Effective Network-Wide Traffic Measurement: A Lightweight Distributed Sketch Deployment

Fuliang Li[†], Kejun Guo[†], Jiaxing Shen[‡], Xingwei Wang[†✉]
[†]Northeastern University, Shenyang 110819, China, [‡]Lingnan University, Hong Kong, China
Email: lifuliang@mail.neu.edu.cn, 2301867@stu.neu.edu.cn, jiaxingshen@LN.edu.hk, wangxw@mail.neu.edu.cn

***Abstract*—Network measurement is critical for various network applications, but scaling measurement techniques to the network-wide level is challenging for existing sketch-based solutions. Centralized sketch deployment provides low resource usage but suffers from poor load balancing. In contrast, collaborative measurement achieves load balancing through flow distribution across switches but requires high resource usage. This paper presents a novel lightweight distributed deployment framework that overcomes the limitations above. First, our framework is lightweight such that it splits sketches into segments and allocates them across forwarding paths to minimize resource usage and achieve load balancing. This also enables per-packet load balancing by distributing computations across switches. Second, our framework is also optimized for load balancing by coordinating between flows and enabling finer-grained flow distribution. We evaluate the proposed framework on various network topologies and different sketch deployments. Results indicate our solution matches the load balancing of collaborative measurement while approaching the low resource usage of centralized deployment. Moreover, it achieves superior performance in per-packet load balancing, which is not considered in previous deployment policies. Our work provides efficient distributed sketch deployment to strike a balance between load balancing and resource usage enabling effective network-wide measurement.**

***Index Terms*—Sketch, network measurement, distributed deployment, load balancing**

I. Introduction

A. Background and Motivations

Network measurement serves as the foundation for various network applications, such as traffic engineering [1], congestion control [2], and anomaly detection. Existing sketch-based solutions have been widely used due to their ability to achieve high accuracy with low memory usage. However, in the context of data centers and backbone networks, the network topologies are often too large and complex to measure. Nevertheless, existing sketches primarily focus on individual points without considering network-wide traffic measurement. Therefore, to achieve network-wide traffic measurement, sketch-based solutions require deployment policies which mainly consist of collaborative measurement [3]–[8], and centralized deployment [9], [10].

Collaborative measurement involves each switch in the network measuring a subset of flows to achieve optimal load balancing. However, it is limited to the following two aspects:

- **High resource usage:** In collaborative measurement, each switch in the network measures a subset of flows and each packet performs a complete sketch operation at a certain switch. To achieve good load balancing effect, nearly all switches will be involved in the measurement process. In programmable switches like Tofino [11], resources are allocated statically. Since each packet performs a complete sketch operation at a certain switch, all switches deploying the sketch in the network must reserve hash units and Stateful ALUs (SALUs) for a complete sketch, which results in high resource usage. It becomes even worse when it is necessary to deploy multiple sketches [12].
- **Poor per-packet load balancing:** Collaborative measurement suffers from *per-packet load imbalance*, as each packet undergoes a complete sketch operation at a certain switch rather than having all switches along the path jointly perform the sketch operation. Per-packet load balancing is thus crucial, as it distributes multiple hash computations and memory accesses for each packet evenly across multiple switches. Collaborative measurement divides traffic by flow granularity, causing severe load imbalance between switches handling elephant flow versus mouse flow. Since per-packet load balancing shares the measurement of each packet across multiple switches, it reduces the impact of differing flow sizes and further optimizes load balancing.

Centralized deployment involves deploying sketches on core switches, which can be obtained through historical traffic statistics or network topology. Since the sketch is only deployed on core switches, centralized deployment minimizes resource usage. However, it is limited in the following aspects:

- **Poor load balancing:** Centralized deployment concentrates the measurement load and resource usage on core switches, resulting in an unbalanced distribution between the core switches and other switches.
- **Poor per-packet load balancing:** Similar to collaborative measurement, each packet undergoes a complete sketch operation in a certain switch, leading to per-packet load imbalance.
- **Redundant measurement:** Due to lack of collaborative strategy, some flows may traverse multiple switches where the sketch is deployed, leading to redundant measurements.

B. The Proposed Solution and Contributions

As presented in Table I, centralized deployment exhibits low resource usage but suffers from load imbalance, while collaborative measurement achieves good load balancing but entails high resource usage. In this paper, we propose a distributed sketch deployment that could strike a balance

TABLE I: Comparison of existing solutions and design goals.

Advantages	Centralized deployment	Collaborative measurement	Our goal
Low resource usage	√	×	√
Load balancing	×	√	√
Per-packet load balancing	×	×	√

between resource usage and load balancing while ensuring per-packet load balancing. The previous distributed deployment solution most similar to ours is DISCO [13]. However, DISCO ignores interactions between forwarding paths and fine-grained flow distribution in real network topologies, leading to severe switch load imbalance. Moreover, DISCO is limited to heavy hitter detection and certain sketches, lacking generalizability to other scenarios. Lastly, DISCO requires substantial domain expertise for deployment. To summarize, the contributions of our proposed lightweight distributed sketch deployment are illustrated as follows.

Contribution I: lightweight distributed deployment framework for sketch. We introduced a lightweight distributed deployment framework for sketch, aiming to combine the benefits of centralized deployment and collaborative measurement. The proposed framework achieves low resource usage, load balancing, and per-packet load balancing. By leveraging the concept of the k-hash model, we divide the existing sketch into segments and deploy them across multiple switches. Each switch stores a segment of the sketch, performs partial hash calculations and memory accesses, and the measurement of each packet is collaboratively completed by multiple switches. This approach ensures low resource usage, balanced measurement workload, and per-packet load balancing. Additionally, distributed deployment offers the potential to expand resource utilization beyond the constraints of existing single-point sketches.

Contribution II: optimally distributed deployment framework for sketch. To address issues such as load imbalance in complex topologies, we propose an optimally distributed deployment framework for sketch, which further optimizes the load balancing process. This framework incorporates two optimizations. First, we eliminated the mutual influence between flows through coordination, effectively reducing the load on the aggregation nodes. Second, we enhanced load balancing by fine-grained flow distribution. Although the optimally distributed deployment framework requires a slight increase in resource usage, it remains significantly lower than that of collaborative measurement.

Contribution III: extensive experimental verification. To validate the effectiveness of our proposed solutions, we conducted comprehensive experiments across three different topologies and four kinds of sketches. The results demonstrate that our approach surpasses current state-of-the-art techniques, achieving low resource usage, optimal load balancing, and per-packet load balancing.

II. Related Work

In this section, we provide background on different types of sketches and prior work on network-wide traffic measurement.

A. Different Kinds of Sketches

Sketch is a kind of probabilistic data structure. Classic sketches support single-point deployment without considering the possibility of distributed deployment. According to the supported queries, we classify them into three kinds.

1) Sketches for Membership Queries: Membership queries check whether a flow is present. A typical membership query sketch is Bloom Filter [14]. Bloom Filter consists of k equal-length register arrays, and each register array is associated with a hash function. When inserting a flow, Bloom Filter maps it into k registers through k hash functions and sets the mapped bits to 1. When querying a flow, it checks the mapped bits of the k hashes, reporting true only if all are 1. Recently, variants of Bloom Filter have been proposed to meet the requirements of different applications such as CBF [15], EBF [16], NBF [17], and HABF [18].

2) Sketches for Frequency Estimation: Frequency estimation counts the number of packets in a flow. Typical frequency estimation sketches include CM sketch [19], CU sketch [20], and CO sketch [21]. The CM sketch comprises k equal-length counter arrays, where each array uses a hash function. When inserting a flow, CM sketch maps it into k counters via the k hashes, incrementing each counter by 1. When querying a flow, it checks the k mapped counters and reports the smallest value. To improve memory utilization given skewed traffic, recent sketches split flows by size such as Tower sketch [22], Elastic sketch [23], BitSense [24] and HeavyGuardain [25].

3) Sketches for Heavy Hitter Detection: Heavy hitter detection identifies flows exceeding a frequency threshold. A typical example is the MV sketch [26], comprising k rows of bucket arrays. The MV sketch uses the MJRTY algorithm for insertion. When querying a flow, it checks the k mapped buckets and returns the minimum value. MV sketch provides high recall and precision. Recently, there are also some other sketches that perform well, such as HeavyKeeper [27], Hashpipe [28] and Unbiased Space Saving [29].

B. Sketch Deployment for Network-Wide Traffic Measurement

The existing sketch network-wide deployment solutions are mainly centralized deployment [9], [10], collaborative measurement [3], [5]–[8], and distributed deployment solutions [13].

Centralized deployment is to deploy sketch at core switches, which can be obtained through historical traffic statistics. It is modeled as an integer linear programming problem, and the switch aggregating traffic is selected for sketch deployment. Centralized deployment has the lowest resource usage but results in a poor load-balancing effect.

Collaborative measurement means that each switch in the network measures a subset of flows, which has an excellent load-balancing effect. But all collaborative measurement solutions

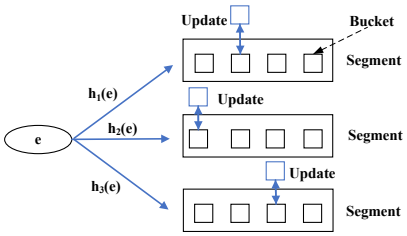


Fig. 1: An illustration of k-hash model sketch where $k = 3$.

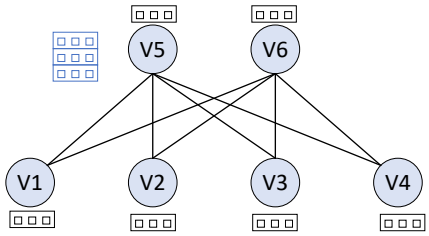


Fig. 2: A distributed sketch deployment in leaf-spine topology.

suffer from extensive resource usage and per-packet load imbalance. Currently, the most typical collaborative measurement solution is NSPA [3] that maintains a sampling probability for each switch. If the packet is not metered by one of the preceding switches on the routing path, a hash calculation is required to determine whether to perform the measurement. If the packet has not been metered until the egress switch, the packet will be metered at the egress switch. NSPA has a good load-balancing effect. Although Distributed Sketch [4] claims to be a distributed framework, we find that it still needs to reserve the hash units and SALUs for a complete sketch. Therefore, it is more like a collaborative measurement solution. There are many other existing collaborative solutions, such as DCM [5], CFS [6], cSamp [7], and HiFi [8].

DISCO [13] is a recent distributed deployment solution that is most relevant to our proposed solution. However, our framework differs from DISCO in four key aspects. First, DISCO overlooks interactions between forwarding paths and fine-grained flow distribution in real topologies, leading to severe switch load imbalance. Second, DISCO is limited to heavy hitter detection and certain sketches, which could not be easily extend to other scenarios. Third, for any given topology and traffic data, DISCO cannot directly provide an intuitive deployment solution, requiring domain expertise. Fourth, DISCO lacks rigorous theoretical analysis of feasibility.

III. Distributed Sketch Deployment Framework: Basic Version

In this section, we propose a lightweight distributed deployment framework for sketch, which only requires us to deploy the sketch segment on the specific switches.

A. Sketch Model

Sketch has various data structures. One of the most advanced classes is the k-hash model. The details of this model are illustrated as follows.

Data structure: As shown in Fig. 1, the k-hash model sketch consists of k segments, where each segment contains multiple elements and is associated with a hash function. The elements within a segment are called buckets, which could be bits, counters, or key-value pairs depending on specific sketches.

Insertion: When inserting a flow e, the k-hash model sketch maps it into k buckets through k hash functions, one in each segment. Insertion is performed independently for each mapped bucket.

TABLE II: Comparison of different solutions in leaf-spine topology.

Solutions	Resource usage (Number of SALUs)	Load standard deviation	Per-packet load standard deviation
CD	6	67.9	1.41
NSPA	18	0	1.41
LDD	6	0	0

It should be noted that the sketch-like Hashpipe [28] is not a k-hash model because its multiple insertions are not independent.

B. Baseline Solutions and Our Observations

To motivate our design, we first introduce two baseline solutions and illustrate them using network examples. As shown in Fig. 2, it is a typical leaf-spine network topology consisting of two spine switches and four leaf switches. Each leaf switch receives 24 different flows with only one packet and forwards them equally to the remaining three leaf switches. Let's take the k-hash model sketch that needs to perform three hash calculations and memory accesses as an example ($k = 3$). Previous practice has proved that it is good enough for the k-hash model sketch to perform 2-3 hash calculations in the network. The resources of the programmable switch are allocated statically. In the k-hash model sketch, we use the Tofino switch to measure the number of hash units that need to be used, which is about three times the number of SALUs. Therefore, for the convenience of comparison, we use the number of SALUs to represent the resource usage of the k-hash model sketch. Since the measurement load depends on the number of flows measured by the switch and the number of hash calculations that need to be performed by the sketch within the switch, we use the product of the two to represent the measurement load. We use the standard deviation of the number of hash calculations or memory accesses to reflect the per-packet load balancing, performed by each packet on each switch in its forwarding path. Table II gives a comparison of different solutions for measurement load and resource usage.

The first solution is centralized deployment (CD) [9], [10]. The deployment results of [9] and [10] are similar. Sketch is deployed on the switches with the most concentrated traffic in the network. In leaf-spine topology, we deploy the complete sketch on all spine switches (V5 and V6). In other words, a complete sketch of three segments is deployed on each spine switch. Only the spine switches need to reserve $3k$ hash units and k SALUs. It minimizes overall resource usage but provides no load balancing.

The second solution is collaborative measurement. Each switch in the network measures a subset of flows. We take the state-of-the-art NSPA [3] as an example. The complete sketch of three segments is deployed on all switches (V1 - V6), and each switch measures a subset of the flows. Each switch needs to reserve $3k$ hash units and k SALUs. NSPA has a good load-balancing effect, but it does not take into account the limitations of switch resources and the possibility of sketch segments allocation. Inspired by this, we propose the distributed deployment solution.

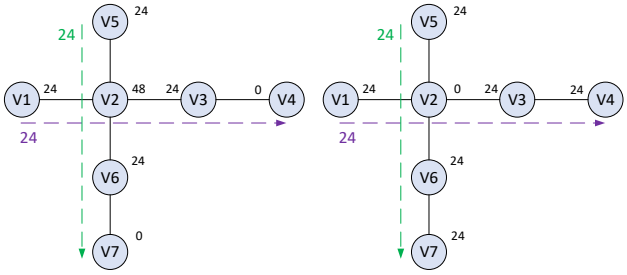
The third solution is our proposed lightweight distributed deployment framework (LDD). It leverages the inherent parallelizability of k-hash sketches. LDD distributedly deploys the k sketch segments across switches along forwarding paths. This realizes two key benefits: First, by allocating one segment per switch, LDD achieves resource efficiency equaling centralized deployment and load balancing effect equivalent to collaborative measurement. Second, LDD provides per-packet load balancing by involving all switches equally in measurement. A potential accuracy concern is that distributed segments induce disjoint flow sets. We mathematically prove in Section V that LDD preserves the statistical guarantees of centralized deployment despite this segment splitting.

C. Lightweight Distributed Deployment Framework for Sketch

Collaborative measurement only considers load balancing by dividing flows and ignores the possibility of load balancing by splitting sketch segments. Our proposed solution is based on two key facts: 1) k-hash sketches allow parallel computing since each segment buckets operate independently, and 2) forwarding paths contain multiple switches. We leverage these by dividing sketches into segments and deploying across switches. Each switch stores one segment, performing partial computation and memory accesses per packet. This collaboratively completes measurement over multiple nodes. To preserve sketch accuracy guarantees under distribution, the path must have k or more segments with independent hash functions. Our framework has three design goals:

Objective-I: minimizing resource usage in each forwarding paths. It ensures that the distributed deployment framework for sketch has the least resource usage close to the centralized deployment. It needs to accumulate the resources used by the switches on each forwarding path. The resources of overlapping switches on the forwarding path need to be accumulated repeatedly. The least resource usage of the forwarding path promotes the unification of the solution set of the distributed deployment and the solution set of the centralized deployment solution into one solution set. As shown in Fig. 2, assuming the forwarding path V1-V5-V3, Objective-I will drive the solution deploying the complete sketch of three segments in V5 and the solution deploying one segment each in V1, V5, and V3 into one solution set.

Objective-II: balance the number of sketch segments in each forwarding path. It ensures per-packet load balancing. In the above solutions, we select the distributed deployment solution which deploys a sketch segment in V1, V5, and V3, respectively. Because each switch only stores a segment, performs one hash calculation and memory access, and the measurement of each packet is collaboratively completed by multiple switches. Thus the distributed deployment solution has better per-packet load balancing. At the same time, since it distributes segments evenly across multiple switches as much as possible, we will see that it will help select the deployment solution with the lowest resource usage in Section IV.



(a) Deployment Solution-I. (b) Deployment Solution-II.

Fig. 3: Deployment satisfies Objective-I and II. The black numbers represent switch's measurement load. Solution-II that satisfies Objective-III is more load balanced.

Objective-III: network-wide measurement load balancing. It ensures that the solution with the most balanced measurement load is selected in the above solution sets. As shown in Fig. 2, the deployment solution that satisfies Objective-I and II in the leaf-spine network happens to be load balanced. Therefore, to further illustrate the role of Objective-III, we give another example. As shown in Fig. 3, assume that there are two paths, each path has 24 flows, and each flow has one packet that needs to perform three hash calculations and memory accesses. The two deployment solutions simultaneously satisfy the Objective-I and II. And Objective-III guarantees the choice of solution Fig. 3b, because it is more load balanced.

The priorities of Objective-II and Objective-III can be adjusted according to different requirements. If you think measurement load balancing is more important than per-packet load balancing, you can set Objective-III priority higher than that of Objective-II. In this paper, we set the priority of Objective-II higher than that of Objective-III.

In order to formulate this problem, we set the set of switches in the network as $V = \{v_1, v_2 \dots v_n\}, n = |V|$. The number of sketch segments deployed by each switch is expressed as D_v $(v \in V)$. We use S_i^j to express the flows and total flow size from the ingress switch v_i to the egress switch v_j from the measured flow matrix, which is also known to the controller. We set the set of flows is Γ $(S_i^j \in \Gamma)$ and the forwarding path of flow S_i^j is $P_{S_i^j} = \{v_i, \dots v_j\}$. We use σ to represent the variance and $L_v(v \in V)$ to represent the measurement load on the switch v.

We formulate the problem as follows:

$$Opt. \begin{cases} min \sum\limits_{S_i^j \in \Gamma} \sum_{v \in P_{S_i^j}} D_v \\ min \sum\limits_{S_i^j \in \Gamma} \sigma\left(\{D_v\}, \forall v \in P_{S_i^j}\right) \\ min\ \sigma\left(\{L_v\}, \forall v \in V\right) \end{cases} \tag{1}$$

$$S.t. \begin{cases} \sum\limits_{v \in P_{S_i^j}} D_v \geq k \quad \forall S_i^j \in \Gamma \\ L_v = \sum\limits_{S_i^j \in \Gamma} \left(D_v \times S_i^j,\ if\ v \in P_{S_i^j}\right) \quad \forall v \in V \\ D_v \in [0, 1, ...k] \quad \forall v \in V \end{cases} \tag{2}$$

The first constraint in Eq. 2 restricts each forwarding path to have k or more sketch segments, which preserves the error bound of sketch. The three objective functions in Eq. 1 correspond to Objective-I, II and III, respectively. We use the solver Gurobi [30] to solve.

Then, let's explain how many sketch segments are deployed in the switch and the memory size of each sketch. The number of sketch segments to be deployed is D_v. The memory size of sketch is calculated by $\frac{L_v}{\sum_{v \in V} L_v} \times Memory$ in switch v.

Our solution unifies the benefits of centralized deployment and collaborative measurement. We realize collaborative measurement's load balancing with resource efficiency approaching centralized deployment. Additionally, we uniquely achieve per-packet load balancing, with each packet measured collaboratively across multiple switches. By solely leveraging the distributed sketch, our solution provides a lightweight and balanced network measurement.

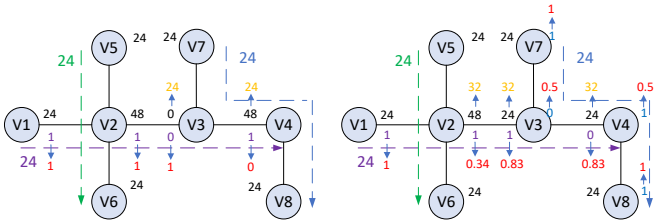
IV. Distributed Sketch Deployment Framework: Optimized Version

The lightweight distributed deployment framework for sketch performs well in most data center topologies. Unfortunately, the lightweight distributed deployment framework suffers from load imbalance in complex network topologies. In order to optimize load balancing performance, we present the optimally distributed deployment framework for sketch.

A. Baseline Solutions and Our Observations

We use a more complex network topology to reveal the problems of the lightweight distributed deployment framework for sketch. In the topology shown in Fig. 4, we assume that there are three paths, each path has 24 flows, and each flow has one packet that needs to perform three hash calculations and memory accesses ($k = 3$). Centralized deployment and collaborative measurement will not be described.

The third solution is the lightweight distributed deployment framework (LDD). It has two problems. First, multiple flow forwarding paths will affect the deployment of the sketch segments, which causes the aggregation switches to be heavily loaded. For example, as shown in Fig. 4a, we need to have each



(a) Optimization-I: eliminate the impact on the allocation of segments between flows.

(b) Optimization-II: fine-grained flow distribution.

Fig. 4: Comparison of distributed deployment solutions in synthetic topology. The red numbers represent the value of the optimized $D_v^{S_i^j}$. The yellow numbers represent optimized switch's measurement load.

TABLE III: Comparison of different solutions in synthetic topology.

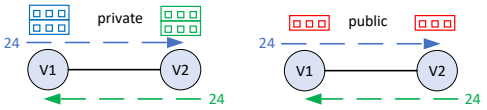
Solutions	Resource usage (Number of SALUs)	Load standard deviation	Per-packet load standard deviation
CD	6	62.4	1.33
NSPA	24	4.8	1.33
LDD	7	14.4	0.29
ODD	8	3.9	0.29

packet perform three hash calculations and memory accesses. In order to satisfy Objective-I and II, suppose the blue path flow choose switch V7, V4 and V8. Since the green path flow has only three switches, it can only choose switch V5, V2 and V6. Due to the influence between paths, this results in purple path flows having to use switch V2 and V4 even when switch V1 and V3 are all available. Second, the flow distribution of the previous solution is coarse-grained, and there is the possibility of further optimization. For example, in the k-hash model sketch, we set $k = 2$ for this example. In the leaf-spine topology, we will deploy a sketch segment in all leaf switches. Spine switches will be unused. If one sketch segment is deployed on each switch, the ingress leaf switch completes the first measurement of the first 67% of flows, the spine switch completes the first measurement of the remaining 33% of flows and the second measurement of the first 33% of flows, and the egress leaf switch completes the second measurement of the remaining 67% of flows, the better load balancing will be achieved. In other words, each switch performs one measurement on 67% of the path flows.

The fourth solution is the optimally distributed deployment framework (ODD). It performs the two-step optimization of LDD. First, we eliminate the impact on the allocation of segments between flows. Second, we achieve further load balancing through fine-grained flow distribution.

B. Optimally Distributed Deployment Framework for Sketch

First, in order to eliminate the impact on the allocation of sketch segments between flows, we need to redefine D_v. In Eq. 1 and Eq. 2, D_v represents the number of sketch segments of each switch, which is prepared for all flows passing through



(a) Deployment Solution-I. (b) Deployment Solution-II.

Fig. 5: Deployment satisfies Objective-I and III. Solution-II that satisfies Objective-II has the lowest resource usage.

the switch. In other words, each flow passing through the switch where sketch is deployed needs to be measured in LLD, which causes load imbalance. We modify D_v to $D_v^{S_i^j}$, which represents the number of sketch segments specially prepared by the switch for the certain path flow S_i^j. We do not make any changes to other goals and constraints.

We formulate the problem as follows:

$$Opt. \begin{cases} min \sum\limits_{S_i^j \in \Gamma} \sum_{v \in P_{S_i^j}} D_v^{S_i^j} \\ min \sum\limits_{S_i^j \in \Gamma} \sigma\left(\left\{D_v^{S_i^j}\right\}, \forall v \in P_{S_i^j}\right) \\ min\ \sigma\left(\{L_v\}, \forall v \in V\right) \end{cases} \tag{3}$$

$$S.t. \begin{cases} \sum\limits_{v \in P_{S_i^j}} D_v^{S_i^j} \geq k \quad \forall S_i^j \in \Gamma \\ L_v = \sum\limits_{S_i^j \in \Gamma} \left(D_v^{S_i^j} \times S_i^j,\ if\ v \in P_{S_i^j}\right) \quad \forall v \in V \\ D_v^{S_i^j} \in [0, 1, ...k] \quad \forall v \in V, S_i^j \in \Gamma \end{cases} \tag{4}$$

As shown in Fig. 4a, the result of Eq. 3 and Eq. 4 prompted the sketch segment of S_1^4 to change from being deployed on switches V1, V2 and V4 to being deployed on switches V1, V2 and V3. We only modify D_v to $D_v^{S_i^j}$ to avoid the impact on the allocation of sketch segments between flows, which achieves better load balancing.

In Eq. 3, Objective-II is important. In addition to per-packet load balancing, since we modify D_v to $D_v^{S_i^j}$, this results in the inability to select the solution with the lowest resource usage from the measurement load balancing solutions. As shown in Fig. 5 ($k = 2$ for this example), segments of different colors are private to the path flow of the corresponding color. Two deployment solutions simultaneously satisfy Objective-I and III, but Solution-II that satisfies Objective-II has the lowest resource usage.

Second, in order to achieve finer-grained allocation. We further optimize the results above. $Last\_D_v^{S_i^j}$ in Eq. 6 is the result of the previous step (Eq. 1 and Eq. 2 or Eq. 3 and Eq. 4 are both feasible). At the same time, we relax $D_v^{S_i^j}$ to float, which represents the percentage of the flows needs to be measured. We further formulate the problem as follows:

$$Opt. \begin{cases} min \sum\limits_{S_i^j \in \Gamma} \sum_{v \in P_{S_i^j}} D_v^{S_i^j} \\ min\ \sigma\left(\{L_v\}, \forall v \in V\right) \\ min \sum\limits_{S_i^j \in \Gamma} \sigma\left(\left\{D_v^{S_i^j}\right\}, \forall v \in P_{S_i^j}\right) \end{cases} \tag{5}$$

$$S.t. \begin{cases} \sum\limits_{v \in P_{S_i^j}} D_v^{S_i^j} \geq k \quad \forall S_i^j \in \Gamma \\ L_v = \sum\limits_{S_i^j \in \Gamma} \left(D_v^{S_i^j} \times S_i^j,\ if\ v \in P_{S_i^j}\right) \quad \forall v \in V \\ D_v^{S_i^j} \geq min\left(\left\{Last\_D_l^{S_i^j}\right\}, \forall l \in P_{S_i^j}\right) \forall v \in V, S_i^j \in \Gamma \\ D_v^{S_i^j} \leq max\left(\left\{Last\_D_l^{S_i^j}\right\}, \forall l \in P_{S_i^j}\right) \forall v \in V, S_i^j \in \Gamma \\ D_v^{S_i^j} \in [0, k], Float \quad \forall v \in V, S_i^j \in \Gamma \end{cases} \tag{6}$$

In Eq. 5, we set the priority of Objective-III (Network-wide measurement load balancing) over Objective-II (Balance the number of sketch segments in each forwarding path). There are two reasons. 1) Since we relax $D_v^{S_i^j}$ to float, this will always result in an even distribution of the number of segments if we do not change the priorities of the objectives, which makes it impossible to further optimize the measurement load balancing. 2) Meanwhile, since we set the priority of Objective-III over Objective-II, in order to prevent poor per-packet load balancing and high resource usage (high resource usage as shown in Fig. 5), we have already approximated Objective-II in the third constraint and the fourth constraint of Eq. 6 with the help of the previous solution. The result is shown in Fig. 4b. The red number represents the value of the optimized $D_v^{S_1^4}$ and $D_v^{S_7^8}$.

The number of sketch segments that each switch v needs to deploy is $\left\lceil max\ D_v^{S_i^j}, \forall S_i^j \in \Gamma \right\rceil$. The memory size of sketch is calculated by $\frac{L_v}{\sum_{v \in V} L_v} \times Memory$ in switch v. Finally, we can add some additional fields to the forwarding table in the ingress switch to achieve coordination. Our approach is similar to previous work [31]. When a packet arrives, we match the forwarding table entry based on the destination IP. Once an entry is matched, we can obtain the additional fields we added. We add it to the packet header to achieve collaborative measurement. For example, by adding the field "4 1, 0, 1, 1", it means that this path passes through four switches, and the second switch does not perform measurements. The first field represents the number of switches on the flow path, and the second field represents how many segments the packet will be inserted into in the corresponding switch. When the packet passes through a switch, we use the value of the first field as an index and take the value of the corresponding index in the second field to determine how many segments the packet will be inserted into. Then we decrement the first field by one.

As shown in Fig. 4b, $D_{v_1}^{S_1^4}$, $D_{v_2}^{S_1^4}$, $D_{v_3}^{S_1^4}$ and $D_{v_4}^{S_1^4}$ are 1, 0.34, 0.83 and 0.83 respectively. For the path flow S_1^4, the switch V1 completes the first measurement, the switch V2 completes the second measurement of the first 34% of flows, switch V3 completes the second measurement of the remaining 66% of flows and the third measurement of the first 17% of flows, and the switch V4 completes the third measurement of the remaining 83% of flows. Therefore, in the ingress switch V1, we can simply set the first 17% of matching entries to "4 1, 1, 1, 0", 17%-34% of matching entries to "4 1, 1, 0, 1", and the last 66% of matching entries to "4 1, 0, 1, 1".

In summary, the optimally distributed deployment framework for sketch performs the two-step optimization on the measurement load, which achieves better measurement load balancing at the cost of slightly more resource usage.

V. Theoretical Proof

We give the theoretical proof that the difference in the flow set between different segments of the sketch has no effect on the error bound. We refer to previous work SketchConf [32] to provide theoretical proof. It should be noted that we do not emphasize the innovation of our theoretical proof. We just want to illustrate the theoretical basis of our solution. Let's take the CM sketch [19] as an example.

Single-segment sketch error bound: For one segment of the CM sketch with w counters, we consider the counter that e is hashed to. For any one of the other $N-1$ distinct flows, the possibility that it collides with e is $\frac{1}{w}$. Therefore, the number of distinct colliding flows Z follows binomial distribution $B(N-1, \frac{1}{w})$. As $N-1$ is usually large and $\frac{1}{w}$ is small, we approximate that Z follows Poisson distribution, with $\lambda = \frac{N-1}{w}$. In other words, the number of collisions is only related to the ratio of the total flow number to the number of counters. Assume that two segments hashed by flow e have the same number of counters and the same number of flows. Except for flow e, none of their remaining $N-1$ flows are the same. Since they have the same number of counters and the same amount of flows, they have the same number of collisions. Therefore, at this time, the error bound is only related to the characteristics of the $N-1$ flows. Since the flows in the local network are independent and identically distributed, their error bounds are the same.

Multi-segment sketch error bound: As different segments are associated with independent hash functions, the error bound in each segment can be regarded as independent.

In summary, even if the flow set between different segments is different, it will not affect the error bound in the case of independent and identical distribution of the flow.

VI. Experimental results

We apply the lightweight distributed deployment framework for sketch and the optimally distributed deployment framework for sketch to three topologies (Fat tree topology, Leaf-spine topology, and Synthetic topology) and four sketches (Bloom Filter, CM sketch, CO sketch and MV sketch).

A. Test Setup

We use anonymous IP tracking collected from CAIDA in 2018 [33]. Each trace contains about 2.5 million packets. In the case of aggregation with source IP, there are about $70k$ flows. We conduct experiments in three network topologies. The first topology is the fat tree network topology [34], which consists of 4 core switches, 8 aggregation switches, and 8 edge switches. The second topology is leaf-spine network topology, which contains 4 spine switches and 8 leaf switches. The third topology is a synthetic network topology, as shown in Fig. 4. It should be noted that the experimental results are similar in larger topologies. We hash 2.5 million packets to each path through a hash function. We conduct experiments using four solutions. The first solution is the lightweight distributed deployment framework for sketch (LDD). The second solution is the optimally distributed deployment framework for sketch (ODD). The third solution is centralized deployment (CD) [9], [10]. The fourth solution is NSPA [3]. Due to the limitation of the number of Tofino switches, we evaluate the resource usage of each switch in the Tofino switch and then accumulate the resource usage of all switches in the topology. And other experiment is carried out by simulation.

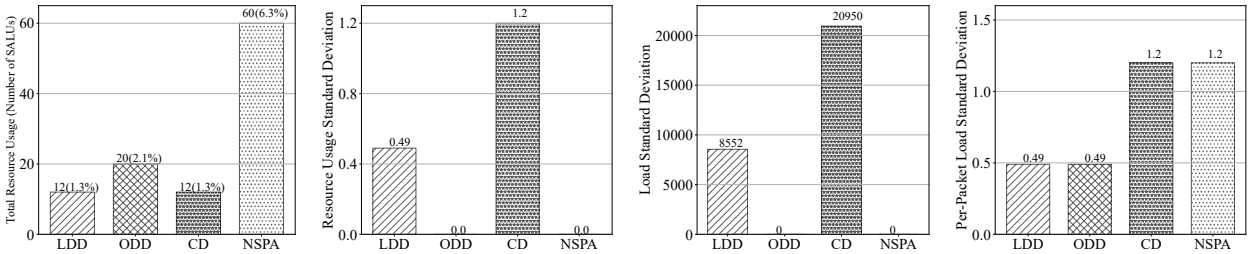
B. Experimental Results on Resource Usage and Load Balancing

1) Experimental Setup: We apply the above four solutions in three network topologies. We set $k = 3$ for the k-hash sketch. The results of CM sketch are as follows. Since most k-hash model sketches are k hash calculations and memory accesses, the results of other k-hash model sketches are also similar. Due to page limitations, the experimental metrics are described in detail in Section III-B.

2) Performance on Resource Usage and Load Balancing: The following are the results of the above experiment.

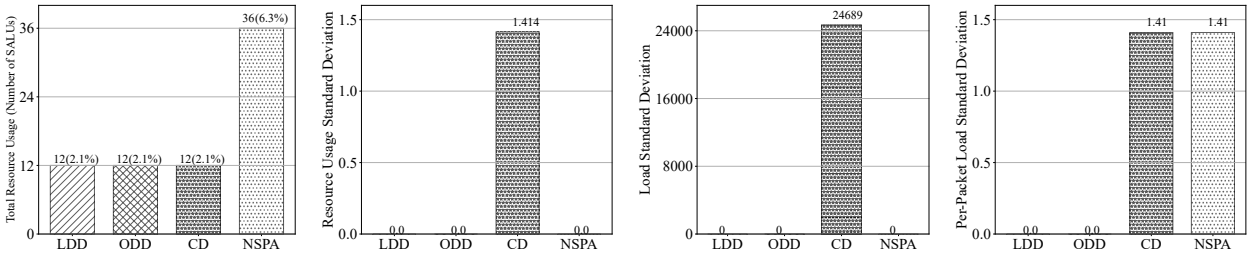
Resource usage: As shown in Fig. 6a, Fig. 7a and Fig. 8a, the percentage in the figure indicates the ratio of the number of SALUs used to the number of SALUs of all switches in the topology. We find that the lightweight distributed deployment framework and centralized deployment has the lowest resource usage. And as shown in Fig. 6b, Fig. 7b and Fig. 8b, the resource allocation of the lightweight distributed deployment framework is more balanced than the centralized deployment. The resource usage of the optimally distributed deployment framework is slightly higher than the resource usage of both but much lower than that of the current optimal collaborative measurement solution NSPA. The resource usage of NSPA is much larger than these three solutions. In the three topologies, the lightweight distributed deployment framework resource usage is only 0.2, 0.33, and 0.29 of the NSPA resource usage. The optimally distributed deployment framework resource usage is only 0.33, 0.33, and 0.33 of the NSPA resource usage.

Measurement load balancing: As shown in Fig. 6c, Fig. 7c and Fig. 8c, we find that the optimally distributed deployment framework and NSPA have the good load balancing effect, and the optimally distributed deployment framework achieves better load balancing effect than NSPA. The lightweight



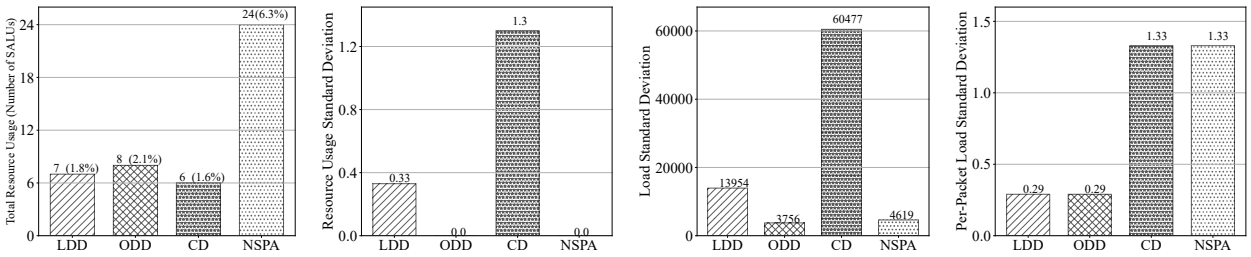
(a) Resource usage. (b) Resource usage load balancing. (c) Measurement load balancing. (d) Per-packet load balancing.

Fig. 6: Resource usage, measurement load balancing and per-packet load balancing on fat-tree topology.



(a) Resource usage. (b) Resource usage load balancing. (c) Measurement load balancing. (d) Per-packet load balancing.

Fig. 7: Resource usage, measurement load balancing and per-packet load balancing on leaf-spine topology.



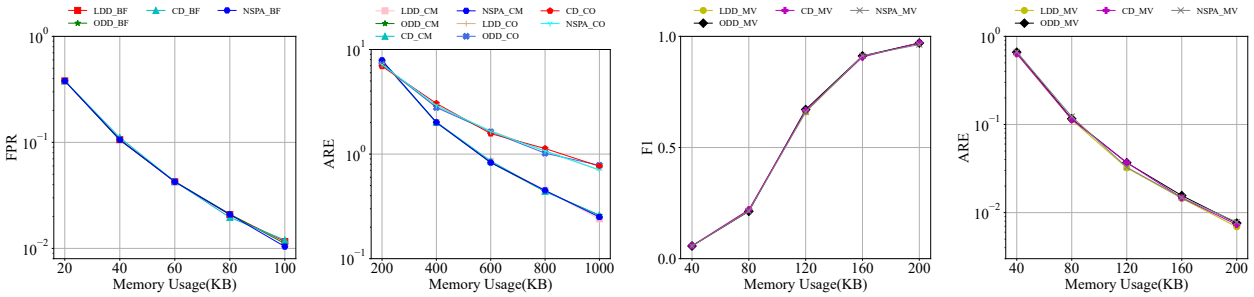
(a) Resource usage. (b) Resource usage load balancing. (c) Measurement load balancing. (d) Per-packet load balancing.

Fig. 8: Resource usage, measurement load balancing and per-packet load balancing on synthetic topology.

distributed deployment framework is slightly worse than the load-balancing effect of both. But on the one hand, the lightweight distributed deployment framework has the lowest resource usage. On the other hand, the lightweight distributed deployment framework has a far better load-balancing effect than centralized deployment. Centralized deployments have the worst load-balancing effect. In the fat tree topology, the measurement load standard deviation of both the optimally distributed deployment framework and NSPA is 0, and the measurement load standard deviation of the lightweight distributed deployment framework is only 0.41 of the measurement load standard deviation of centralized deployment. In the synthetic topology, the optimally distributed deployment framework achieves better load balancing than NSPA. The lightweight distributed deployment framework, the optimally distributed deployment framework, and NSPA are 0.23, 0.06, and 0.08 of the standard deviation of the measurement load of the centralized deployment, respectively.

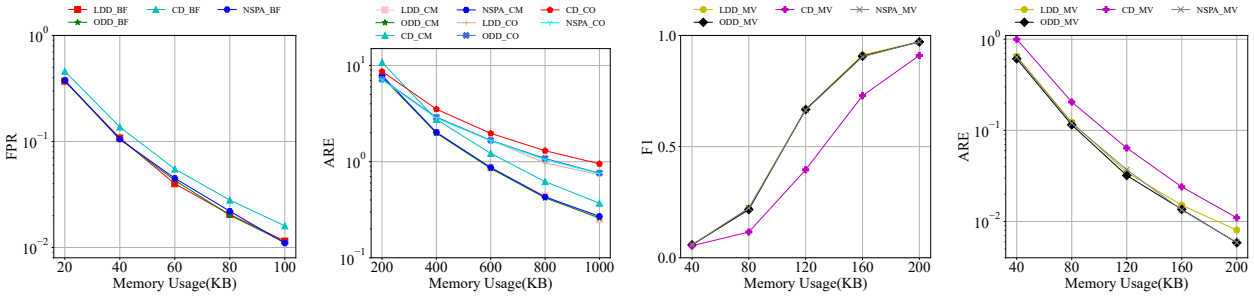
Per-packet load balancing: As shown in Fig. 6d, Fig. 7d and Fig. 8d, we find that the per-packet load standard deviation in the switch of the lightweight distributed deployment framework and the optimally distributed deployment framework is only 0.41 and 0.22 of the centralized deployment and NSPA in fat-tree topology and synthetic topology. As mentioned above, since each packet is measured by multiple switches, per-packet load balancing can reduce the impact of different flow sizes on the measurement load so as to optimize the load balancing effect further.

In summary, the distributed deployment frameworks combine the advantages of centralized deployment and collaborative measurement, which achieve optimal load balancing effects with low resource usage. At the same time, our distributed deployment frameworks achieve per-packet load balancing. Multiple hash calculations and memory accesses for each



(a) Membership query. (b) Frequency estimation. (c) Heavy hitter detection. (d) Heavy hitter detection.

Fig. 9: Accuracy of membership query, frequency estimation, and heavy hitter detection on fat-tree and spine-leaf topology.



(a) Membership query. (b) Frequency estimation. (c) Heavy hitter detection. (d) Heavy hitter detection.

Fig. 10: Accuracy of membership query, frequency estimation, and heavy hitter detection on synthetic topology.

packet are evenly distributed to multiple switches, which can better adapt to elephant flows.

C. Experimental Results on Accuracy

1) Experimental Setup: We apply the above four solutions and compare the metrics of four sketches in three topologies. These sketches belong to three classes of tasks: membership query, frequency estimation, and heavy hitter detection. They are typical k-hash model sketches, we set $k = 3$. We set the heavy hitter threshold to 0.01% of the number of packets. Since the results of the fat tree and the leaf-spine topology are almost identical, we show them in the same figure.

2) Performance on Accuracy: The following are the results of the above experiment.

Membership query: As shown in Fig. 9a and Fig. 10a, in both fat-tree and leaf-spine topologies, we find that the FPR of these four solutions are almost equal as the memory varies. In synthetic topologies, centralized deployment performs slightly worse due to redundant measurement.

Frequency estimation: As shown in Fig. 9b and Fig. 10b, in both fat-tree and leaf-spine topologies, we find that the ARE of these four solutions are almost equal as the memory varies. In synthetic topologies, centralized deployment performs slightly worse due to redundant measurement.

Heavy hitter detection: As shown in Fig. 9c, Fig. 9d, Fig. 10c, and Fig. 10d, in both fat-tree and leaf-spine topologies, we find that the F_1 and ARE of four solutions are almost equal as the memory varies. In synthetic topologies, centralized deployment performs worse due to redundant measurement.

In summary, even if the flow set stored in each segment of the sketch is different, this has no effect on the accuracy of the sketch which is consistent with our theoretical proof.

VII. Conclusion

This paper proposes two distributed deployment frameworks for sketch that support lightweight or optimally distributed deployment for network-wide traffic measurement. Our framework combines the advantages of centralized deployment and collaborative measurement, which achieves a better load-balancing effect than collaborative measurement with low resource usage close to centralized deployment. We applied the proposed framework to diverse topologies and sketches, and the experimental results demonstrated that the proposed framework achieves good load balancing with low resource usage. Meanwhile, our framework achieves per-packet load balancing by evenly distributing hash calculations and memory accesses to multiple switches for each packet.

VIII. Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant Nos. U22B2005, 62032013, 92267206 and 62072091 and the financial support of Lingnan University (LU) (DB23A9) and Lam Woo Research Fund at LU (871236).

REFERENCES

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT'11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2079296.2079304

[2] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "Hpcc: High precision congestion control," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 44–58. [Online]. Available: https://doi.org/10.1145/3341302.3342085

[3] H. Xu, S. Chen, Q. Ma, and L. Huang, "Lightweight flow distribution for collaborative traffic measurement in software defined networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1108–1116.

[4] L. Gu, Y. Tian, W. Chen, Z. Wei, C. Wang, and X. Zhang, "Per-flow network measurement with distributed sketch," *IEEE/ACM Transactions on Networking*, pp. 1–16, 2023.

[5] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 85–90. [Online]. Available: https://doi.org/10.1145/2620728.2620739

[6] R. B. Basat, G. Einziger, and B. Tayh, "Cooperative network-wide flow selection," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, 2020, pp. 1–11.

[7] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "Csamp: A system for network-wide flow monitoring," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. USA: USENIX Association, 2008, p. 233–246.

[8] G. Zhao, H. Xu, J. Fan, L. Huang, and C. Qiao, "Hifi: Hybrid rule placement for fine-grained flow management in sdns," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2341–2350.

[9] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An incrementally-deployable p4-enabled architecture for network-wide heavy-hitter detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 75–88, 2020.

[10] Y. Shi, M. Wen, and C. Zhang, "Incremental deployment of programmable switches for sketch-based network measurement," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–7.

[11] https://barefootnetworks.com/products/brief-tofino/, Barefoot Tofino Switch.

[12] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "Sketchovsky: Enabling ensembles of sketches on programmable switches," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 1273–1292. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/namkung

[13] V. Bruschi, R. B. Basat, Z. Liu, G. Antichi, G. Bianchi, and M. Mitzenmacher, "Discovering the heavy hitters with disaggregated sketches," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 536–537. [Online]. Available: https://doi.org/10.1145/3386367.3431674

[14] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422–426, jul 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, p. 254–265, oct 1998. [Online]. Available: https://doi.org/10.1145/285243.285287

[16] Y. Wu, J. He, S. Yan, J. Wu, T. Yang, O. Ruas, G. Zhang, and B. Cui, "Elastic bloom filter: Deletable and expandable filter using elastic fingerprints," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 984–991, 2022.

[17] H. Dai, J. Yu, M. Li, W. Wang, A. X. Liu, J. Ma, L. Qi, and G. Chen, "Bloom filter with noisy coding framework for multi-set membership testing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6710–6724, 2023.

[18] M. Li, R. Xie, D. Chen, H. Dai, R. Gu, H. Huang, W. Dou, and G. Chen, "A pareto optimal bloom filter family with hash adaptivity," *The VLDB Journal*, vol. 32, no. 3, p. 525–548, jul 2022. [Online]. Available: https://doi.org/10.1007/s00778-022-00755-z

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

[20] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 323–336. [Online]. Available: https://doi.org/10.1145/633025.633056

[21] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," ser. ICALP '02. Berlin, Heidelberg: Springer-Verlag, 2002, p. 693–703.

[22] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang, and N. Zhang, "LightGuardian: A Full-Visibility, lightweight, in-band telemetry system using sketchlets," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 991–1010. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/zhao

[23] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 561–575. [Online]. Available: https://doi.org/10.1145/3230543.3230544

[24] R. Ding, S. Yang, X. Chen, and Q. Huang, "Bitsense: Universal and nearly zero-error optimization for sketch counters with compressive sensing," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 220–238. [Online]. Available: https://doi.org/10.1145/3603269.3604865

[25] T. Yang, J. Gong, H. Zhang, L. Zou, L. Shi, and X. Li, "Heavyguardian: Separate and guard hot items in data streams," ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2584–2593. [Online]. Available: https://doi.org/10.1145/3219819.3219978

[26] L. Tang, Q. Huang, and P. P. C. Lee, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2026–2034.

[27] T. Yang, H. Zhang, J. Li, J. Gong, S. Uhlig, S. Chen, and X. Li, "Heavykeeper: An accurate algorithm for finding top-k elephant flows," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, p. 1845–1858, oct 2019. [Online]. Available: https://doi.org/10.1109/TNET.2019.2933868

[28] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 164–176. [Online]. Available: https://doi.org/10.1145/3050220.3063772

[29] D. Ting, "Data sketches for disaggregated subset sum and frequent item estimation," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1129–1140. [Online]. Available: https://doi.org/10.1145/3183713.3183759

[30] https://www.gurobi.com/, Gurobi.

[31] Y. Zhai, H. Xu, H. Wang, Z. Meng, and H. Huang, "Joint routing and sketch configuration in software-defined networking," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2092–2105, 2020.

[32] R. Miao, F. Dong, Y. Zhao, Y. Zhao, Y. Wu, K. Yang, T. Yang, and B. Cui, "Sketchconf: A framework for automatic sketch configuration," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 2022–2035.

[33] https://catalog.caida.org/dataset/passive_2018_pcap., Anonymized Internet Traces 2018.

[34] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 63–74. [Online]. Available: https://doi.org/10.1145/1402958.1402967