heavy flow detection significantly boosts the performance of flow size estimation, affirming that the proposed mechanism excels beyond existing solutions in achieving accurate estimation results.

Impact of memory size M on accuracy. In Figures 5 and 6, we vary the available memory usage of the adopted sketches to show the flow size estimation accuracy of their detected heavy flows under different topologies. The experimental results show that FEA-Sketch consistently achieves lower ARE than existing solutions under different memory consumption limitations. Notably, as the available memory space for FEA-Sketch increases, its estimation accuracy improves markedly. And, according to

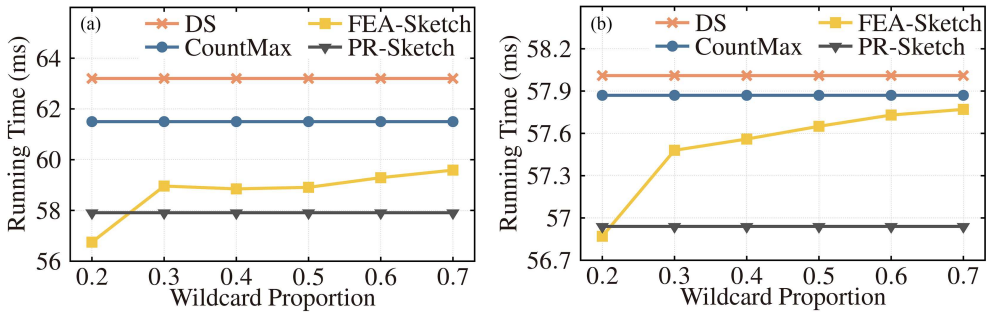

Figure 7 (Color online) Running time of solutions under different topologies. (a) Fat-Tree topology; (b) Spine-Leaf topology.

the data in Table 2, with 30 kB of memory space, the ARE of FEA-Sketch is up to 43%, 45%, and 48% lower than CountMax, DS, and PR-Sketch in the Fat-Tree topology, and up to 37%, 58%, and 49% lower in the Spine-Leaf topology. These results demonstrate that FEA-Sketch can sufficiently utilize the flow entries and has a better memory efficiency than existing solutions.

5.5 Evaluation on running time

Figure 7 illustrates the average running time of the forwarded packets under Fat-Tree topology and Spine-Leaf topology. The experimental results show that FEA-Sketch is always faster than Count-Max and DS while marginally slower than PR-Sketch under different topologies. However, its deficiency in running time still seems acceptable, since FEA-Sketch consistently achieves better performance than existing solutions under different settings while its computation overhead also holds on a satisfactory level.

6 Conclusion

This paper proposes a heavy flow detection solution FEA-sketch that employs a differentiated flow recording strategy to record flow statistics using the flow table and sketch for memory-efficient and computationally efficient heavy flow detection. Meanwhile, we present an optimization-based estimation algorithm to accurately recover per-flow sizes from the aggregated statistics in those sharing entries in the table or counters in the sketch, refining the estimation accuracy for detected heavy flows. After that, we expand FEA-Sketch to the distributed measurement setting with a hop-based collaborative measurement strategy, reducing switches' measurement workloads across the network by avoiding redundant measurements. The experimental results on real Internet traces show that the accuracy of heavy flow detection is improved up to 1.95 times, and the bias of flow size estimation is improved up to 2.99 times. Therefore, leveraging the recorded traffic statistics in both the flow table and the sketch can significantly improve the performance of heavy flow detection. Apparently, flow size is not the only metric in traffic measurement tasks. In future work, we plan to further exploit the various types of statistics maintained in the counter field of the flow table, as well as expand the idea of joint utilization of both flow table and sketch for other possible traffic measurement tasks with different metrics.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant Nos. 62332013, 62072322, U20A20182, 62202322) and Natural Science Foundation of Jiangsu Province (Grant No. BK20210706).

References

1 Hong C-Y, Kandula S, Mahajan R, et al. Achieving high utilization with software-driven WAN. In: Proceedings of ACM SIGCOMM, 2013. 15–26
2 Zhao G M, Xu H L, Fan J Y, et al. Achieving fine-grained flow management through hybrid rule placement in SDNs. IEEE Trans Parallel Distrib Syst, 2021, 32: 728–742
3 Feldmann A, Greenberg A, Lund C, et al. Deriving traffic demands for operational IP networks: methodology and experience. IEEE ACM Trans Networking, 2001, 9: 265–279
4 Krishnamurthy B, Sen S, Zhang Y, et al. Sketch-based change detection: methods, evaluation, and applications. In: Proceedings of ACM IMC, 2003. 234–247
5 Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams. In: Proceedings of Springer ICALP, 2002. 693–703
6 Cormode G, Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications. J Algorithms, 2005, 55: 58–75
7 Yu X W, Xu H L, Yao D, et al. CountMax: a lightweight and cooperative sketch measurement for software-defined networks. IEEE ACM Trans Networking, 2018, 26: 2774–2786
8 Gu L Y, Tian Y, Chen W, et al. Per-flow network measurement with distributed sketch. IEEE ACM Trans Netwing, 2024, 32: 411–426

9 BMV2. The behavioral model for P4. 2024. https://github.com/p4lang/behavioral-model
10 Xu H L, Huang H, Chen S G, et al. Achieving high scalability through hybrid switching in software-defined networking. IEEE ACM Trans Networking, 2018, 26: 618–632
11 Lu Y, Montanari A, Prabhakar B, et al. Counter braids: a novel counter architecture for per-flow measurement. In: Proceedings of ACM SIGMETRICS, 2008. 121–132
12 Sheng S Y, Huang Q, Wang S, et al. PR-sketch: monitoring per-key aggregation of streaming data with nearly full accuracy. Proc VLDB Endow, 2021, 14: 1783–1796
13 Li Y P, Yu X, Yang Y L, et al. Pyramid family: generic frameworks for accurate and fast flow size measurement. IEEE ACM Trans Networking, 2022, 30: 586–600
14 Li H Y, Chen Q Z, Zhang Y X, et al. Stingy sketch. Proc VLDB Endow, 2022, 15: 1426–1438
15 Metwally A, Agrawal D, Abbadi A E. Efficient computation of frequent and top-k elements in data streams. In: Proceedings of International Conference on Display Technology, 2005. 398–412
16 Yang T, Zhang H W, Li J Y, et al. HeavyKeeper: an accurate algorithm for finding top-k elephant flows. IEEE ACM Trans Networking, 2019, 27: 1845–1858
17 Zhao B H, Li X, Tian B Y, et al. DHS: adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In: Proceedings of ACM SIGKDD, 2021. 2285–2293
18 Huang H, Yu J K, Du Y, et al. Memory-efficient and flexible detection of heavy hitters in high-speed networks. Proc ACM Manag Data, 2023, 1: 1–24
19 Xu H L, Chen S G, Ma Q P, et al. Lightweight flow distribution for collaborative traffic measurement in software defined networks. In: Proceedings of IEEE INFOCOM, 2019. 1108–1116
20 Basat R B, Einziger G, Tayh B. Cooperative network-wide flow selection. In: Proceedings of IEEE ICNP, 2020. 1–11
21 Ben-Basat R, Einziger G, Feibish S L, et al. Routing-oblivious network-wide measurements. IEEE ACM Trans Networking, 2021, 29: 2386–2398
22 Kingma D P, Ba J. Adam: a method for stochastic optimization. In: Proceedings of International Conference on Learning Representations, 2015
23 Jin X, Li X Z, Zhang H Y, et al. NetCache: balancing key-value stores with fast in-network caching. In: Proceedings of ACM SOSP, 2017. 121–136
24 Sivaraman A, Kim C, Krishnamoorthy R, et al. DC.p4: programming the forwarding plane of a data-center switch. In: Proceedings of ACM SOSR, 2015
25 Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. In: Proceedings of ACM SIGCOMM, 2008. 63–74
26 Alizadeh M, Edsall T. On the data path performance of leaf-spine datacenter fabrics. In: Proceedings of IEEE HOTI, 2013. 71–74
27 CAIDA. The CAIDA UCSD Anonymized Internet Traces 2016. 2019. http://www.caida.org/data/passive/passive_2016_dataset.xml.
28 Papagiannaki K, Taft N, Zhang Z L, et al. Long-term forecasting of internet backbone traffic. IEEE Trans Neural Netw, 2005, 16: 1110–1124
29 Draves R P, King C, Venkatachary S, et al. Constructing optimal IP routing tables. In: Proceedings of IEEE INFOCOM, 1999. 88–97
30 Source Code 2024. The P4 source codes of FEA-Sketch and baseline algorithms. https://github.com/WuXiaoCan/FEA-Sketch.git