Distributed Sketch Deployment for Software Switches

Kejun Guo, Fuliang Li, *Member, IEEE,* Jiaxing Shen, *Member, IEEE,* Xingwei Wang, *Member, IEEE* and Jiannong Cao, *Fellow, IEEE*

***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. In software switches, centralized deployment provides low resource usage but suffers from poor load balancing. In contrast, collaborative measurement achieves load balancing through flow distribution across software switches but requires high resource usage. This paper presents a novel 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 software switches. Second, through a novel collaborative strategy, our framework achieves finer-grained flow distribution and further optimizes load balancing. Third, we further optimize load balancing by eliminating the mutual influence among forwarding paths. We evaluate the proposed framework on various network topologies and different sketches. 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 solutions.**

***Index Terms*—software switches, 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, congestion control, and anomaly detection [2]–[5]. 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 [6]–[10], and centralized deployment [11], [12].

A preliminary version of this work was presented at the IEEE International Conference on Computer Communications (INFOCOM'24) [1], May 2024.

Kejun Guo, Fuliang Li and Xingwei Wang are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China. E-mail: kejunguo@163.com, lifuliang@cse.neu.edu.cn, wangxw@mail.neu.edu.cn.

Jiaxing Shen is with the School of Department of the School of Computing and Decision Sciences, Lingnan University, Hong Kong, China. E-mail: jiaxingshen@LN.edu.hk.

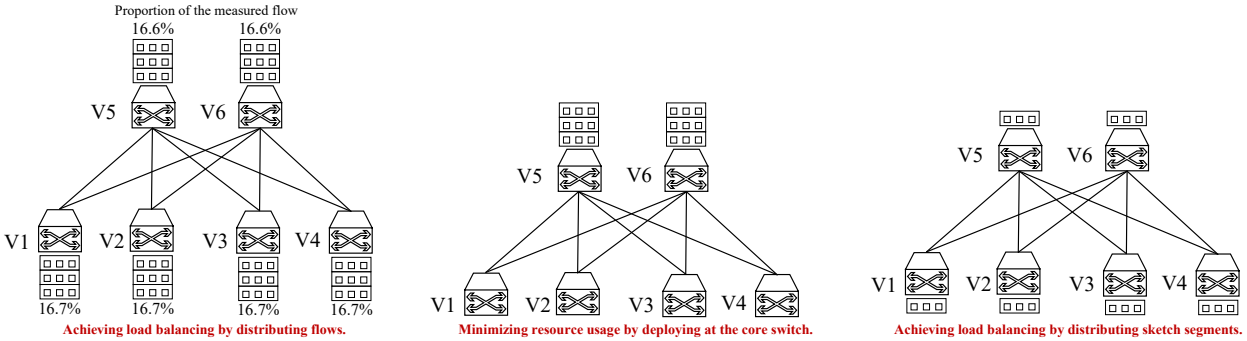
Jiannong Cao is with the School of Department of Computing, Hong Kong Polytechnic University, Hong Kong, E-mail: csjcao@comp.polyu.edu.hk.

As shown in Fig. 1a, collaborative measurement involves each switch in the network measuring a subset of flows to achieve 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 load balancing, nearly all switches deploy sketches to participate in the measurement process. In programmable switches like Tofino [13], 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 multiple sketches are required to support various tasks.
- **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 forwarding path jointly perform the sketch operation. Collaborative measurement divides traffic by flow granularity, causing severe load imbalance between switches handling elephant flow versus mouse flow. Per-packet load balancing is thus crucial, as it distributes multiple hash computations and memory accesses for each packet evenly across multiple switches. This reduces the impact of varying flow sizes, thereby further optimizing load balancing.

As shown in Fig. 1b, 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 to the following three 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



(a) Collaborative measurement. (b) Centralized deployment. (c) Distributed deployment.

Fig. 1. Comparison of three different sketch-based network-wide deployment solutions in leaf-spine network topology.

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	×	×	√

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 load balancing but entails high resource usage. In this paper, as shown in Fig. 1c, we propose to divide the existing sketch into segments and deploy them across multiple switches. It could strike a balance between resource usage and load balancing while ensuring per-packet load balancing. The previous distributed deployment solution most similar to ours is DISCO [14]. However, DISCO overlooks the interactions between forwarding paths in real network topologies and the finer-grained flow distribution, leading to severe switch load imbalance. And DISCO is limited to heavy hitter detection and certain sketches, lacking generalizability to other scenarios. Moreover, DISCO requires substantial domain expertise for deployment. Lastly, DISCO lacks rigorous theoretical analysis of feasibility. To summarize, the contributions of our proposed distributed sketch deployment are illustrated as follows.

Contribution I: lightweight distributed deployment framework for sketch. We introduce the 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 independent 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, load balancing, 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: collaborative distributed deployment framework for sketch. To address issues such as load imbalance in complex topologies, we propose the collaborative distributed deployment framework for sketch, which further optimizes the load balancing. Our collaborative strategy supports distributing a flow across multiple switches, whereas previous strategies allocated a flow to a single switch. Specifically, each switch maintains a value indicating the percentage of flows it needs to measure. Subsequently, flow distribution is performed using our collaborative strategy, which determines whether measurement should be performed on this switch based on hash value and values stored in the switch. This further optimizes load balancing.

Contribution III: optimal distributed deployment framework for sketch. To eliminate the mutual influence between forwarding paths, we propose the optimal distributed deployment framework for sketch, which further optimizes the load balancing. Unlike collaborative distributed deployment framework, within the switch, we maintain a separate percentage value for each forwarding path, thereby eliminating mutual influence between forwarding paths and achieving improved load balancing.

Contribution IV: 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 query checks whether a flow is present. A typical membership query sketch is Bloom Filter [15]. 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. Bloom Filter come with a unilateral error, resulting in false positives. In other words, a flow present in a Bloom Filter is certainly reported as true; however, a flow not present in the Bloom Filter may also result in false positives. Recently, variants of Bloom Filter have been proposed to meet the requirements of different applications such as CBF [16], DBF [17], EBF [18], and NBF [19].

2) Sketches for Frequency Estimation: Frequency estimation counts the number of packets in a flow. Typical frequency estimation sketches include CM sketch [20], CU sketch [21] and CO sketch [22]. 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. CO sketch and CM sketch are similar. The difference lies in how they handle insertion and query. When inserting a flow, for each mapped counter, CO sketch increments or decrements it probabilistically. When querying a flow, it checks the k mapped counters and returns the median of these values as the estimate. To improve memory utilization given skewed traffic, recent sketches split flows by size such as Tower sketch [23], Elastic sketch [24], One sketch [25], BitSense [26] and HeavyGuardain [27].

3) Sketches for Heavy Hitter Detection: Heavy hitter detection identifies flows exceeding a frequency threshold. A typical example is the MV sketch [28], comprising k rows of bucket arrays. The MV sketch uses the majority vote 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. There are also some other sketches that perform well, such as HeavyKeeper [29], Space Saving [30], and Unbiased Space Saving [31].

B. Sketch Deployment for Network-Wide Traffic Measurement

The existing sketch network-wide deployment solutions are mainly centralized deployment [11], [12], collaborative measurement [6]–[10], and distributed deployment solutions [14].

Centralized deployment is to deploy sketch at core switches, which can be obtained through historical traffic statistics or network topology. 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 suffer from extensive resource usage and per-packet load imbalance. Currently, the most typical collaborative measurement solution is NSPA [6] that maintains a sampling probability for each switch. If the packet is not metered by one of the preceding switches on the forwarding 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 [32] 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 [7], CFS [8], CountMax [9], and HiFi [10].

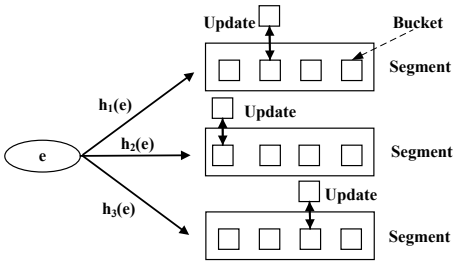


Fig. 2. An illustration of k-hash independent model sketch where $k = 3$.

DISCO [14] 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 the interactions between forwarding paths in real network topologies and the finer-grained flow distribution, 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 the lightweight distributed deployment framework for sketch, which only requires network operator to deploy the sketch segment on the specific switches. Firstly, we introduce a common model that many sketches use. Secondly, we give baseline solutions and our observations. Finally, based on this common model, we present the lightweight distributed deployment framework for sketch.

A. Sketch Model

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

Data structure: As shown in Fig. 2, the k-hash independent 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.

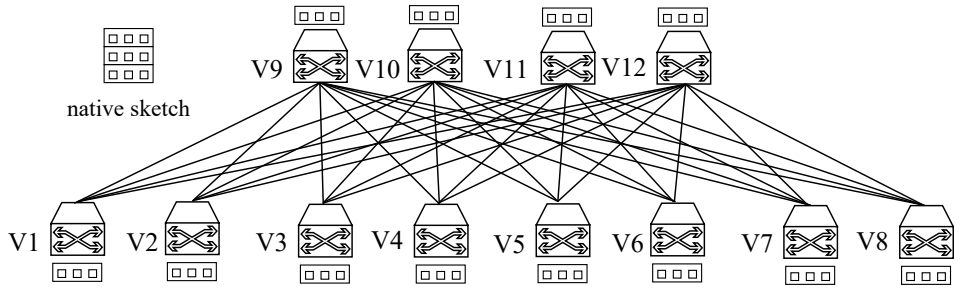


Fig. 3. A distributed sketch deployment solution in leaf-spine topology.

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

It should be noted that the sketches such as CU sketch [21] and Hashpipe [33] are not a k-hash independent model because 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. 3, it is a typical leaf-spine network topology consisting of 4 spine switches and 8 leaf switches. Each pair of leaf switches at the ends of each forwarding path send 24 flows with each other. Let's take the k-hash independent 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 independent model sketch to perform 2-3 hash calculations in the network. The resources of the programmable switch are allocated statically. In the k-hash independent 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 independent 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. We calculate the standard deviation of per-packet load balancing on every path and then take the average. For example, in the leaf-spine topology shown in Fig. 3, flows between leaf switches traverse two leaf switches and one spine switch. In centralized deployment, sketch is deployed at the spine switches. In a k-hash model sketch with $k = 3$, each flow will perform 0, 0, and 3 hash computations and memory accesses at the two leaf switches and one spine switch, respectively, resulting in a per-packet load standard deviation of 1.41. Table II gives a comparison of different solutions for measurement load and resource usage.

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	12	1901	1.41
NSPA	36	0	1.41
LDD	12	0	0

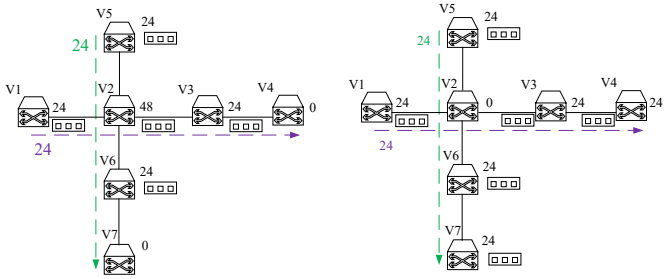
The first solution is centralized deployment (CD) [11], [12]. Sketch is deployed on the switches with the most concentrated traffic in the network. In leaf-spine topology, we deploy the complete sketch on all spine switches ($V9$-$V12$). In other words, a complete sketch of three segments is deployed 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 [6] as an example. The complete sketch of three segments is deployed on all switches ($V1$ - $V12$), 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 for sketch (LDD). It leverages the inherent parallelizability of k-hash independent 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 concern is that in previous sketch, the flow sets for each sketch segment are identical, whereas in our distributed deployment, the individual sketch segments may have different flow sets. In Section VII, we provide a mathematical proof that this difference in flow sets does not compromise the accuracy of the sketch.

C. Lightweight Distributed Deployment Framework for Sketch

Collaborative measurement only considers load balancing by dividing flows and ignores the possibility of load balancing



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

Fig. 4. Deployment satisfies Objective-I and II. The black numbers represent switch's measurement load. Solution-II that satisfies Objective-III is more load balanced.

by splitting sketch segments. Our proposed solution is based on two key facts: 1) k-hash independent sketches allow parallel computing since each segment 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: minimize 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. 3, Objective-I drives both the centralized deployment solution, where a complete sketch with three segments is deployed on spine switches, and the distributed deployment solution, where each switch deploys a single segment, into a solution set.

Objective-II: balance the number of sketch segments in each forwarding path. It ensures per-packet load balancing. In the above solutions, Objective-II assists in selecting the distributed deployment solution. 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 V.

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. 3, 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. 4, 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. 4b, because it is more load balanced.

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 the number of flows 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:

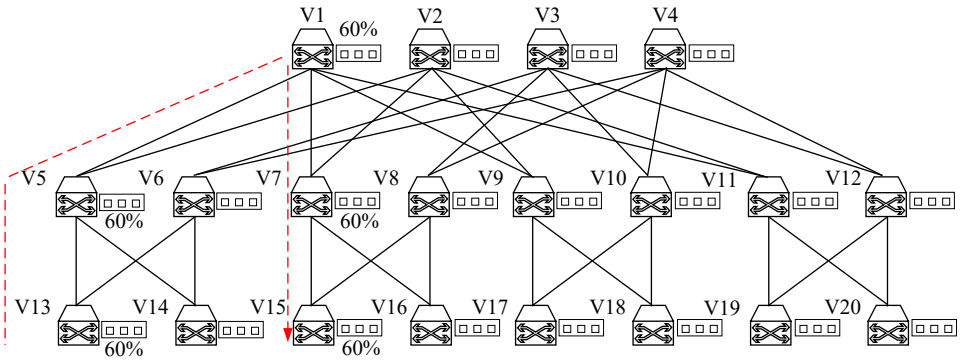
$$Opt. \begin{cases} min \sum\limits_{S_i^j \in \Gamma} \sum_{v \in P_{S_i^j}} D_v \\ min \sum\limits_{S_i^j \in \Gamma} \sigma\left(\{D_v\}, \forall v \in P_{S_i^j}\right) \\ min\ \sigma\left(\{L_v\}, \forall v \in V\right) \end{cases} \tag{1}$$

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

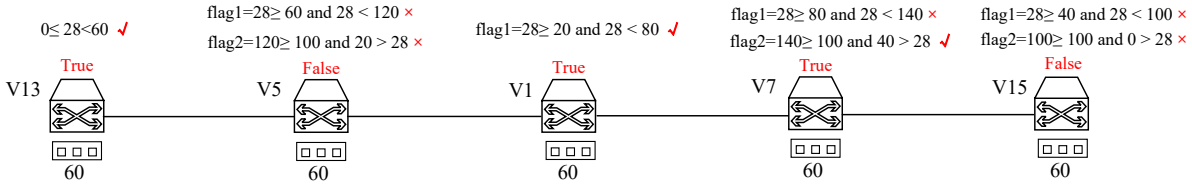
The first constraint in Eq. 2 restricts each forwarding path to have k or more sketch segments, which preserves the error bound of sketch. The three objective functions in Eq. 1 correspond to Objective-I, II and III, respectively. The priority of Objective-I is higher than that of Objective-II, and the priority of Objective-II is higher than that of Objective-III. We use the solver Gurobi [34] to solve.

Then, let's explain how many sketch segments are deployed in the switch and the memory size of each sketch. In switch v, the number of sketch segments to be deployed is D_v. The memory size of sketch is calculated by $\frac{L_v}{\sum_{v \in V} L_v} \times Memory$, and $Memory$ represents the total memory size of sketch across all switches, which is a configurable parameter set by the network operator.

Basic version unifies the benefits of centralized deployment and collaborative measurement, which realizes collaborative measurement's load balancing with resource efficiency approaching centralized deployment. Additionally, basic version uniquely achieve per-packet load balancing, with each packet measured collaboratively across multiple switches. By solely leveraging the distributed sketch, basic version provides a lightweight and balanced network measurement.



(a) The collaborative distributed sketch deployment solution in fat tree topology.



(b) An example of collaborative strategy in fat tree topology.

Fig. 5. The collaborative distributed sketch deployment framework for sketch in fat tree topology.

TABLE III
COMPARISON OF DIFFERENT SOLUTIONS IN FAT TREE TOPOLOGY.

Solutions	Resource Usage (Number of SALUs)	Load Standard Deviation	Per-Packet Load Standard Deviation
CD	12	1382	1.2
NSPA	60	0	1.2
LDD	12	564	0.49
MDD	20	0	0.49

IV. DISTRIBUTED SKETCH DEPLOYMENT FRAMEWORK: COLLABORATIVE VERSION

The lightweight distributed deployment framework for sketch performs well in leaf-spine topology. Unfortunately, the lightweight distributed deployment framework suffers from load imbalance in complex network topologies. In order to optimize load balancing performance, we present the collaborative distributed deployment framework for sketch.

A. Baseline Solutions and Our Observations

We employ a fat-tree topology with four pods to reveal the problems of the lightweight distributed deployment framework for sketch. As shown in Fig. 5a, between pods, each pair of edge switches at the ends of each forwarding path send 24 flows with each other. Each flow consists of a single packet that needs to perform three hash calculations and memory accesses ($k = 3$). As presented in Table III, centralized deployment and collaborative measurement will not be described.

The third solution is the lightweight distributed deployment framework for sketch (LDD). One deployment solution that satisfies LDD is to deploy a sketch segment on all core and aggregation switches ($V1$-$V12$). In this case, all edge switches remain unused, resulting in uneven load distribution.

The fourth solution is the collaborative distributed deployment framework for sketch (MDD). To differentiate from centralized deployment (CD), we abbreviate it as MDD instead of CDD. The collaborative distributed deployment framework deploys a sketch segment on each switch, with each switch individually performing measurements for 60% of the traffic, and a total of three measurements are conducted collectively by five switches along the path. Compared to the lightweight distributed deployment framework, the collaborative distributed deployment framework achieves better load balancing through finer-grained flow distribution.

B. Collaborative Distributed Deployment Framework for Sketch

Built upon the lightweight distributed deployment framework, the core concept of the collaborative distributed deployment framework lies in executing finer-grained flow distribution through a novel collaborative strategy, thereby enhancing improved load balancing. Previous collaborative measurement solutions allocate a specific flow to a single switch for measurement. Our collaborative strategy is distinct from previous collaborative strategy. Our collaborative strategy distributes specific flow across multiple switches, which previous collaborative solutions, such as NSPA [6], are unable to achieve.

Firstly, let us explain how to determine the percentage of flows that each switch needs to measure. To achieve finer-grained flow distribution, we relax D_v to float, which represents the percentage of the flows that switch v needs to measure, rather than the number of sketch segments mentioned in the Section III-C.

We further formulate the problem as follows:

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

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

Unlike the lightweight distributed deployment framework, the priority of Objective-III (network-wide measurement load balancing) is higher than that of Objective-II (balance the number of sketch segments in each forwarding path). Since we relax D_v to float, this will always result in an even distribution of the percentage if we do not change the priorities of the objectives, which makes it impossible to further optimize the measurement load balancing.

Then, let's explain how many sketch segments are deployed in the switch and the memory size of each sketch. In switch v, the number of sketch segments to be deployed is $\lceil D_v \rceil$. The memory size of sketch is calculated by $\frac{L_v}{\sum_{v \in V} L_v} \times Memory$, and $Memory$ represents the total memory size of sketch across all switches, which is a configurable parameter set by the network operator. Within each switch, the memory allocated to the sketch should be further divided among different sketch segments based on their respective loads.

Next, we explain our proposed collaborative strategy. First, in each switch, we store the percentage of flow that each sketch segment needs to measure in this switch, denoted as Seg_i. It should be noted that Seg_i is not D_v. For example, if D_v is 120% in switch v, this means switch v will be deployed with two sketch segments, with Seg_1 = 100% and Seg_2 = 20%. Since the programmable switch does not support float, we can simply multiply Seg_i by a scaling factor num ($num \in N+$) to convert Seg_i into an integer. For example, when num is 100 and Seg_i is 60%, $Seg_i \times num$ in the programmable switch is stored as 60. If higher precision is desired, you can set num to a larger positive integer. Second, we explain how to ensure that each switch measures the corresponding percentage of flow. For each packet e, the hash value range is [0, num). At the ingress switch v_i, v_i performs a hash computation on packet e and denotes it as $hash(e)$. Then, the first sketch segment checks if

$$\begin{cases} ceil = Seg_1 \times num \\ 0 \leq hash(e) < ceil \end{cases} \tag{5}$$

If yes, the packet e is inserted into the first sketch segment. On sketch segments other than the first sketch segment, the sketch segment checks if

$$\begin{cases} cur\_ceil = ceil + Seg_i \times num \\ flag1 = (hash(e) \geq ceil\ and\ hash(e) < cur\_ceil) \\ flag2 = (cur\_ceil \geq num\ and\ cur\_ceil - num > hash(e)) \\ flag1\ or\ falg2 == True \end{cases} \tag{6}$$

If yes, the packet e is inserted into the corresponding sketch segment. When each sketch segment check is completed, if cur_ceil is greater than or equal to num, we update $ceil = cur\_ceil - num$; otherwise, we update $ceil = cur\_ceil$. Finally, when all sketch segments in the ingress switch have checked, we insert the $hash(e)$ and $ceil$ into the packet header and forward it to the next switch. Alternatively, instead of inserting $hash(e)$ into the packet header, each switch can maintain an identical hash function, which still results in significantly lower overhead compared to collaborative measurement. The sketch segments in subsequent switches also execute the same operation as described in Eq. 6. The only difference is that we need to extract $hash(e)$ and $ceil$ from the header of packet e. Overall, the proposed collaborative strategy incurs a certain amount of bandwidth overhead, which is minimal as it requires the transmission of only two additional values. Moreover, the collaborative strategy relies solely on the uniformity of the employed hash function. Fortunately, most modern hash functions can achieve a relatively uniform distribution, mitigating this potential concern.

For example, in the fat-tree topology shown in Fig. 5a, according to the solution of Eq. 3 and 4, each switch deploys a sketch segment to measure 60% of the traffic. We assume $num = 100$. And let's consider the flow with a hash value of 28 along the path $V13-V5-V1-V7-V15$ as an example. As shown in Fig. 5b, for packet e, the sketch segment in switch $V13$ is first encountered. Therefore, we perform a hash calculation in switch $V13$ and then check if packet e is inserted into this sketch segment according to Eq. 5. Subsequently, we insert $hash(e)$ and the $ceil$ into the header of packet e. Since the sketch segment in subsequent switches is not the first one encountered by packet e, we check if packet e is inserted into the corresponding sketch segment according to Eq. 6. The detailed calculation process is illustrated in Fig. 5b.

Finally, we elaborate on some of the issues encountered in deploying programmable switches. On one hand, due to the hash value range of programmable switches being from 0 to a power of 2, similarly, we can conveniently set num to a power of 2. On the other hand, the ingress switch does not require additional hash units. We can utilize the hash computation results calculated during sketch insertion for shifting to serve as $hash(e)$ ($0 \leq hash(e) < num$).

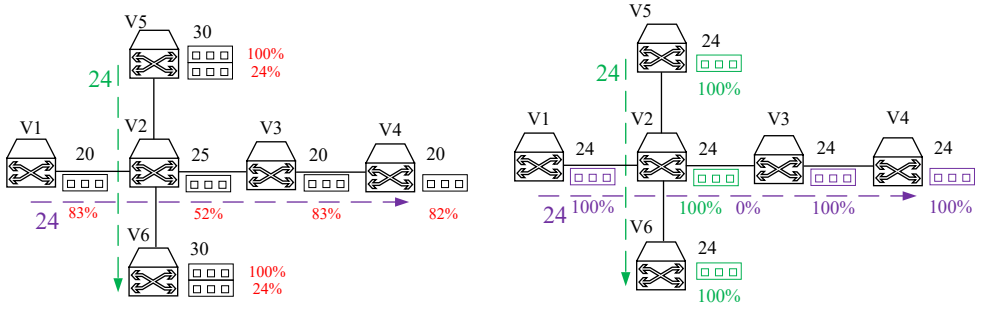
In summary, collaborative version achieves finer-grained flow distribution through our collaborative strategy, achieving better load balancing effects at the expense of consuming minimal bandwidth resources.

V. Distributed Sketch Deployment Framework: Optimal Version

The collaborative distributed deployment frameworks for sketch perform well in most network topologies. However, it does not account for interactions between forwarding paths. To optimize load balancing, we propose the optimal distributed deployment framework for sketch.

A. Baseline Solutions and Our Observations

We use a simple network topology to reveal the problems of the collaborative distributed deployment framework for sketch.



(a) The collaborative distributed deployment solution. (b) The optimal distributed deployment solution.

Fig. 6. Comparison of distributed deployment frameworks. The black numbers represent switch's measurement load. The red numbers represent the percentage of flow that the corresponding sketch segment needs to measure. The green and purple numbers represent the percentage of flow that the sketch segment of the corresponding colors needs to measure.

TABLE IV
COMPARISON OF DIFFERENT SOLUTIONS.

Solutions	Resource Usage (Number of SALUs)	Load Standard Deviation	Per-Packet Load Standard Deviation
CD	3	53.7	1.36
NSPA	18	3.6	1.36
LDD	5	13.9	0.22
MDD	8	4.5	0.41
ODD	6	0	0.22

In the topology shown in Fig. 6, there are two paths, each flow consists of a single packet that needs to perform three hash calculations and memory accesses ($k = 3$). As presented in Table IV, centralized deployment, collaborative measurement and the lightweight distributed deployment framework will not be described in detail.

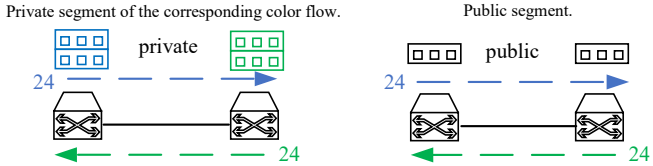
The fourth solution is the collaborative distributed deployment framework (MDD). As shown in Fig. 6a, since switch $V2$ serves as the convergence switch for path flows S_1^4 and S_5^6, its D_v needs to simultaneously accommodate both S_1^4 and S_5^6. This makes it challenging to achieve optimal load balancing. We refer to this phenomenon as the mutual influence between forwarding paths.

The fifth solution is the optimal distributed deployment framework (ODD). As shown in Fig. 6b, each switch is deployed a sketch segment. Subsequently, S_1^4 is inserted into switches $V1$, $V3$, and $V4$, while S_5^6 is inserted into switches $V5$, $V2$, and $V6$. It is evident that the optimal distributed deployment framework eliminates the mutual influence between forwarding paths, allowing each path flow to have its own unique D_v at each switch, thereby achieving better load balancing.

B. Optimal distributed Deployment Framework for Sketch

In order to eliminate the mutual influence between forwarding paths, we need to redefine D_v. In the collaborative distributed deployment framework, D_v is prepared for all flows passing through the switch v. We modify D_v to $D_v^{S_i^j}$, which represents the percentage of the certain path flow S_i^j that switch v needs to measure. We do not make any changes to other goals and constraints.

We formulate the problem as follows:



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

Fig. 7. Deployment satisfies Objective-I and III in MDD. Solution-II that satisfies Objective-II has the lowest resource usage.

$$Opt. \begin{cases} min \sum\limits_{S_i^j \in \Gamma} \sum_{v \in P_{S_i^j}} D_v^{S_i^j} \\ min\ \sigma\left(\{L_v\}, \forall v \in V\right) \\ min \sum\limits_{S_i^j \in \Gamma} \sigma\left(\left\{D_v^{S_i^j}\right\}, \forall v \in P_{S_i^j}\right) \end{cases} \tag{7}$$

$$S.t. \begin{cases} \sum\limits_{v \in P_{S_i^j}} D_v^{S_i^j} \geq k \quad \forall S_i^j \in \Gamma \\ L_v = \sum\limits_{S_i^j \in \Gamma} \left(D_v^{S_i^j} \times S_i^j,\ if\ v \in P_{S_i^j}\right) \quad \forall v \in V \\ D_v^{S_i^j} \in [0, k], Float \quad \forall v \in V, S_i^j \in \Gamma \end{cases} \tag{8}$$

In Eq. 7, Objective-II (balance the number of sketch segments in each forwarding path) 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. 7 ($k = 2$ 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.

Next, we will elaborate on the implementation of the optimal distributed deployment framework for sketch. Similar to the collaborative distributed deployment framework, the collaborative strategy for optimal distributed deployment is fundamentally the same as the one proposed in collaborative

distributed deployment, except that now a unique $D_v^{S_i^j}$ is maintained for each path flow. As a result, the efficacy of this approach still primarily depends on the uniformity of the employed hash function. A switch v only needs to maintain $O(|P_v|)$ match action entries, where P_v is the set of the concurrent active paths passing v. Therefore, the optimal distributed deployment framework for sketch necessitates the consumption of both bandwidth and TCAM resources. The bandwidth usage is similar to that in the collaborative distributed deployment framework for sketch. The TCAM resources can be easily accommodated by programmable switches, as each switch v only needs to maintain $O(|P_v|)$ entries.

Finally, let's explain how many sketch segments are deployed in the switch and the size of each sketch segment. The number of sketch segments that each switch v needs to deploy is $\left\lceil max\left(\left\{D_v^{S_i^j}\right\}, \forall S_i^j \in \Gamma\right)\right\rceil$. The memory size of sketch is calculated by $\frac{L_v}{\sum_{v\in V} L_v} \times Memory$, and $Memory$ represents the total memory size of sketch across all switches, which is a configurable parameter set by the network operator. Within each switch, the memory allocated to the sketch should be further divided among different sketch segments based on their respective loads.

In summary, optimal version eliminates the mutual influence between forwarding paths, achieving better measurement load balancing at the cost of consuming certain resource.

VI. Discussion

We discuss the generality of the proposed framework and some challenges encountered in practical implementation. First, the proposed framework is designed to be general and applicable to networks composed of programmable devices. It can directly derive deployment solutions based on the network topology and traffic matrix, making it suitable for both ISP and data center networks. While traditional network devices may not be compatible due to a lack of programmability, the gradual replacement of existing equipment with programmable devices mitigates this limitation. Second, the computational overhead required for deployment is low, equivalent to the computational cost of a sketch or, at most, one additional hash calculation. Finally, while both collaborative distributed deployment and optimal distributed deployment incur some bandwidth overhead, it remains minimal, requiring only two additional numbers to be stored in the packet header.

Next, from the perspective of network operator, who are also the end-users, we discuss the advantages of distributed deployment. First, distributed deployment achieves low resource usage and load balancing. Second, the accuracy of sketch is not compromised by distributed deployment, which will be theoretically proven in Section VII. Finally, since sketch is distributed across various nodes, distributed deployment offers enhanced robustness. For example, consider a flow traversing three switches along a path. Even if the bandwidth between one switch and the controller is constrained, sketch segments from the other two switches can still be transmitted to the controller in real time, avoiding the issue faced by centralized deployment or collaborative measurement where incomplete sketches may prevent some traffic measurement information from being reported.

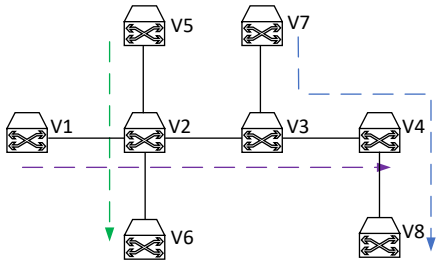


Fig. 8. Synthetic topology. It contains 8 switches and three different forwarding paths.

VII. 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 [35] 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 [20] 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.

VIII. Experimental results

We apply the distributed deployment framework for sketch to three topologies (Fat tree topology, Leaf-spine topology, and Synthetic topology) and four sketches (Bloom Filter [15], CM sketch [20], CO sketch [22] and MV sketch [28]).

A. Test Setup

We use anonymous IP tracking collected from CAIDA in 2018 [36], which has been used extensively in previous

articles. 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, as shown in Fig. 5a, which consists of 4 core switches, 8 aggregation switches, and 8 edge switches. The second topology is leaf-spine network topology, as shown in Fig. 3, which contains 4 spine switches and 8 leaf switches. The third topology is a synthetic network topology, as shown in Fig. 8, which has three forwarding paths. It should be noted that the experimental results are similar in larger topologies. We evenly distribute all flows across each path. We conduct experiments using five solutions. The first solution is the lightweight distributed deployment framework for sketch (LDD). The second solution is the collaborative distributed deployment framework for sketch (MDD). The third solution is the optimal distributed deployment framework for sketch (ODD). The fourth solution is centralized deployment (CD) [11], [12]. The fifth solution is NSPA [6]. The code for these five solutions is available at [37]. 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.

We compare the following metrics across three network topologies and four sketches.

- Total Resource Usage: the total usage of SALUs, hash units, or SRAM resources in all programmable switches across the network. For the sake of comparison, we use the SALUs resource usage as a representative measure for the overall resource usage.
- Resource Usage Standard Deviation: the standard deviation of switch resource usage across the network. We use the standard deviation of resource usage to assess the balance of resource usage among switches across the network.
- Load Standard Deviation: the standard deviation of measurement load among switches across the network. 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 load standard deviation to evaluate whether the measurement load across switches is balanced.
- Per-Packet Load Standard Deviation: the standard deviation of the number of hash calculations or memory accesses, performed by each packet on each switch in its forwarding path. We calculate the standard deviation of per-packet load balancing on every path and then take the average. We use per-packet load standard deviation to assess whether each packet evenly distributes its measurement load among the switches along the forwarding path.
- False Positive Rate (FPR): $\frac{n}{m}$, m represents the total number of flows that do not appear in the time period, and n represents the number of flows that are mistaken for flows that appeared in the time period.
- Average Relative Error (ARE): $\frac{1}{n}\sum_{i=1}^{n}\frac{\left|f_i-\hat{f}_i\right|}{f_i}$, where n is the number of flows, and f_i and $\hat{f}_i$ are the actual and estimated flow sizes respectively.
- F_1 Score: $\frac{2\cdot RR\cdot PR}{RR+PR}$, where PR (Precision Rate) refers to the ratio of the number of the correctly reported instances to the number of all reported instances, and RR (Recall Rate) refers to the ratio of the number of the correctly reported instances to the number of all correct instances.

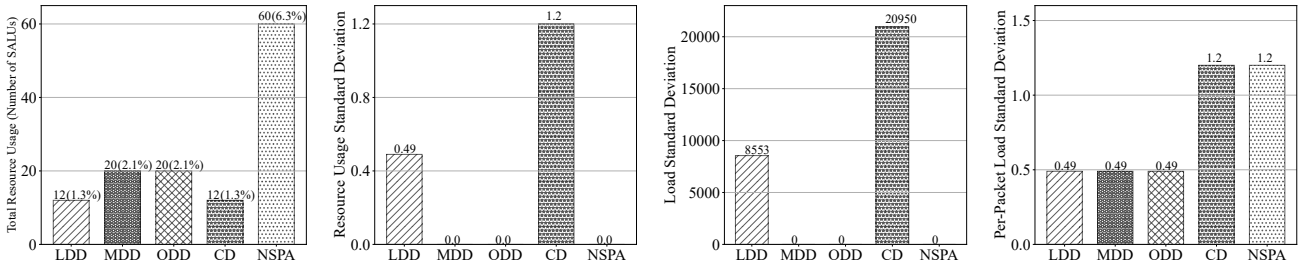
B. Experimental Results on Resource Usage and Load Balancing

1) Experimental Setup: We apply the above five solutions in three network topologies. We set $k = 3$ for the k-hash independent sketch. As mentioned before, previous practice has proved that it is good enough for the k-hash independent model sketch to perform 2-3 hash calculations in the network. The results of CM sketch are as follows. Since most k-hash independent model sketches are k hash calculations and memory accesses, the results of other k-hash independent model sketches are also similar.

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

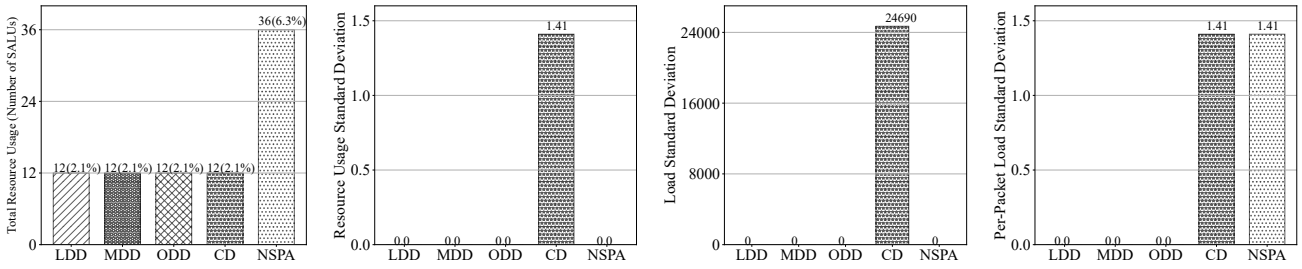
Resource usage: As shown in Fig. 9a, Fig. 10a and Fig. 11a, the percentage in the figure indicates the ratio of the number of SALUs used to the number of SALUs of all switches in the topology. We find that the lightweight distributed deployment framework and centralized deployment has the lowest resource usage. And as shown in Fig. 9b, Fig. 10b and Fig. 11b, the resource allocation of the lightweight distributed deployment framework is more balanced than the centralized deployment. The resource usage of the collaborative distributed deployment framework and the optimal 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 collaborative distributed deployment framework resource usage is only 0.33, 0.33, and 0.46 of the NSPA resource usage. The optimal distributed deployment framework resource usage is only 0.33, 0.33, and 0.54 of the NSPA resource usage.

Measurement load balancing: As shown in Fig. 9c, Fig. 10c and Fig. 11c, we find that the collaborative distributed deployment framework, the optimal distributed deployment framework and NSPA have the good load balancing effect, and the collaborative distributed deployment framework and the optimal distributed deployment framework achieve better load balancing effect than NSPA. The lightweight distributed deployment framework is slightly worse than the load-balancing effect of them. 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 the collaborative distributed deployment



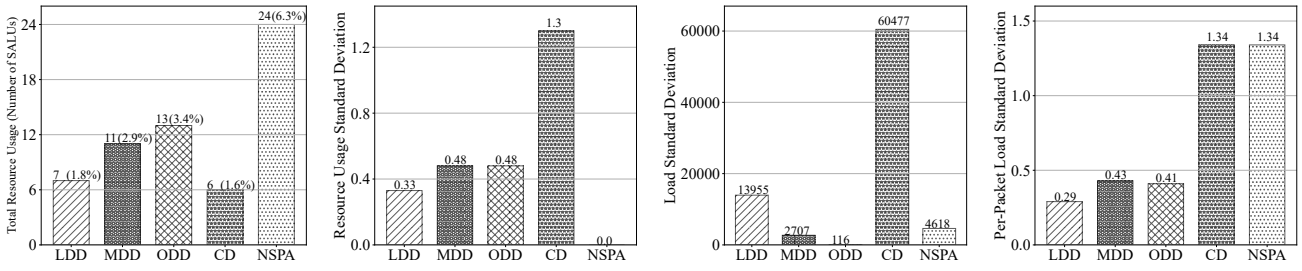
(a) Resource usage. (b) Resource usage load balancing. (c) Measurement load balancing. (d) Per-packet load balancing.

Fig. 9. 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. 10. 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.

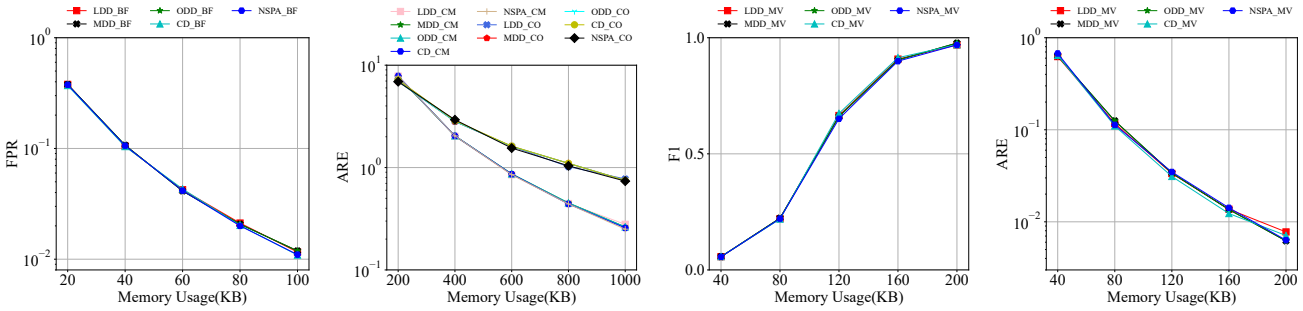
Fig. 11. Resource usage, measurement load balancing and per-packet load balancing on synthetic topology.

framework, the optimal distributed deployment framework and NSPA are all 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 collaborative distributed deployment framework and the optimal distributed deployment framework achieve better load balancing than NSPA. The lightweight distributed deployment framework, the collaborative distributed deployment framework, the optimal distributed deployment framework, and NSPA are 0.23, 0.045, 0.002 and 0.076 of the standard deviation of the measurement load of the centralized deployment, respectively.

Per-packet load balancing: As shown in Fig. 9d, Fig. 10d and Fig. 11d, we find that the per-packet load standard deviation in the switch of the lightweight distributed deployment framework, the collaborative distributed deployment framework and the optimal distributed deployment framework is only 0.41 of the centralized deployment and NSPA in fat-tree topology. And the per-packet load standard deviation in the switch of the lightweight distributed deployment framework, the collaborative distributed deployment framework and the optimal distributed deployment framework are 0.22, 0.32 and 0.31 of the centralized deployment and NSPA in the synthetic topology, respectively. 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.

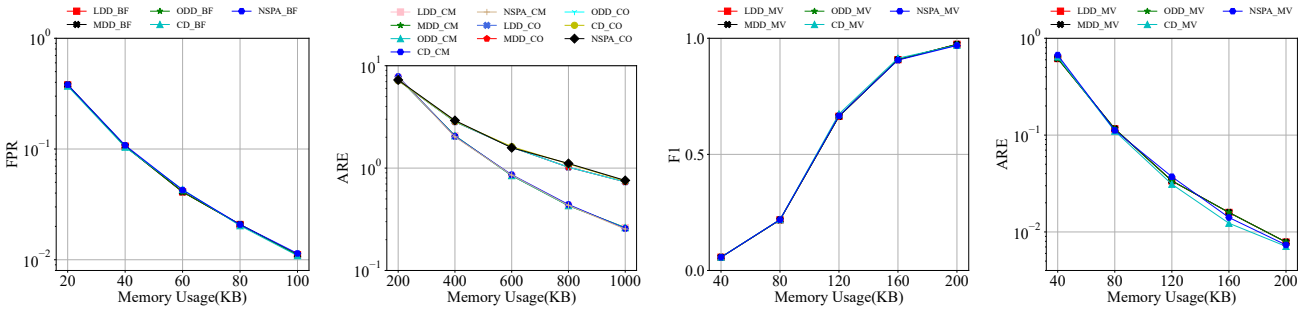
Analysis: Some argue that reserving the full resources required for a sketch on a switch, without splitting, does not result in high resource usage. However, it is important to emphasize that, as demonstrated in Sketchovsky [38], multiple sketches are often needed in the network to meet different tasks. This makes the switch's resources a bottleneck, unable to accommodate multiple sketches. Our distributed deployment solution alleviates this bottleneck.

In summary, the distributed deployment frameworks combine the advantages of centralized deployment and collab-



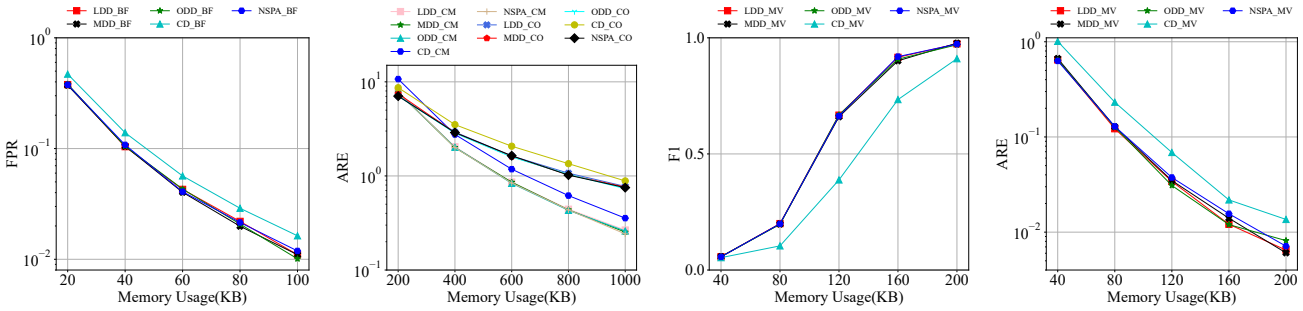
(a) Membership query. (b) Frequency estimation. (c) Heavy hitter detection. (d) Heavy hitter detection.

Fig. 12. Accuracy of membership query, frequency estimation, and heavy hitter detection on fat-tree topology.



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

Fig. 13. Accuracy of membership query, frequency estimation, and heavy hitter detection on spine-leaf topology.



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

Fig. 14. Accuracy of membership query, frequency estimation, and heavy hitter detection on synthetic topology.

orative measurement, which achieve optimal load balancing effects with low resource usage. Compared with centralized deployment, they achieve measurement load balancing, and resource usage is also more balanced. Compared with the state-of-the-art collaborative measurement solution NSPA, they only use 0.2-0.54 of NSPA resource usage to achieve better load balancing effect. At the same time, our distributed deployment frameworks achieve per-packet load balancing. Multiple hash calculations and memory accesses for each packet are evenly distributed to multiple switches, which can reduce the impact of varying flow sizes.

C. Experimental Results on Accuracy

1) Experimental Setup: We apply the above five 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 independent model sketches, we set $k = 3$. We set the heavy hitter threshold to 0.01% of the number of packets.

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

Membership query: As shown in Fig. 12a, Fig. 13a and Fig. 14a, in both fat-tree and leaf-spine topologies, we find that the FPR of these five 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. 12b, Fig. 13b and Fig. 14b, in both fat-tree and leaf-spine topologies, we find that the ARE of these five 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. 12c, Fig. 12d, Fig. 13c, Fig. 13d, Fig. 14c, and Fig. 14d, in both fat-tree and leaf-spine topologies, we find that the F_1 and ARE of five solutions are almost equal as the memory varies. In synthetic topologies, centralized deployment performs worse due to redundant measurement.

Analysis: 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 in Section VII.

IX. Conclusion

This paper proposes three distributed deployment frameworks for sketch that support lightweight, collaborative or optimal 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 apply 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.

X. Limitations and future work

First, our proposed three distributed deployment frameworks struggle to strike a balance between resource usage and load balancing. The lightweight distributed deployment framework achieves the lowest resource usage, but it also exhibits the poorest load balancing performance. The collaborative distributed deployment framework achieves good load balancing, but inevitably incurs some bandwidth overhead to enable finer-grained flow distribution. The optimal distributed deployment framework achieves the best load balancing performance, but it requires both bandwidth and TCAM resources. Therefore, in the future, we will continue to explore how to achieve a better balance between resource usage and load balancing.

Second, we only address the network-wide deployment problem of a single sketch. In practice, it is often necessary to deploy multiple sketches simultaneously to fulfill different tasks. In the future, we will explore the network-wide deployment problem of multiple sketches.

Third, we only address the network-wide deployment problem of k-hash independent model sketches, while neglecting the network-wide deployment problem of non-k-hash independent model sketches. In the future, we will simultaneously consider the network-wide deployment problem of both k-hash independent and non-k-hash independent model sketches.

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] F. Li, K. Guo, J. Shen, and X. Wang, "Effective network-wide traffic measurement: A lightweight distributed sketch deployment," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 181–190.

[2] H. Sun, Q. Huang, J. Sun, W. Wang, J. Li, F. Li, Y. Bao, X. Yao, and G. Zhang, "Autosketch: Automatic sketch-oriented compiler for query-driven network telemetry," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1551–1572.

[3] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 113–126.

[4] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hpcc: High precision congestion control," in *Proceedings of the ACM special interest group on data communication*, 2019, pp. 44–58.

[5] K. Guo, F. Li, J. Shen, and X. Wang, "Advancing sketch-based network measurement: A general, fine-grained, bit-adaptive sliding window framework," in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2024, pp. 1–10.

[6] 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*. IEEE, 2019, pp. 1108–1116.

[7] 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*, 2014, pp. 85–90.

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

[9] X. Yu, H. Xu, D. Yao, H. Wang, and L. Huang, "Countmax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2774–2786, 2018.

[10] 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*. IEEE, 2020, pp. 2341–2350.

[11] 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.

[12] 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)*. IEEE, 2020, pp. 1–7.

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

[14] 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*, 2020, pp. 536–537.

[15] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[16] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM transactions on networking*, vol. 8, no. 3, pp. 281–293, 2000.

[17] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and J. L. Larriba-Pey, "Dynamic count filters," *Acm Sigmod Record*, vol. 35, no. 1, pp. 26–32, 2006.

[18] 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, 2021.

[19] 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, 2022.

[20] 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.

[21] 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*, 2002, pp. 323–336.

[22] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.

[23] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang *et al.*, "Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 991–1010.

[24] 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*, 2018, pp. 561–575.

[25] Z. Fan, R. Wang, Y. Cai, R. Zhang, T. Yang, Y. Wu, B. Cui, and S. Uhlig, "Onesketch: A generic and accurate sketch for data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 887–12 901, 2023.

[26] 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*, 2023, pp. 220–238.

[27] T. Yang, J. Gong, H. Zhang, L. Zou, L. Shi, and X. Li, "Heavyguardian: Separate and guard hot items in data streams," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2584–2593.

[28] L. Tang, Q. Huang, and P. P. 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*. IEEE, 2019, pp. 2026–2034.

[29] 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 Transactions on Networking*, vol. 27, no. 5, pp. 1845–1858, 2019.

[30] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International conference on database theory*. Springer, 2005, pp. 398–412.

[31] D. Ting, "Data sketches for disaggregated subset sum and frequent item estimation," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1129–1140.

[32] 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*, vol. 32, no. 1, pp. 411–426, 2023.

[33] 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*, 2017, pp. 164–176.

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

[35] 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)*. IEEE, 2023, pp. 2022–2035.

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

[37] https://github.com/QingYeyyds/Distributed-Sketch-Deployment.git.

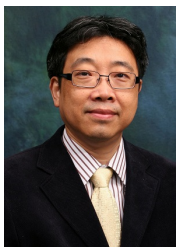
[38] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "Sketchovsky: Enabling ensembles of sketches on programmable switches," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1273–1292.

Kejun Guo received the B.Sc. degree in computer science from Northeastern University, China in 2023. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Northeastern University, China. His research interests include network measurement, data stream interests, and distributed training acceleration.

Fuliang Li (Member, IEEE) received the B.Sc. degree in computer science from Northeastern University, China, in 2009, and the Ph.D. degree in computer science from Tsinghua University, China, in 2015. He is currently an Professor with the School of Computer Science and Engineering, Northeastern University. He has published more than 50 journals/conference papers. His research interests include network management and measurement, cloud computing, and network security.

Jiaxing Shen (Member, IEEE) is an Assistant Professor with the Department of Computing and Decision Sciences at Lingnan University. He received the B.E. degree in Software Engineering from Jilin University in 2014, and the Ph.D. degree in Computer Science from the Hong Kong Polytechnic University in 2019. He was a visiting scholar at the Media Lab, Massachusetts Institute of Technology in 2017. His research interests include mobile computing, data mining, and IoT systems. His research has been published in top-tier journals such as IEEE TMC, ACM TOIS, ACM IMWUT, and IEEE TKDE. He was awarded conference best paper twice including one from IEEE INFOCOM 2020.

Xingwei Wang (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from Northeastern University in 1989, 1992, and 1998, respectively. He is currently a Professor with the School of Computer Science and Engineering, Northeastern University. He has published more than 100 journal articles, books, book chapters, and refereed conference papers. His research interests include cloud computing, future internet, and others. He has received several best paper awards.



Jiannong Cao (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in computer science from Washington State University, Pullman, WA, USA, in 1986 and 1990, respectively. He is currently a Chair Professor with the Department of Computing, The Hong Kong Polytechnic University (PolyU), Hong Kong. He is also the Dean of Graduate School, the Director of Research Institute of Artificial Intelligent of Things, and the Internet and Mobile Computing Lab, and the Vice Director of the University's Research Facility in Big Data Analytics, PolyU. He has coauthored five books, coedited nine books, and authored or coauthored over 500 papers in major international journals and conference proceedings. His research interests include distributed systems and blockchain, wireless sensing and networking, Big Data and machine learning, and mobile cloud and edge computing.