Per-flow Network Measurement with Distributed Sketch

Liyuan Gu, Ye Tian, *Member, IEEE,* Wei Chen, Zhongxiang Wei,
Cenman Wang, Xinming Zhang, *Senior Member, IEEE*

***Abstract*—Sketch-based method has emerged as a promising direction for per-flow measurement in data center networks. Usually in such a measurement system, a sketch data structure is placed as a whole at one switch for counting all passing packets, but when summarizing measurement results from multiple switches, the overall accuracy is generally constrained by a few individual switches with small-sized sketches due to their limited memory resources. To address this problem, in this paper, we present *Distributed Sketch*, a new method for per-flow network measurement in data center networks. In Distributed Sketch, each network path is associated with a logical sketch, whose data structure is collectively maintained by all the switches along the path; meanwhile, each switch multiplexes its physical sketch to the constructions of the logical sketches of all the paths it belongs to. With Distributed Sketch, switches collaborate to measure network flows, and the network-wide measurement workload is fairly distributed among all the switches in the network. We implement Distributed Sketch with P4 on commodity hardware programmable switch, and in particular, to overcome the limitation that hardware switches do not support float-point computation, we present an optimal approximation method that involves only integer operations. We also propose an In-band Network Telemetry (INT) based method for addressing the challenges in deploying Distributed Sketch in large-scale data centers. Experiment results and theoretical analysis show that our proposed method is lightweight regarding measurement overhead, and by aggregating and making fair uses of resources from all the switches in the network, Distributed Sketch achieves a higher measurement accuracy compared with the state-of-the-art solutions.**

***Index Terms*—Per-flow network measurement, distributed sketch, programmable switch, network collaboration**

I. Introduction

PER-flow measurement, which aims to estimate the size of each flow in the network, has a wide range of applications in data center networks. Examples include counting active flows [1], finding heavy hitters [2], [3], identifying persistent flows [4], and detecting heavy changes [5], etc.

In recent years, sketch-based method has become a promising direction for per-flow measurement [6]–[16]. In such a system, a probabilistic data structure, namely *sketch*, is maintained by switch for aggregating per-flow statistics. Existing sketches could be roughly divided into two categories: classical sketches and recently proposed advanced sketches. Classical sketches such as count-min (CM) [17], CU [18], and Count [19] associate multiple bucket arrays with hash functions. When receiving a packet, a switch computes hash values from the packet's flow ID, locates a bucket from each sketch array and updates the bucket value. Although easy to implement, however, a classical sketch typically requires a large memory for ensuring its estimating accuracy, as all its buckets are equal-sized and must be large enough for accommodating an elephant flow.

On the other hand, traffic in real-world networks is composed of a few elephant flows and a large amount of mice flows [20], [21]. By exploiting such a fact, a number of advanced sketches have been proposed in recent years [10]–[16], [22]. Typically, such a sketch uses multiple layers of basic sketch structures for counting flows of different sizes with different-sized counters. For example, TowerSketch [15] uses different-sized counters in different arrays under the property that bits used in each array stays the same; the Elastic sketch [10] employs a hash table and a CM sketch to measure elephant and mice flows respectively, and differentiates the flows with a voting-based mechanism. For both classical and advanced sketches, the larger a sketch is, the more accurate it can estimate with fewer hash collisions. Unfortunately, a commodity hardware programmable switch (e.g., the Barefoot Tofino switch [23]) usually has limited SRAM memory varying from a few kB to dozens of MB, which prohibits it to host a large sketch.

Most existing measurement systems place a sketch data structure, which could be either a classical or an advanced sketch, as a whole at one switch for counting all passing packets, and use multiple switches at strategic positions to collectively measure the entire network's traffic. To recover errors from measurement data, a number of control plane algorithms have been proposed [22], [24]–[26]. Nevertheless, with a few switches hosting small-sized sketches due to their limited memory resources, the overall measurement accuracy is constrained, and the control plane algorithms can only mitigate such inaccuracy. Moreover, since switches work independently, packets passing multiple switches would be counted repeatedly, and considerable measurement workload is redundant.

In this paper, we present *Distributed Sketch*, a new method for per-flow network measurement. Rather than placing a sketch as a whole at one switch, in Distributed Sketch, each network path is associated with a logical sketch, whose data structure is collectively maintained by all the switches along the path. Meanwhile, a switch that is crossed by multiple paths multiplexes its physical sketch to participate in the constructions of the logical sketches for all these paths. With Distributed Sketch, switches collaborate to measure network flows, and the network-wide measurement workload is fairly

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 61672486 and 62072425 and in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDC02070300. *(Corresponding author: Ye Tian (yetian@ustc.edu.cn).)*

The authors are with the Anhui Key Laboratory on High-Performance Computing, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230026. (email: {guliyuan1, szcw33, wz199758, wangcenman}@mail.ustc.edu.cn, {yetian, xinming}@ustc.edu.cn)

distributed among all the switches according to their memory resources, topological positions, as well as their current workloads.

Distributed Sketch has the following merits: First, it improves utilizations of the scarce in-switch memory resources, as in Distributed Sketch, rather than making inaccurate estimations, a switch with limited memory resource can collaborate with other switches by taking a fair share of the global workload, and perform measurement in a distributed and collective manner. Second, by involving resources from many switches and allowing them to collaborate, Distributed Sketch is capable to aggregate more resources for measuring network flows, therefore provides a higher network-wide accuracy. Third, Distributed Sketch avoids the redundant measurement workload, as the workload is exclusively distributed to the collaborating switches, thus switches can avoid double counting packets while ensure a complete flow coverage. Fourth, Distributed Sketch can be easily implemented on hardware programmable switches, without requiring any additional capability (such as the support of float-point computation) from commodity devices. Finally, Distributed Sketch imposes reasonable costs on switches as well as the measurement control plane, and is practical to be deployed in a large-scale data center networking environment. In the design and implementation of Distributed Sketch, we have made the following key contributions.

- **Design**: We introduce the concepts of the *logical path sketch* and the *physical switch sketch* as building blocks of Distributed Sketch, and propose a novel method for allowing switches to collaboratively construct logical sketches associated with network paths. In particular, we establish methods for mapping logical sketch buckets to physical sketch cells, and develop techniques to improve the mapping fairness across the switches in the network.
- **Implementation**: We implement a Distributed Sketch prototype composed of control and data plane components, and realize the data plane on P4-programmable switches. In particular, to overcome the problem that existing hardware programmable switches do not support float-point computation, we develop an *approximation* method that involves only integer operations, and minimize the approximation error. We also discuss the challenges in deploying Distributed Sketch in large-scale data center networks, and propose a solution based on In-band Network Telemetry (INT).
- **Analysis and evaluation**: We theoretically prove that Distributed Sketch has a lower error bound than alternative solutions; and by conducting extensive experiments driven by real-world traffic traces, we show that Distributed Sketch achieves a higher measurement accuracy compared with the state-of-the-art solutions.

To our best knowledge, Distributed Sketch is the first comprehensive per-flow measurement solution that is based on splitting sketch structure for aggregating and making fair use of resources from all the switches across the network, and is practical to be implemented on commodity hardware programmable switches and deployed in data center networks. For the remainder part of this paper, Sec. II discusses the related works; We present the design and analysis of Distributed Sketch in Sec. III; Sec. IV describes the system implementation and discusses the practical deployment issues; We evaluate Distributed Sketch in Sec. V and conclude this paper in Sec. VI.

II. Related Work

A. Sketch-based Network Measurement

With the development of Software-defined Networking (SDN) and data plane programmability, sketch-based method is considered as a promising direction for fine-grained measurement in data center networks. However, a drawback of the classical sketches of CM [17], CU [18], and Count [19] is their low switch memory utilizations. To overcome this problem, Liu et al. [7] assign sketches to switches by solving an integer linear programming (ILP) problem under the constraint of the switches' available memories. Huang et al. [8] present SketchVisor, which augments a conventional sketch with a fast path for tracing large flows. Yang et al. [10] propose a generic sketch named Elastic sketch that differentiates large flows from small ones, and is adaptive to traffic variances. Yang et al. [11] design the Diamond sketch for dynamically assigning appropriate amount of resources to each flow on demand. Liu et al. [12] propose a new sketch called the Slim-Fat (SF) sketch that improves measurement accuracy without sacrificing the update and query speeds. Zhang et al. [13] propose the On-Off sketch by modifying and extending the classical CM sketch to achieve a smaller error and a higher throughput. Zhang et al. [14] design CocoSketch that is capable to support partial key queries. Yang et al. [15] propose a novel sketch named TowerSketch that records flows using counters of different sizes. Li et al. [16] present Pyramid, a framework that improves both accuracy and throughput via counter sharing. However, most existing works require that an individual switch should host a complete sketch, therefore the network-wide measurement accuracy is constrained by a few switches that have limited memory resources.

B. Coordinated Measurement

Some works exploit the fact that network flows pass multiple switches, and propose methods for coordinating measurement activities on multiple switches or correlating the measurement results. These works can be generally divided into three categories.

1) Merging distributedly collected measurement data: In this category, each switch independently measures passing packets, and reports measurement data to a centralized analyzing server. The server merges and correlates the data to recover errors and improve the estimating accuracy.

To report measurement data, Harris et al. [27] develop a method that allows an individual switch to choose a ratio to compress its local sketch. For merging and correlating measurement data, Li et al. [24] employs a coding-based scheme to encode measurement data at switches and decode data from multiple switches at the control plane. Basat et al. [26] propose to sample packets at switches, and develop control plane algorithms to merge the samples for flow size estimating.

However, [26] is difficult to be realized on hardware data plane, as $O(\log n)$ bits are required on a switch for storing each flow's sampled packets, where n is the upper bound of the flow size, but for a hardware switch, it is infeasible to know n and allocate memory in advance. We stress that the control plane algorithms for error recovery are orthogonal to Distributed Sketch, as the latter seeks to improve measurement accuracy by enabling collaborations among the switches on data plane.

2) Distributing measurement workload by dividing network flows: To fairly distribute measurement workload to all switches along a network path, one approach is to equally divide the flows among the switches, so that each flow is counted by one and only one switch. For example, Yu et al. [9] simply divide network flows between ingress and egress switches. Basat et al. [28] propose that each switch measures a number of carefully selected flows, and switches use a random algorithm to avoid double counting without explicit coordination. A drawback of [28] is that a complete flow coverage can not be realized due to randomness of the algorithm, and as a result, the algorithm must run alongside with a full-coverage measurement system like FlowRadar [24], which definitely increases the workload and resource usages on each individual switch.

3) Distributing measurement workload by splitting sketch structure: Another approach for distributing measurement workload is to split sketch structure and host parts of the sketch on multiple switches. In particular, the work that is most similar to us is DISCO [29], in which the authors propose to allow each switch on a flow's forwarding path to store one or multiple rows of a classical CM sketch, thus distribute the workload across the switches. However, [29] only considers one single path, and restricts the collaboration within that path; in addition, by placing sketch rows on switches arbitrarily, the measurement workload is distributed in an unbalanced way. Cornacchia et al. [30] show that even with a simple bus topology, DISCO can not effectively measure network flows on different paths, and suggest that sketch parameters should be configured with an awareness of the traffic characteristic; unfortunately, such an awareness can not be obtained before conducting the measurement. Different from previous works, in this paper, we consider per-flow measurement for the entire network, and fairly distribute the global measurement workload to all the switches in the network. As far as we know, our work is the first comprehensive solution based on splitting sketch structure for fairly distributing measurement workload across the network, and is practical to be implemented on commodity hardware programmable switches and deployed in large-scale data centers.

III. Design and Analysis

In this section, we first describe the key concepts in Distributed Sketch and present its basic design (Sec. III-A); then we describe how the network-wide measurement workload is distributed among the switches (Sec. III-B); finally, we analyze the estimating accuracy theoretically (Sec. III-C). Table I lists the notations frequently used in this paper.

TABLE I
Frequently used notations

Notation	Meaning
$\mathbf{A}_p$	Logical path sketch of path $p = (s_1, \cdots, s_l)$
$\mathbf{B}_j$	Physical switch sketch of switch s_j on path p
d	Num. of rows in a path sketch or a switch sketch
w	Num. of columns in a path sketch
w_j	Num. of columns in the switch sketch of $\mathbf{B}_j$
$\alpha_j@p, \beta_j@p$	Parameters of bucket mapping interval for s_j on path p
$\mathbf{B}_j[i][r_{ij}(f)]$	Switch sketch cell to which $\mathbf{A}_p[i][h_i(f)]$ is mapped when applying the interval-based mapping
$n_j@p$	Num. of columns in path sketch $\mathbf{A}_p$ that are mapped to s_j on path p
$L_j@p, R_j@p$	Parameters of approximation mapping for s_j on path p
$\mathbf{B}_j[i][r'_{ij}(f)]$	Switch sketch cell to which $\mathbf{A}_p[i][h_i(f)]$ is mapped when applying the approximation mapping

A. Basic Design

1) Path sketch and switch sketch: Without loss of generality, we define an undirected path $p = (s_1, \cdots, s_l)$ as a fixed sequence of switches in the network, where s_1 and s_l are the switches connecting to the origins and destinations of the network flows on the path. Given a path p, we envision that all its switches collectively maintain a logical count-min (CM) sketch, which is referred to as the path's *path sketch*, denoted as $\mathbf{A}_p$. A path sketch has d rows and w columns of logical buckets, and all paths in the network have their path sketches of same structure, i.e., $d \times w$, regardless of their path lengths.

Being a classical CM sketch, a path sketch is associated with d pair-wise independent hash functions, denoted as $h_i(\cdot)$, $i = 1, \cdots, d$. A hash function takes the flow ID (i.e., the 5-tuple) as input and computes a hash value in the range of $\{1, \cdots, w\}$. Logically, when receiving a packet of flow f, the path sketch applies $h_i(\cdot)$ on f's flow ID, locates the d buckets $\mathbf{A}_p[i][h_i(f)]$, $i = 1, \cdots, d$, and increments the bucket values. When querying f, the path sketch returns $\min_i\{\mathbf{A}_p[i][h_i(f)]\}$ as the estimation. All paths in a network have their path sketches associated with a same set of hash functions, which are globally known to all the switches in the network.

A switch s in the network maintains d rows and w_s columns of physical cells, with each cell a b-bit counter. We refer to the $d \times w_s$ cell matrix as the switch's *switch sketch*, denoted as $\mathbf{B}_s$. Unlike the logical path sketch, a switch sketch is a physical data structure in the switch's SRAM memory. Note that all the switches in the network have d rows of cells, which is same to the path sketches, but different switches may have different w_s, i.e., columns of cells, as it depends on a switch's available SRAM memory resource.

For a network path p, we aim to construct its path sketch $\mathbf{A}_p$ collectively on the physical switch sketches of $\mathbf{B}_1, \cdots, \mathbf{B}_l$ along the path. Since $\mathbf{A}_p$ is a logical sketch, we set its column number w sufficiently large. More specifically, we require that for any path $p = (s_1, \cdots, s_l)$, $w \gg \Sigma_{j=1}^{l} w_j$, which means that on any network path, there should be far more logical path sketch buckets than the physical sketch cells from all the switches along the path combined. Note that in Distributed Sketch, although a path sketch is collectively constructed with the switch sketches along the path, however, a switch sketch is not dedicated to one path, but is utilized for constructing

path sketches of all the paths it belongs to.

2) Bucket mapping and sketch operation: A switch s_j on a path p maintains two parameters in $[0, 1]$ for the path, denoted as $\alpha_j@p$ and $\beta_j@p$, with $\alpha_j@p \leq \beta_j@p$. The two parameters are used to determine whether a bucket from the path sketch $\mathbf{A}_p$ should be maintained by a cell in s_j's switch sketch $\mathbf{B}_j$. We refer to the interval between the two parameters as s_j's *bucket mapping interval* on path p, denoted as $[\alpha_j, \beta_j]@p$.

For collectively constructing $\mathbf{A}_p$ on $\mathbf{B}_1, \cdots, \mathbf{B}_l$, we require that the bucket mapping intervals of the switch sketches along a path are mutually exclusive and collectively cover $[0, 1]$, i.e.,

$$\begin{cases} [\alpha_j, \beta_j]@p \cap [\alpha_k, \beta_k]@p = \emptyset, & \forall j, k \in \{1, \cdots, l\}, j \neq k \\ \bigcup_{j=1}^{l} [\alpha_j, \beta_j]@p = [0, 1] \end{cases} \tag{1}$$

Given the conditions in (1), in the following, we describe how the insert and query operations on the path sketch $\mathbf{A}_p$ can be collaboratively realized by the switch sketches. We will describe the methods for assigning the bucket mapping intervals that satisfy (1) to switches in Sec. III-B.

- **Insert operation**: When a switch s_j on the path p receives a packet of flow f forwarded along the path, for each hash value $h_i(f)$, s_j checks if

$$\lfloor \alpha_j@p \cdot w \rceil + 1 \leq h_i(f) \leq \lfloor \beta_j@p \cdot w \rceil \tag{2}$$

where $\lfloor \cdot \rceil$ returns the nearest integer. If yes, s_j computes

$$r_{ij}(f) = \left\lfloor \frac{h_i(f) - \lfloor \alpha_j \cdot w \rceil - 1}{\lfloor \beta_j \cdot w \rceil - \lfloor \alpha_j \cdot w \rceil - 1} \times (w_j - 1) \right\rfloor + 1 \tag{3}$$

to map the path sketch bucket $\mathbf{A}_p[i][h_i(f)]$ to the switch sketch cell $\mathbf{B}_j[i][r_{ij}(f)]$, and increments the cell value. The packet is forwarded to the next hop after s_j has checked all the hash values $h_i(f), i = 1, \cdots, d$ with its bucket mapping interval $[\alpha_j, \beta_j]@p$ and updated the corresponding cells.

- **Query operation**: Switches report their switch sketches to analyzing servers periodically. When querying a flow f that is forwarded along the path p, for each logical path sketch bucket $\mathbf{A}_p[i][h_i(f)]$, the analyzing server first locates the switch s_j to which $\mathbf{A}_p[i][h_i(f)]$ is mapped to with (2), then it applies (3) to find the cell $\mathbf{B}_j[i][r_{ij}(f)]$ in s_j's switch sketch. The size of flow f is estimated as $\min_i\{\mathbf{B}_j[i][r_{ij}(f)]\}$.

Since the mapping from path sketch buckets to switch sketch cells are determined by the bucket mapping intervals, we refer to the above described method as *interval-based mapping*. In Fig. 1, we use an example to demonstrate how the method works. As shown in Fig. 1(a), in a network composed of five switches, we consider two paths, denoted as $p_1 = (s_1, s_3, s_4)$ and $p_2 = (s_2, s_3, s_5)$. For p_1, we suppose that the bucket mapping intervals assigned to s_1, s_3, and s_4 are $[\alpha_1, \beta_1]@p_1 = [0.0, 0.4]$, $[\alpha_3, \beta_3]@p_1 = [0.4, 0.6]$, and $[\alpha_4, \beta_4]@p_1 = [0.6, 1.0]$; and the intervals assigned to s_2, s_3, and s_5 on p_2 are $[0.0, 0.4]$, $[0.4, 0.6]$, and $[0.6, 1.0]$ respectively. Note that since s_3 is on two paths, it maintains two bucket mapping intervals, one for each path, and its switch sketch $\mathbf{B}_3$ is multiplexed to construct both $\mathbf{A}_{p_1}$ (together with $\mathbf{B}_1$ and $\mathbf{B}_4$) and $\mathbf{A}_{p_2}$ (together with $\mathbf{B}_2$ and $\mathbf{B}_5$) simultaneously. Obviously, the bucket mapping intervals on both paths satisfy the conditions in (1).

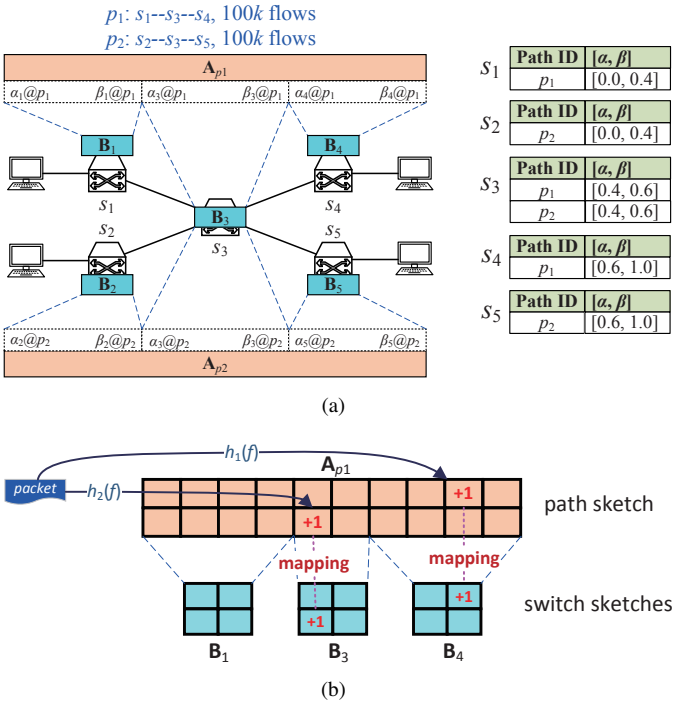


Fig. 1. A demonstrating example of Distributed Sketch and interval-based mapping.

An insert operation is demonstrated in Fig. 1(b). Suppose that $w = 10$ for the path sketch, and s_1, s_3, and s_4 have $w_1 = w_3 = w_4 = 2$ columns of physical cells in their switch sketches. For a network flow f on p_1, if $h_1(f) = 9$ and $h_2(f) = 5$, according to the bucket mapping intervals assigned to s_1, s_3, and s_4 and by applying the interval-based mapping in (2) and (3), the logical path sketch bucket $\mathbf{A}_{p_1}[1][9]$ should be mapped to the physical switch sketch cell $\mathbf{B}_4[1][2]$ on switch s_4, and $\mathbf{A}_{p_1}[2][5]$ should be mapped to $\mathbf{B}_3[2][1]$ on s_3. The two switch sketch cells should be incremented each time a packet of f travels along the path.

When querying the size of flow f, the analyzing server simply applies the interval-based mapping to locate $\mathbf{B}_4[1][2]$ and $\mathbf{B}_3[2][1]$ that $\mathbf{A}_{p_1}[1][9]$ and $\mathbf{A}_{p_1}[2][5]$ are mapped to, and returns $\min\{\mathbf{B}_4[1][2], \mathbf{B}_3[2][1]\}$ as the estimation.

For the collectively constructed path sketch, we have the following result.

Theorem 1. *For a network flow f on a path $p = (s_1, \cdots, s_l)$ whose path sketch $\mathbf{A}_p$ is constructed by applying the interval-based mapping under the conditions in (1), each insert or query operation of f on $\mathbf{A}_p$ maps the flow consistently to d fixed physical switch sketch cells in d different arrays on $\mathbf{B}_1, \cdots, \mathbf{B}_l$.*

Proof. The proof is contained in the Supplementary Material. □

Theorem 1 indicates that functionally, a collectively constructed path sketch is equivalent to a conventional CM sketch

TABLE II
COMPARISON OF WORKLOAD DISTRIBUTIONS UNDER VARIOUS MEASUREMENT METHODS FOR NETWORK IN FIG. 1.

	s_1	s_2	s_3	s_4	s_5
Count-at-core	0	0	$200k$	0	0
Count-at-edge	$100k$	$100k$	0	0	0
Count-everywhere	$100k$	$100k$	$200k$	$100k$	$100k$
Flow-divide	$33.3k$	$33.3k$	$66.7k$	$33.3k$	$33.3k$
Distributed Sketch	$40k$	$40k$	$40k$	$40k$	$40k$

placed at one single switch.

3) Workload distribution: The mapping from path sketch buckets to switch sketch cells decides how a path's measurement workload is distributed to the switches along the path. To show this, look at the bucket mapping intervals assigned to the switches in Fig. 1(a) and suppose that the paths p_1 and p_2 each has $100k$ flows. For the $100k$ flows on p_1, by applying the interval-based mapping, s_1, s_3, and s_4 take measurement workloads equivalent to counting $40k$, $20k$, and $40k$ flows respectively; and for the $100k$ flows on p_2, 40%, 20%, and 40% of the path's workload is imposed on s_2, s_3, and s_5 respectively. For the entire network, each switch takes a fair share of the global workload equivalent to counting $40k$ flows, as listed in Table II.

For comparison, we consider the conventional approach that places a sketch entirely at one switch for counting all passing packets. In particular, we compare Distributed Sketch with three typical sketch placement strategies, namely a) *count-at-core*, b) *count-at-edge*, and c) *count-everywhere*. In the count-at-core strategy, a sketch is placed at the core switch of s_3 to measure the traffic on the paths of p_1 and p_2; in the count-at-edge strategy, sketches are placed at the edge switches of s_1 and s_2 to measure the flows on p_1 and p_2 respectively; and the count-everywhere strategy places a sketch at every switch in the network. Inspired by [28], we also consider an ideal strategy named *flow-divide*, which evenly partitions the network flows along a path among the switches. Under the flow-divide strategy, each switch on paths p_1 or p_2 measures a disjoint set of $1/3$ flows on the path, and the three switches collectively cover all the flows on p_1 or p_2.

Table II lists the measurement workload imposed on each switch in Fig. 1(a) under various strategies. We can see that with the conventional approaches, the measurement workload is distributed in an unbalanced way, as a switch takes either no measurement workload, or a much heavier workload by counting all passing packets without collaborating with other switches; moreover, under the count-everywhere strategy, considerable measurement workload is redundant. The flow-divide strategy avoids redundant measurement workload, but the global workload is distributed unevenly. On the other hand, with the bucket mapping intervals as assigned in Fig. 1(a), the network-wide measurement workload is fairly distributed among the five switches, and redundant workload is avoided.

B. Assigning Bucket Mapping Intervals

Our study in the previous section suggests that the bucket mapping intervals assigned to switches should achieve two objectives: First, the assignment should satisfy the conditions in (1), i.e., the intervals should be mutually exclusive and collectively cover $[0, 1]$; second, the assignment should facilitate the global measurement workload to be fairly distributed to all the switches in the network, as we have seen in Fig. 1 and Table II. In the following, we first present a method that assigns bucket mapping intervals to switches based on switches' memory resources and topological positions, then we incorporate their measurement workloads to adjust the assignment.

1) Topology-based interval assignment: In the topology-based assignment, for each switch s, we first compute its *betweenness centrality* σ_s, which is defined as the number of the distinct undirect paths crossing s in the network [31]. Note that a switch's betweenness centrality is static as long as the topology is unchanged, regardless of the traffic on the paths.

Then for the switches on a network path $p = (s_1, \cdots, s_l)$, we compute their bucket mapping intervals on p as

$$\begin{cases} \alpha_1@p = 0,\ \beta_l@p = 1 \\ \alpha_j@p = \frac{\sum_{k=1}^{j-1} \frac{1}{\sigma_k} \times w_k}{\sum_{k=1}^{l} \frac{1}{\sigma_k} \times w_k}, & \text{for } j = 2, \cdots, l \\ \beta_j@p = \frac{\sum_{k=1}^{j} \frac{1}{\sigma_k} \times w_k}{\sum_{k=1}^{l} \frac{1}{\sigma_k} \times w_k}, & \text{for } j = 1, \cdots, l-1 \end{cases} \tag{4}$$

where w_k is the number of the columns in s_k's switch sketch and σ_k is s_k's betweenness centrality.

The assignment in (4) achieves the above-mentioned objectives. First, the intervals $[\alpha_j, \beta_j]@p, j = 1, \cdots, l$ satisfy the conditions in (1). Second, the length of a switch's bucket mapping interval, i.e., $(\beta_s@p - \alpha_s@p)$, is proportional to w_s and inversely proportional to σ_s, which means that a switch with a larger physical sketch will take more measurement workload, and a switch on many paths will take less workload compared with the ones that are crossed by fewer paths.

We use the example in Fig. 1 to demonstrate how to assign the bucket mapping intervals. Since there are 6 undirect paths in the network, i.e., (s_1, s_3, s_4), (s_2, s_3, s_5), (s_1, s_3, s_2), (s_4, s_3, s_5), (s_1, s_3, s_5), and (s_2, s_3, s_4), for the 5 switches, their betweenness centralities are $\sigma_1 = \sigma_2 = \sigma_4 = \sigma_5 = 3$, and $\sigma_3 = 6$. Suppose $w_1 = w_2 = w_3 = w_4 = w_5 = 2$, then for any 3-hop path in the network, such as p_1 and p_2 in Fig. 1(a), by applying (4), we assign an interval of $[0.4, 0.6]$ to s_3, and the two intervals of $[0.0, 0.4]$ and $[0.6, 1.0]$ are assigned to the two edge switches of the path respectively, as shown in Fig. 1(a).

2) Workload-aware adjustment: The topology-based method in (4) computes bucket mapping intervals based on switches' sketch sizes and betweenness centralities, both are static properties. However, in real-world networks, traffic is distributed unevenly and change dynamically [20], [32]. For example, if the $100k$ flows on path p_2 in Fig. 1(a) complete soon after the interval assignment and there are only the $100k$ flows on path p_1 left in the network, then the interval assignment in Fig. 1(b) is no longer optimal, as s_1 and s_4 each takes a measurement workload of counting $40k$ flows, while s_3 counts only $20k$ flows. Obviously in this case, after the flows on p_2 complete, the assignment should be adjusted to $[0, \frac{1}{3}]$, $[\frac{1}{3}, \frac{2}{3}]$, and $[\frac{2}{3}, 1]$ on s_1, s_3, and s_4 respectively.

From this example, we can see that it is necessary to consider the switches' measurement workloads and adjust the mapping interval assignments dynamically.

We suppose that each switch in the network has two identical switch sketch instances, when one instance is active, the other is backup. The switch swaps its two instances in rounds: after every T seconds, the backup instance becomes active and measures network flows with a set of bucket mapping intervals that are newly assigned by the measurement control plane; while the original active instance becomes backup, and the switch clears all its cells after reporting the counter values to the analyzing server.

We propose that each switch continuously monitors number of the cells in the switch's active sketch instance that are occupied with measurement data, and computes an *occupation ratio* for the switch as

$$\rho_s = \frac{\text{num. of occupied cells on switch } s}{d \times w_s}$$

With the occupation ratios, for the switches on a network path $p = (s_1, \cdots, s_l)$, the measurement control plane assigns their bucket mapping intervals for the next round as

$$\begin{cases} \alpha_1@p = 0,\ \beta_l@p = 1 \\ \alpha_j@p = \frac{\sum_{k=1}^{j-1} \frac{1}{\sigma_k} \times w_k \times \frac{1}{\rho_k^\gamma}}{\sum_{k=1}^{l} \frac{1}{\sigma_k} \times w_k \times \frac{1}{\rho_k^\gamma}}, & \text{for } j = 2, \cdots, l \\ \beta_j@p = \frac{\sum_{k=1}^{j} \frac{1}{\sigma_k} \times w_k \times \frac{1}{\rho_k^\gamma}}{\sum_{k=1}^{l} \frac{1}{\sigma_k} \times w_k \times \frac{1}{\rho_k^\gamma}}, & \text{for } j = 1, \cdots, l-1 \end{cases} \tag{5}$$

where ρ_k is switch s_k's occupation ratio and $\gamma \geq 0$ is a parameter controlling the aggressiveness of the adjustment. The assignment in (5) is easy to understand: among all the switches along a path, the higher occupation ratio a switch has comparing with the other switches, the smaller bucket mapping interval (and consequently, a lighter workload) it will be assigned in the next round.

One requirement for dynamically adjusting the interval assignment is that switches should be synchronized to swap their sketch instances. We believe that this is not an issue, as the state-of-the-art Precision Time Protocol (PTP) can provide a nanosecond-scale time synchronization for networked devices [33]. Even without time synchronization, the controller can configure switches with new bucket mapping intervals ahead of time, and instructs all the switches to swap at a same moment.

Another issue is that during a packet's journey along a network path, the controller instructs the switches along the path to swap their sketch instances. In this case, the packet will be counted by some switch sketch instances before the swap and by some other instances after the swap. One possible solution is that when a packet is forwarded by the first-hop switch, the switch stamps the packet with the current round number, and when a down-stream switch receives the packet, it checks the round number, and counts the packet with the corresponding switch sketch instance, regardless whether the instance is backup or not. However, although the method ensures that a packet is counted by switch sketch instances of a same round, it can not solve the problem completely, as when a sketch instance becomes backup, the switch starts to send it, either in pieces [34] or as a whole [10], to analyzing servers. If data in a switch sketch cell has already been transferred to the analyzing server, any subsequent updates on it is meaningless. Nevertheless, the number of the packets impacted by such an error is up to the flow's bandwidth delay product (BDP), which is small because of the short RTTs and shallow switch buffers in data center networks [35]. For example, with a typical path RTT of $200\,\mu$s [36], for a network flow of $100\,\mathrm{Mbit/s}$, its BDP is less than 2 MTU-sized packets, or in other words, the flow has at most 2 packets that might pass switch sketch instances of different rounds and be counted mistakenly.

C. Accuracy Analysis

Intuitively, by aggregating resources from multiple switches, Distributed Sketch would achieve a higher measurement accuracy than the conventional approaches that place a complete CM sketch at one single switch. In the following, we formally analyze the measurement accuracy, and without loss of generality, we consider a network path $p = (s_1, \cdots, s_l)$, on which a switch s_j has $d \times w_j$ sketch cells. We have the following result.

Theorem 2. *For a network flow f on path p, let n_f be f's ground-truth size, and let $\hat{n}_f$ be its size estimated by Distributed Sketch, then for any $\varepsilon > 0$, we have*

$$\Pr\left[\hat{n}_f > n_f + \varepsilon N\right] \leq \left(\frac{1}{\varepsilon W}\right)^d$$

where $W = \sum_{j=1}^{l} w_j$ and N is the total number of the packets traveled along the path.

Proof. The proof is contained in the Supplementary Material. □

Theorem 2 indicates that under Distributed Sketch, the measurement accuracy on a network path p directly depends on the aggregated size of the switch sketches along the path. If we apply the count-at-core or count-at-edge strategy, then from Theorem 2, it is easy to see that

$$\Pr\left[\hat{n}_f > n_f + \varepsilon N\right] \leq \left(\frac{1}{\varepsilon \times w_s}\right)^d$$

where w_s is the column number of the CM sketch at the core or edge switch. While with the count-everywhere strategy that places a CM sketch at each switch along the path, we have

$$\Pr\left[\hat{n}_f > n_f + \varepsilon N\right] \leq \left(\frac{1}{\varepsilon \times \max_{s \in p}\{w_s\}}\right)^d$$

Since $W = \sum_{j=1}^{l} w_j > \max_{s \in p}\{w_s\} \geq w_s$, one can see that Distributed Sketch has an lower error bound than the conventional CM placement strategies.

IV. Implementation

In this section, we present the implementation of Distributed Sketch. In particular, to overcome the problem that hardware programmable switches do not support float-point computation [37], [38], we present an *approximation* method that involves only integer operations to map logical sketch buckets to physical sketch cells, while minimizes the approximation

error. We also discuss the challenges for practically deploying Distributed Sketch on large-scale data center networks, and propose an In-band Network Telemetry (INT) based solution.

A. Approximation Mapping

From Sec. III-A, one can see that for implementing Distributed Sketch, a programmable switch needs to: a) maintain a counter array as the switch sketch in its SRAM memory; b) perform hash computations on each packet's flow ID; and c) check whether a logical path sketch bucket is mapped to a cell in its switch sketch, locate the cell and update its value. Note that both a) and b) are also required when implementing a classical CM sketch on a hardware programmable switch (e.g., the Barefoot Tofino switch), and have been successfully realized in previous works (e.g., [10] and [39]). However, it is challenging to realize c), as the interval-based bucket mapping as described in Sec. III-A2 involves float-point computations in (2) and (3), while unfortunately, most hardware programmable switch architectures do not support float-point types [37], [38]. For example, the P4-programmable Portable Switch Architecture (PSA) [40] allows only integer addition/subtraction and arithmetic bit-shift equivalent to multiplication/division by power of 2, and on a programmable switch following the PSA architecture, such as the Barefoot Tofino switch, it is infeasible to program (2) and (3), which contain float-point computations, directly in P4 actions. To overcome this problem, in the following, we present an *approximation* method that involves only integer operations for mapping path sketch buckets to switch sketch cells.

To restrict the computation to integer operations, we require that on each switch s_j in the network, the number of the cell columns in its switch sketch should be power of 2, i.e., $w_j = 2^{m_j}$, where $m_j \in \mathbb{N}^0$ is an integer no smaller than 0. Moreover, we require that for a network path $p = (s_1, \cdots, s_l)$, after the approximation mapping, $2^{n_j@p}$ columns in the path sketch buckets in $\mathbf{A}_p$ are mapped to the switch sketch $\mathbf{B}_j$ on s_j, where $n_j@p$ is an integer no smaller than m_j. Clearly, for $\mathbf{B}_1, \cdots, \mathbf{B}_l$ along the path, $\sum_{j=1}^{l} 2^{n_j@p} = w$.

Recall that when applying the interval-based mapping as in Sec. III-A2, $(\lfloor \beta_j@p \cdot w \rceil - \lfloor \alpha_j@p \cdot w \rceil + 1)$ columns of the path sketch $\mathbf{A}_p$ are mapped to s_j. To compare the approximation mapping with the interval-based mapping, for each switch, we compute *approximation error* as the difference ratio between the numbers of the path sketch columns that are mapped to a same switch s_j by the two methods, which is defined as

$$\Delta_j = \frac{|(\lfloor \beta_j@p \cdot w \rceil - \lfloor \alpha_j@p \cdot w \rceil + 1) - 2^{n_j@p}|}{w}$$

where w is the column number of the path sketch.

For the network path $p = (s_1, \cdots, s_l)$, our objective is to find the approximation mapping that is closest to the bucket mapping intervals of $[\alpha_j, \beta_j]@p$ computed with (4) or (5), in terms of the aggregated approximation error. To achieve this goal, we need to find $n_j@p$ for each switch s_j along the path,

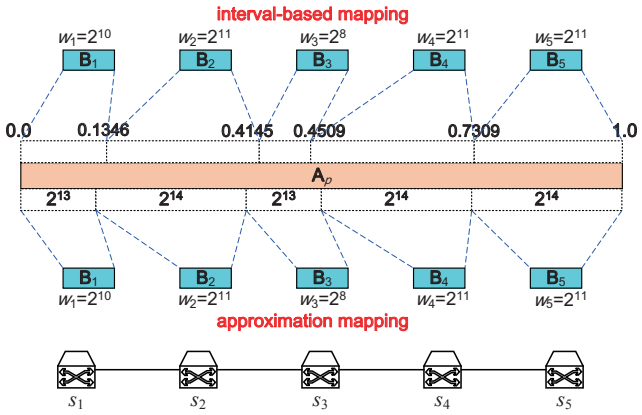


Fig. 2. Comparison between interval-based mapping and approximation mapping on a 5-hop network path.

such that the aggregated approximation error is minimized. Formally, the problem is

$$\begin{array}{ll} minimize & \sum_{j=1}^{l} \Delta_j \\ s.t. & \sum_{j=1}^{l} 2^{n_j@p} = w \\ & n_j@p \geq m_j = \log_2 w_j \\ & n_j@p \in \mathbb{N}^0 \end{array} \tag{6}$$

After solving the problem in (6) and having obtained the solution $(n_1, \cdots, n_l)@p$, for each switch s_j along the path, the measurement control plane computes two integers as:

$$\begin{array}{rcl} L_j@p & = & \sum_{k=1}^{j-1} 2^{n_k@p} + 1 \\ R_j@p & = & \sum_{k=1}^{j} 2^{n_k@p} \end{array} \tag{7}$$

and use them as well as $n_j@p$ to realize the insert and query operations on $\mathbf{A}_p$ with $\mathbf{B}_1, \cdots, \mathbf{B}_l$ as the following.

- **Insert operation**: When a switch s_j on path p receives a packet of flow f forwarded along the path, s_j checks if

$$L_j@p \leq h_i(f) \leq R_j@p \tag{8}$$

  If yes, it computes

$$r'_{ij}(f) = \frac{h_i(f) - L_j@p}{2^{(n_j@p - m_j)}} + 1 \tag{9}$$

  maps $\mathbf{A}_p[i][h_i(f)]$ to $\mathbf{B}_j[i][r'_{ij}(f)]$, and increments the switch sketch cell $\mathbf{B}_j[i][r'_{ij}(f)]$, for $i = 1, \cdots, d$.
- **Query operation**: When querying flow f, the analyzing server applies (8) and (9) to locate the mapped switch sketch cells, and returns $\min_i\{\mathbf{B}_j[i][r'_{ij}(f)]\}$ as the estimation.

Note that both (8) and (9) involve only integer operations, thus can be directly programmed as match+action entries on hardware programmable switches. Moreover, since the integer sets of $\{L_j, R_j\}@p$, $j = 1, \cdots, l$, are disjoint and collectively constitute $\{1, ..., w\}$, and the computation in (9) is deterministic, similar to Theorem 1, it can be proved that by applying the approximation mapping method, a path sketch constructed by the switch sketches along the path is functionally equivalent to a conventional CM sketch.

In Fig. 2 and Table III, we demonstrate an example of the approximation mapping. We consider a 5-hop network

TABLE III
DETAILS OF INTERVAL-BASED MAPPING AND APPROXIMATION MAPPING ON THE 5-HOP PATH IN FIG. 2.

Switch	s_1	s_2	s_3	s_4	s_5
$[\alpha_j, \beta_j]@p$	$[0.0, 0.1346]$	$[0.1346, 0.4145]$	$[0.4145, 0.4509]$	$[0.4509, 0.7309]$	$[0.7309, 1.0]$
$n_j@p$, $L_j@p$, $R_j@p$	13, 1, 8192	14, 8193, 24576	13, 24577, 32768	14, 32769, 49152	14, 49153, 65536
$r'_{ij}(f)$	$\frac{h_i(f)-1}{2^3}+1$	$\frac{h_i(f)-8193}{2^3}+1$	$\frac{h_i(f)-24577}{2^5}+1$	$\frac{h_i(f)-32769}{2^3}+1$	$\frac{h_i(f)-49153}{2^3}+1$

path as $p = (s_1, s_2, s_3, s_4, s_5)$, where each switch hosts a switch sketch of different sizes with w_j varying between 2^8 and 2^{11}. The path sketch is configured to have $w = 2^{16}$ columns of logical buckets. Suppose that by applying (4) or (5), the measurement control plane assigns each switch a bucket mapping interval $[\alpha, \beta]$, which we present in Fig. 2 and list in the 2^{nd} row of Table III. By solving the optimization problem in (6) with $[\alpha_j, \beta_j]@p$ as the parameters, we have obtained $n_j@p$, $L_j@p$, and $R_j@p$ for each switch, and list them in the 3^{rd} row of Table III. We also present the $2^{n_j@p}$ columns of the path sketch $\mathbf{A}_p$ that are mapped to each switch in Fig. 2 for comparison. From the figure we can see that the mapping decided by the approximation method is very close to the one decided by the interval-based method. In the last row of Table III, we derive the formulas based on (9) with $n_j@p$, $L_j@p$, and $R_j@p$ as the parameters for each switch to map path sketch buckets to its switch sketch cells. We can see that all the formulas involve only hash functions, integer additions/subtractions, and bit-shift operations, which are natively supported by nearly all commodity hardware switches.

For a concrete instance, consider a network flow f on the 5-hop path in Fig. 2, which has a hash value $h_1(f) = 26,876$. When applying the interval-based mapping, since $\alpha_2@p < \frac{h_1(f)}{w} < \beta_2@p$, the logical path sketch bucket $\mathbf{A}_p[1][26876]$ should be mapped to the switch sketch cell $\mathbf{B}_2[1][2016]$ on s_2. But with the approximation mapping, as $L_3@p < h_1(f) < R_3@p$, $\mathbf{A}_p[1][26876]$ is now mapped to $\mathbf{B}_3[1][72]$ on s_3 according to the formula of $(\frac{h_i(f)-24577}{2^5}+1)$ as listed in Table III.

We recognize that the approximation mapping does not strictly enforce the bucket mapping intervals as computed with (4) or (5) on switches, therefore the measurement accuracy may be slightly impacted as the workload is not distributed as fairly as under the interval-based mapping. However, the approximation mapping involves only integer operations, which makes it feasible to be programmed on hardware programmable switches. We will evaluate and compare the two mapping methods in Sec. V.

B. System Prototype

We have implemented a Distributed Sketch system prototype as demonstrated in Fig. 3. The system contains both control plane and data plane components. The measurement control plane contains the modules for computing betweenness centrality and solving the integer optimization problem in (6). More importantly, the control plane maintains two tables, a *path table* and a *switch table*. The path table associates each flow with a network path that it travels along, and the table

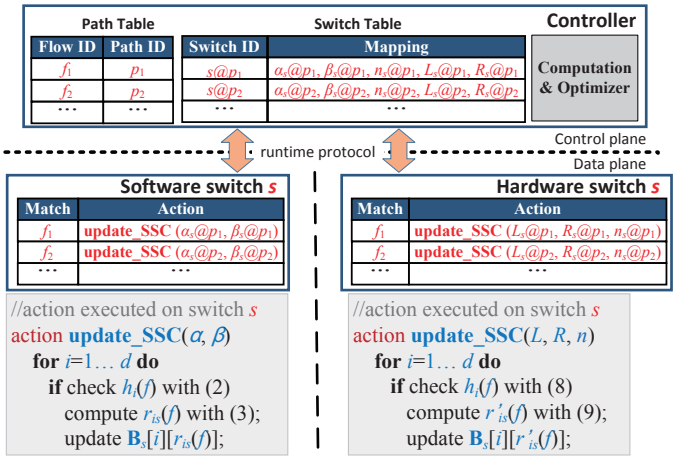


Fig. 3. System overview of Distributed Sketch.

is updated each time a new flow is planned in the network or an inactive flow is evicted.

The switch table keeps the parameters for each switch on a path. More specifically, for switch s on path p, its table entry contains $\alpha_s@p$, $\beta_s@p$, $n_s@p$, $L_s@p$, and $R_s@p$, in which $\alpha_s@p$ and $\beta_s@p$ are computed by applying (4) or (5), $n_s@p$ is obtained by solving the optimization problem in (6), and $L_s@p$ and $R_s@p$ are computed with (7). The parameters are used for software or hardware switches to map logical path sketch buckets to their switch sketch cells, following the interval-based mapping or the approximation mapping methods as described in Sec. III-A2 and Sec. IV-A respectively.

In the data plane, we define a new action namely `update_SSC` for locating and updating the switch sketch cells. As shown in the pseudo code in Fig. 3, the action is implemented differently on software or hardware programmable switches. On a software switch, the action checks each hash value $h_i(f)$ with (2), and applies (3) to locate and update the mapped switch sketch cells; while on a hardware programmable switch, the action applies (8) and (9), which involve only integer operations, to locate and update the mapped switch sketch cells in its SRAM.

The Distributed Sketch system imposes reasonable costs on both the data plane and the measurement control plane. On the data plane, a switch s only needs to maintain $O(|\mathbb{F}_s|)$ match+action entries, where $\mathbb{F}_s$ is the set of the concurrent active flows passing s, which is up to a few thousands in a large data center network and can be easily accommodated by a commodity programmable switch [35]. On the control plane, the path table has a size of $O(|\mathbb{F}| \times |\mathbb{P}|)$, and the size of the switch table is $O(|\mathbb{S}| \times |\mathbb{P}|)$, where $\mathbb{F}$, $\mathbb{P}$, and $\mathbb{S}$ are the

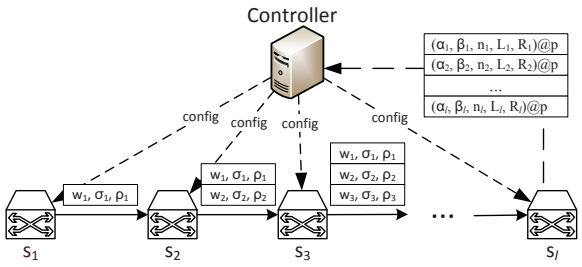


Fig. 4. Computing parameters and configuring switches in distributed manner.

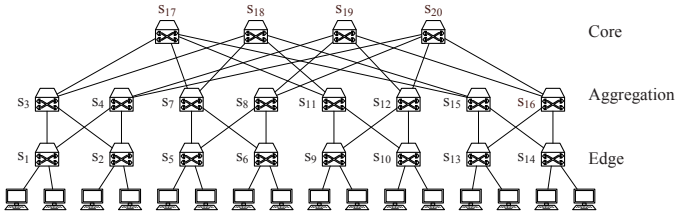


Fig. 5. FatTree testbed.

sets of the active flows, paths, and switches in the network respectively. Note that a large-scale data center network may contain billions of paths [41], which result in a large-sized path table and switch table. Fortunately, as we will discuss in the next section, the two tables can be maintained in a distributed manner.

Our system prototype contains the P4-implementations verified on `bmv2` [42] and the Edgecore Wedge 100BF-32X Tofino-based hardware switch [23], and we have shared the code to the community[1].

C. Deployment Issues

For practically deploying Distributed Sketch in a large-scale data center network environment, there are a few challenges. The first challenge is how to maintain the large-sized path and switch tables on the measurement control plane. One naive way is to keep the two tables with a centralized cluster, however, such an approach requires a dedicated infrastructure and imposes non-trivial overhead.

We propose to maintain the path and switch tables in a distributed manner. In particular, we require each switch that acts as the last hop of a path be responsible for maintaining the path and switch table entries for that path. Since a undirect path has two last-hop switches and the measurement parameters in Distributed Sketch are updated in rounds, the two last-hop switches can take turns to be in charge, that is, in even rounds, the switch at one end computes and configures the parameters for all the switches along the path, and in odd rounds, the switch at the other end takes the responsibility.

More specifically, consider a path $p = (s_1, \cdots, s_l)$ as shown in Fig. 4, on which s_1 and s_l are the last-hop switches responsible for the current and next rounds respectively. We employ INT to allow each switch s to insert the its local states of w_s, σ_s, and ρ_s, which are the parameters for computing the mapping intervals in (5), to the INT fields in packet header. After collecting the local states of all the switches along the path, both s_1 and s_l can apply (5), (6), and (7) to compute the parameters of $\alpha_s@p$, $\beta_s@p$, $n_s@p$, $L_s@p$, and $R_s@p$ for each switch s on the path. Before a new round starts, the responsible last-hop switch sends the parameters to the network controller, which uses them to configure all the switches along the path. In addition, when a switch detects a new path that it is in charge of, it computes the parameters, and contacts the controller to configure the switches along the path immediately.

[1]https://github.com/DpintHPCC416/Distribute

TABLE IV
CONFIGURATIONS OF CM, TOWERSKETCH, AND ELASTIC SKETCH INSTANCES ON SWITCHES OF DIFFERENT MEMORY SIZES.

Sketch columns w_s	2^8	2^9	2^{10}	2^{11}	2^{12}
Mem. size	2 kB	4 kB	8 kB	16 kB	32 kB
CM (buckets)	2×2^8	2×2^9	2×2^{10}	2×2^{11}	2×2^{12}
Tower top (buckets)	2^8	2^9	2^{10}	2^{11}	2^{12}
Tower bottom (buckets)	2^9	2^{10}	2^{11}	2^{12}	2^{13}
Elastic light (buckets)	2×2^9	2×2^{10}	2×2^{11}	2×2^{12}	2×2^{13}
Elastic heavy (entries)	50	100	200	400	800

An issue of the above approach is that when a new path appears in the network and some packets start to travel along it before the path is configured by Distributed Sketch, the packets will not be counted by any switch sketch. To overcome this problem, we suggest that each first-hop switch maintains a *stand-alone sketch*, which could be either a classical sketch or an advanced one, and when a network flow is missed by Distributed Sketch without matching any match+action entry on the first-hop switch, it is counted by the stand-alone sketch. Note that since only the packets in a short interval before a new path is configured by Distributed Sketch are counted, the stand-alone sketch does not need to maintain large counters, and could be small in size.

V. EVALUATION

In this section, we evaluate our proposed Distributed Sketch method with experiments. In particular, we construct a data center network testbed, apply our proposed method for conducting representative per-flow measurement tasks, and compare it with the state-of-the-art alternative solutions.

A. Experiment Setup and Metrics

We emulate a data center network with a FatTree topology [43] using Mininet [44]. As shown in Fig. 5, the network is composed of 20 P4-programmable switches [42], denoted as s_1–s_{20}. Hosts connect to edge switches, and there are a total number of 112 distinct undirect paths in the network: 8 one-hop paths connecting hosts in same subnets, 8 3-hop paths between intra-pod hosts, and 96 5-hop paths connecting inter-pod hosts.

Switches in the FatTree network have different memory sizes for hosting sketch data structures. More specifically, in Distributed Sketch, a switch s is configured to maintain d rows and w_s columns of 32-bit counters in its switch sketch, where $d = 2$ and w_s is randomly selected from $\{2^8, 2^9, 2^{10}, 2^{11}, 2^{12}\}$.

TABLE V
COMPARISONS OF PER-PACKET MEASUREMENT OVERHEADS ON AN l-HOP PATH FOR DIFFERENT METHODS.

	Mem. accesses	Hash calls
Distributed-hardware	d	l
CM-at-core	d	1
CM-at-edge	d	1
CM-everywhere	$d \times l$	l
Tower-at-core	d	1
Tower-at-edge	d	1
Tower-everywhere	$d \times l$	l
Elastic-at-core	1 to $(1+d)$	1
Elastic-at-edge	1 to $(1+d)$	1
Elastic-everywhere	l to $l \times (1+d)$	l

We compare Distributed Sketch with the alternative network measurement strategies as discussed in Sec. III. When applying these strategies, we choose an instance of the classical CM sketch [17], or the recently proposed advanced sketches of TowerSketch [15] or Elastic sketch [10] to place at each individual switch. Note that similar to our proposed Distributed Sketch, TowerSketch contains only counters, but Elastic sketch records flow IDs for elephant flows. For ensuring a fair comparison, each type of sketch on a switch is constrained by the switch's memory size. More specifically, the CM sketch on switch s has $d = 2$ rows and w_s columns of 32-bit buckets; a TowerSketch instance on switch s has $d = 2$ arrays: the top array contains w_s 32-bit buckets, and the bottom array contains $2 \times w_s$ 16-bit counters, as suggested in [15]; for an Elastic sketch placed on s, its light part is composed of $d = 2$ rows and $2 \times w_s$ columns of 8-bit buckets, which consumes 50% of the switch's memory, and the heavy part consumes the other 50% memory with a hash table, where each table entry is 162 bits[2]. The detailed configurations of the sketches are listed in Table IV.

We evaluate and compare the following 12 per-flow measurement methods.

- **Distributed-software**: In this method, we first obtain the bucket mapping intervals for each switch in the network using (4) or (5), then we apply the interval-based mapping as elaborated in Sec. III-A2 to measure the network flows. For each path in the network, we set $w = 2^{16}$ for the logical path sketch.
- **Distributed-hardware**: In this method, we first apply (4) or (5) to obtain the bucket mapping intervals for each switch, then we apply the approximation mapping as described in Sec. IV-A to perform the measurement. The other parameters are same as in **Distributed-software**.
- **CM/Tower/Elastic-at-core**: In CM-at-core / Tower-at-core / Elastic-at-core, the core switches of the FatTree network, i.e., s_{17}, s_{18}, s_{19}, and s_{20}, each hosts an instance of CM sketch / TowerSketch / Elastic sketch as configured in Table IV to perform the measurement.
- **CM/Tower/Elastic-at-edge**: In CM-at-edge / Tower-at-edge / Elastic-at-edge, a CM sketch / TowerSketch / Elastic sketch instance is placed at each edge switch in the FatTree network, i.e., s_1, s_2, s_5, s_6, s_9, s_{10}, s_{13}, and s_{14}, for measuring network flows that pass them.
- **CM/Tower/Elastic-everywhere**: In CM-everywhere / Tower-everywhere / Elastic-everywhere, each switch in the FatTree network employs a CM sketch / TowerSketch / Elastic sketch instance to measure the network traffic. If a flow is counted by multiple switches, we take the minimum as the estimating result.
- **CM-flow-divide**: Inspired by [28], in CM-flow-divide, each switch along a k-hop path maintains a CM sketch, and we assume that each switch ideally selects a disjoint set of $1/k$ flows on the path to perform the measurement.

We apply the above 12 methods to perform the following representative per-flow measurement tasks.

- **Flow size estimation**: In this task, we estimate a flow's size and compare it with the ground truth. We use the metric of *Averaged Relative Error* (*ARE*) to evaluate the measurement accuracy, which is defined as
$$ARE = \frac{1}{|\mathbb{F}|} \sum_{f \in \mathbb{F}} \frac{|n_f - \hat{n}_f|}{n_f}$$
where n_f and $\hat{n}_f$ are the ground-truth and estimated sizes of flow f, and $\mathbb{F}$ is the set of the flows in the network.
- **Heavy hitter detection**: This task seeks to identify the heavy hitter flows, which are defined as the top 10% largest flows in $\mathbb{F}$. We use the *F1-score* to assess the detecting accuracy.
- **Flow size distribution estimation**: We use the *Weighted Mean Relative Error* (*WMRE*) to measure the discrepancy between an estimated flow size distribution and the ground truth as
$$WMRE = \frac{\sum_i |m_i - \hat{m}_i|}{\sum_i (\frac{|m_i + \hat{m}_i|}{2})}$$
where m_i and $\hat{m}_i$ are the ground-truth and estimated numbers of the flows of size i.

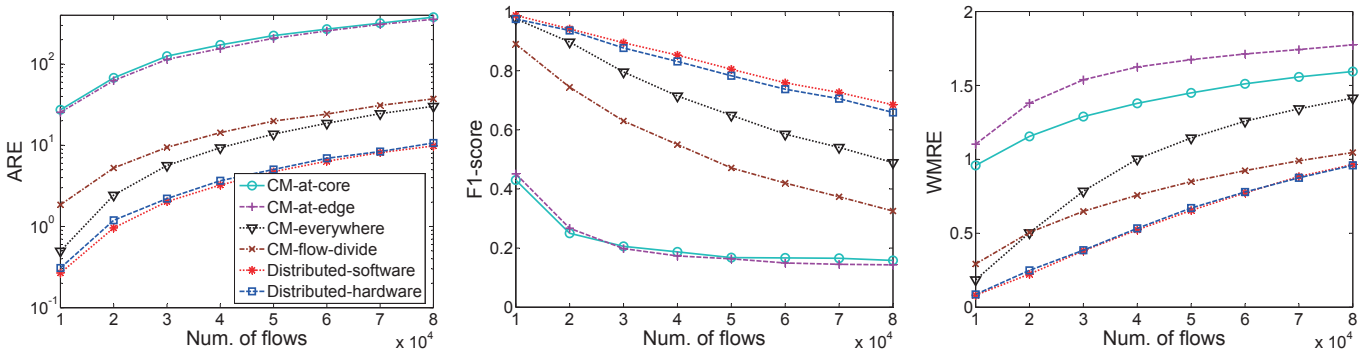
We use the publicly available MAWI packet trace [45] captured from the WIDE backbone to drive our experiments.

B. Measurement Overhead

Before presenting the experiment results, we first analyze the per-packet measurement overhead of different methods under the evaluation. We consider a network flow forwarded along an l-hop path, and examine the overhead including memory accesses and hash computations needed in handling each packet, as they are the major components of the measurement overhead [46]. For memory accesses, with our proposed **Distributed-hardware** method, all the l switches collectively access d switch sketch cells per packet; while in **CM-at-core**, **CM-at-edge**, and **CM-everywhere**, each switch accesses d CM sketch buckets per packet, therefore for the l-hop path, the memory accesses are d, d, and $d \times l$ respectively.

Tower-at-core, **Tower-at-edge**, and **Tower-everywhere** incur same numbers of per-packet memory accesses as the methods of **CM-at-core**, **CM-at-edge**, and **CM-everywhere**,

[2]A table entry in the Elastic sketch heavy part contains a 5-tuple flow ID, two 32-bit counters, and a 1-bit flag. We use 32 bits for an IP address, 16 bits for a TCP/UDP port, and one bit to indicate the protocol, therefore the total entry size is 162 bits.



(a) Flow size estimation (b) Heavy hitter detection (c) Flow size distribution estimation

Fig. 6. Measurement accuracies for (a) flow size estimation, (b) heavy hitter detection, and (c) flow size distribution estimation tasks accomplished by **CM-at-core**, **CM-at-edge**, **CM-everywhere**, **CM-flow-divide**, **Distributed-software**, and **Distributed-hardware** when measuring various numbers of flows.

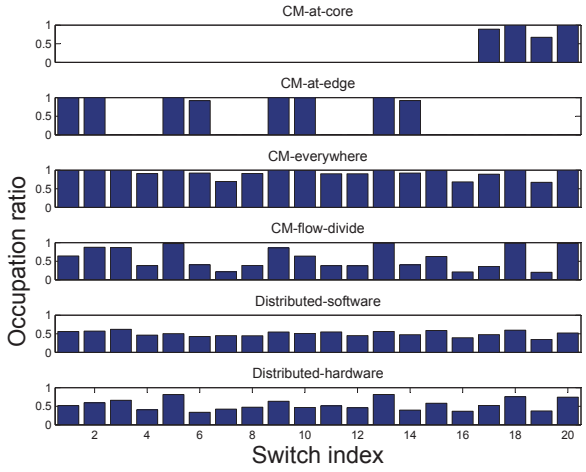


Fig. 7. Bucket/cell occupation ratios of **CM-at-core**, **CM-at-edge**, **CM-everywhere**, **CM-flow-divide**, **Distributed-software**, and **Distributed-hardware**.

as the two sketches only differ in sketch structure and hash functions, but have same procedure in updating sketch buckets.

Under the **Elastic-at-core**, **Elastic-at-edge**, and **Elastic-everywhere** schemes, when receiving a packet, each switch first accesses the hash table in the heavy part of its hosted Elastic sketch, if missed, it then accesses d buckets in the light part, therefore the per-packet memory access overhead is 1 or $1 + d$ on each switch.

For the hash computation overhead, a recent study [46] suggests that a switch can consolidate multiple hash functions into one hash call by dividing and reusing the result, therefore a switch needs to compute only one hash value per packet. We do not examine **CM-flow-divide** as it is an ideal method.

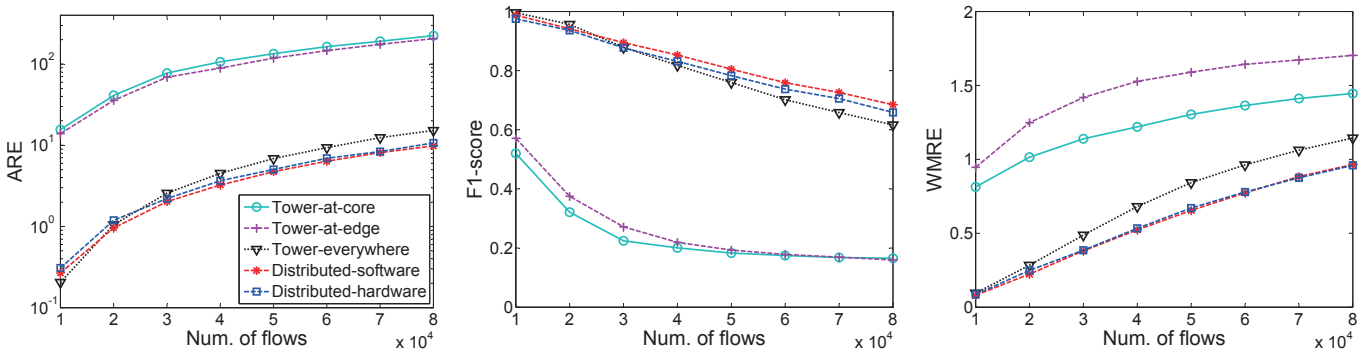
Table V summarizes the per-packet memory access and hash computation overheads of different measurement methods over an l-hop path. From the table one can see that our proposed **Distributed-hardware** method does not introduce extra computation overhead compared with **CM/Tower/Elastic-everywhere**; moreover, its memory access overhead is comparable to the approaches that employ only one switch along the path, i.e., **CM/Tower/Elastic-at-core** and **CM/Tower/Elastic-at-edge**, and is much lighter than **CM/Tower/Elastic-everywhere** whose memory access overheads increase with path length.

C. Experiment Results

We evaluate and compare accuracies of different methods in accomplishing the measurement tasks. In our experiment, we select $10,000$ to $80,000$ flows from the MAWI trace and evenly divide them to the 112 paths in the FatTree network. Each experiment lasted 10 seconds, and was repeated 15 times. For the methods based on Distributed Sketch, i.e., **Distributed-software** and **Distributed-hardware**, we apply (4) to compute the bucket mapping intervals.

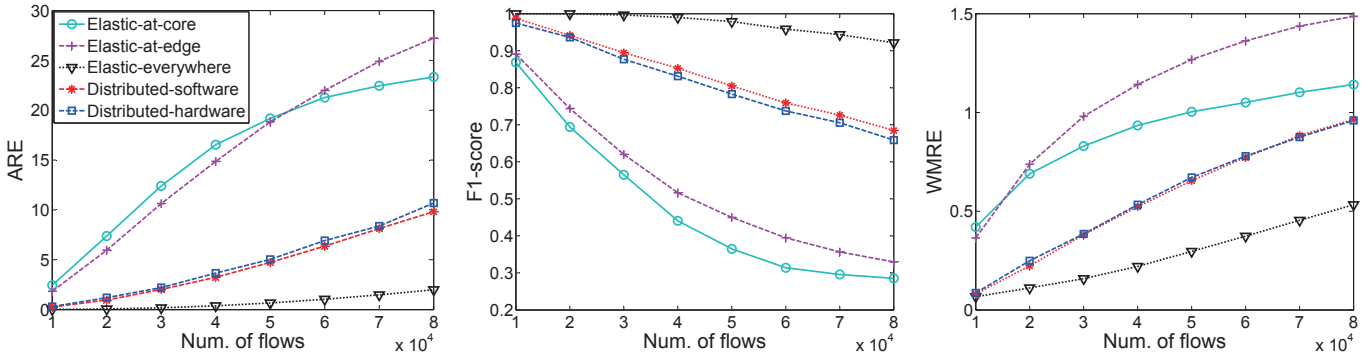
1) Comparison with CM-based methods: Fig. 6 presents ARE, F1-score, and WMRE of the measurement tasks achieved by the **CM-at-core**, **CM-at-edge**, **CM-everywhere**, **CM-flow-divide**, as well as our proposed **Distributed-software** and **Distributed-hardware** methods. From the figures one can see that **CM-everywhere** considerably outperforms **CM-at-core** and **CM-at-edge**, **CM-flow-divide** also has a good performance, and **Distributed-software** and **Distributed-hardware** are significantly more accurate than the other solutions. The better performance of **CM-everywhere** over **CM-at-core** and **CM-at-edge** is easy to understand, as in **CM-everywhere**, a network flow is independently measured by up to five switches along its forwarding path, therefore its estimation accuracy is largely decided by the largest sketch along the path, but in **CM-at-core** and **CM-at-edge**, the CM sketch at the core or edge switch has a random size, which could be small and inaccurate.

Distributed-software and **Distributed-hardware** outperform **CM-everywhere** for the reason that they employ all the cells from the switches along the path, which far exceed the buckets of the largest CM sketch. **Distributed-software/hardware** also outperform the **CM-flow-divide** method, despite that the latter employs all the switches in the network. The reason behind is that **CM-flow-divide** simply equal-divides flows among the switches, but does not consider each switch's memory resource and topological position, thus distributes the measurement workload unfairly among the switches. We find that **Distributed-hardware** is only slightly less accurate than **Distributed-software**, suggesting that the



(a) Flow size estimation (b) Heavy hitter detection (c) Flow size distribution estimation

Fig. 8. Measurement accuracies for (a) flow size estimation, (b) heavy hitter detection, and (c) flow size distribution estimation tasks accomplished by **Tower-at-core**, **Tower-at-edge**, **Tower-everywhere**, **Distributed-software**, and **Distributed-hardware** when measuring various numbers of flows.



(a) Flow size estimation (b) Heavy hitter detection (c) Flow size distribution estimation

Fig. 9. Measurement accuracies for (a) flow size estimation, (b) heavy hitter detection, and (c) flow size distribution estimation tasks accomplished by **Elastic-at-core**, **Elastic-at-edge**, **Elastic-everywhere**, **Distributed-software**, and **Distributed-hardware** when measuring various numbers of flows.

approximation mapping used in **Distributed-hardware** dispatches measurement workload to switches very close to the interval-based mapping used in **Distributed-software**.

To further understand the reason behind the better performances of the methods based on Distributed Sketch, in Fig. 7, we present a switch's bucket/cell occupation ratio for all the 20 switches in the FatTree network when measuring $20,000$ flows using different methods. From the figure one can see that compared with **CM-at-core** and **CM-at-edge**, the **Distributed-software** and **Distributed-hardware** methods distribute the measurement workload to all the switches in the network; compared with **CM-everywhere**, the workload on each switch under **Distributed-software** or **Distributed-hardware** is much lighter; and compared with **CM-flow-divide**, the workload distributed by **Distributed-software** or **Distributed-hardware** is more balanced across the switches. For example, with **CM-everywhere**, the mean occupation ratio is 91.91%, and 7 of the 20 switches have their buckets 100% occupied, while under **CM-flow-divide**, although the averaged occupation ratio is reduced to 58.88%, however, 4 switches have their bucket occupation ratios exceeding 97%. But with the **Distributed-software** and **Distributed-hardware** methods, the mean occupation ratios are as low as 50.25% and 54.15%, and none of the switches in the two methods has an occupation ratio exceeding 65% and 82% respectively. Obviously, the less the sketches are occupied, and the more balanced the measurement workload is distributed among the switches, the higher accuracy we can expect from a sketch-based measurement solution. Finally, Fig. 7 shows that the workload distributed by **Distributed-software** to switches is more balanced than by **Distributed-hardware**, as the latter applies the approximation mapping that does not strictly enforce the intervals assigned by (4) from the control plane.

2) Comparison with TowerSketch-based methods: We compare **Distributed-software** and **Distributed-hardware** with the methods based on TowerSketch [15], i.e., **Tower-at-core**, **Tower-at-edge**, and **Tower-everywhere**, and present the results in Fig. 8. From the figures we can see that the measurement accuracies of **Tower-at-core**, **Tower-at-edge**, and **Tower-everywhere** are improved comparing the their counterparts using CM sketches as in Fig. 6. This is reasonable, as TowerSketch is dedicatedly designed for skewed network traffic as the one in the MAWI trace.

From Fig. 8, we find that our proposed **Distributed-software** and **Distributed-hardware** methods still significantly outperform **Tower-at-core**, **Tower-at-edge**, and are more accurate than **Tower-everywhere**, despite that TowerSketch has more counters than Distributed Sketch on each individual switch. The reason lies in the following fact: In **Tower-everywhere**, the final estimation result is largely decided by the largest TowerSketch instance along a path, and according to Table IV, on a 5-hop path, the largest

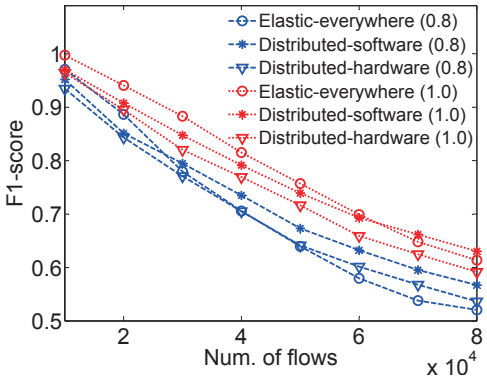


Fig. 10. Comparison of **Elastic-everywhere**, **Distributed-software**, and **Distributed-hardware** on heavy hitter detections when measuring synthesis traces with various number of flows.

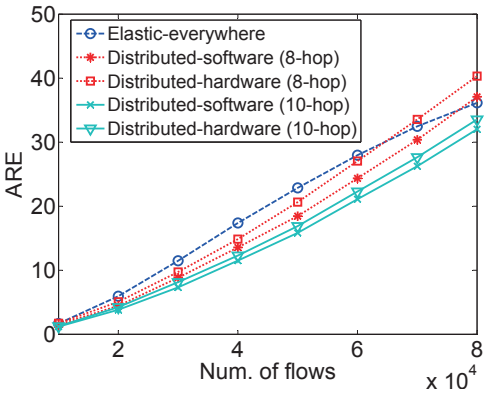


Fig. 11. Comparison of **Elastic-everywhere**, **Distributed-software**, and **Distributed-hardware** for flow size estimations on longer paths when measuring various number of flows.

TowerSketch contains at most $2^{12} + 2^{13} = 12,288$ counters, while in **Distributed-software/hardware**, on average $2 \times (2^8 + 2^9 + 2^{10} + 2^{11} + 2^{12}) = 15,872$ counters could be aggregated from a 5-hop path. Since more counters are employed by **Distributed-software/hardware**, they can provide more accurate measurement results.

3) Comparison with Elastic-based methods: We also compare the measurement accuracies of **Elastic-at-core**, **Elastic-at-edge**, and **Elastic-everywhere** with our proposed **Distributed-software** and **Distributed-hardware** methods, and present the results in Fig. 9.

From Fig. 9, we find that our proposed **Distributed-software** and **Distributed-hardware** methods still outperform **Elastic-at-core** and **Elastic-at-edge**. Only when all the switches host Elastic sketches, **Elastic-everywhere** achieves a higher accuracy, at a cost of a much heavier measurement overhead as we have analyzed in Sec. V-B.

We reveal the reason behind the higher accuracy of **Elastic-everywhere** over **Distributed-software/hardware** as the following: In **Elastic-everywhere**, it is very likely that the largest Elastic sketch along a 5-hop path contains $2 \times 2^{13} = 16,384$ 8-bit counters in its light part, and $1,600$ 32-bit counters in its heavy part according to Table IV; while in **Distributed-software/hardware**, $15,872$ 32-bit counters can be aggregated from a 5-hop path on average. In other words, for most paths in the FatTree network, the largest Elastic sketch along a path has more counters than the ones aggregated from all the switches along the 5-hop path in **Distributed-software/hardware**. As long as the 8-bit counters in the Elastic sketch light part do not overflow (which is indeed the case as our experiments last only 10 seconds), **Elastic-everywhere** achieves a higher measurement accuracy.

D. Outperforming Elastic-everywhere

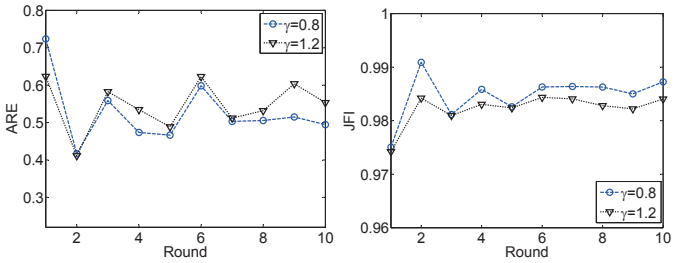
Our investigation shows that **Elastic-everywhere** outperforms **Distributed-software/hardware** on two conditions: 1) The **Distributed-software/hardware** method does not aggregate more counters than the largest Elastic sketch on the path; and 2) the small-sized (e.g., 8-bit) counters in the light part of the Elastic sketch do not overflow. In the following, we use three experiments to show that when either of the two conditions is unsatisfied, **Distributed-software/hardware** will be more accurate.

In our previous experiment on the FatTree network, when the experiment duration is extended to 300 seconds, we find that both **Distributed-software** and **Distributed-hardware** outperforms **Elastic-everywhere** in all the measurement tasks. The reason is straightforward, as a lot of 8-bit counters in the Elastic sketch light part overflow when many mice flows have more than 256 packets in a 300-second duration. We omit the detailed results for brevity.

We then consider a more subtle case in which we synthesize two 10-second packet traces containing $10,000$ to $80,000$ flows following a skewed flow size distribution. We set the distributions' skewness factors as 0.8 and 1.0 respectively, both are smaller than the MAWI trace's skewness factor of 1.47. We repeat the experiment on the FatTree network with the two synthesis traces, and present the heavy hitter detection results in Fig. 10. We focus on the heavy hitter detection as the task is more sensitive to the traffic skewness. As we can see in the figure, under the traffic of 1.0 skewness factor, **Distributed-software** outperforms **Elastic-everywhere** when over $70,000$ flows are measured; and when the traffic is less skewed with a skewness factor of 0.8, both **Distributed-software** and **Distributed-hardware** achieve higher accuracies than **Elastic-everywhere** when measuring over $30,000$ and $50,000$ network flows respectively.

To better understand the experiment result, we investigate the largest Elastic sketches in **Elastic-everywhere**. We find that when there are more flows and flow sizes are less skewed, the heavy part hash table can no longer accommodate all the top-10% largest flows, and some of the heavy-hitter flows have to be counted by the 8-bit counters in the light part, which overflow and result in a reduction on the F1-score.

To enable **Distributed-software/hardware** to aggregate more counters, we consider to apply the **Distributed-software** and **Distributed-hardware** methods on longer network paths and compare it with **Elastic-everywhere**. To this end, we experiment with a linear topology containing 8 or 10 switches as hops. Note that such path lengths are widely observed in inter-data center networks [47]. Switches are configured as in Table IV, and we present how the path sketches of the 8-



(a) (b)

Fig. 12. (a) ARE of flow size estimation and (b) JFI of switches' cell occupation ratios under $\gamma = 0.8$ and 1.2 with network flows arriving and departing dynamically.

and 10-hop paths are mapped to the switch sketches using the interval-based and the approximation mapping methods in the Supplementary Material. Fig. 11 presents the flow size estimation results. Note that for **Elastic-everywhere**, its performance is decided by the largest Elastic sketch along the path, which hardly changes with the path length. From the figure we can see that when the path contains 8 hops, **Distributed-software** is more accurate than **Elastic-everywhere** as long as there are no more than $75,000$ flows, and **Distributed-hardware** is more accurate when the network flows do not exceed $60,000$. When the path contains 10 hops, both **Distributed-software** and **Distributed-hardware** are more accurate than **Elastic-everywhere** all the time.

E. Workload-aware Adjustment

In Sec. III-B, we have proposed a workload-aware method as in (5) for adjusting the bucket mapping intervals by taking the dynamic measurement workload into the consideration. In this experiment, we evaluate the effectiveness of this method. Since **Distributed-hardware** does not enforce the bucket mapping intervals assigned by (5) strictly, and it is only slightly less accurate than **Distributed-software**, we only evaluate **Distributed-software** in this experiment.

The experiment is conducted in rounds with each round lasting 10 seconds. At the beginning of round 0, we feed a total number of $12,000$ flows randomly to the 112 paths in the FatTree network, and apply (4) to compute an initial bucket mapping interval $[\alpha_j, \beta_j]@p$ to assign to each switch per path. A network flow has a duration randomly selected from the distribution in [20] with a mean duration lasting 50 seconds. Given the mean flow duration and according to Little's law, $2,400$ flows arrive to the network per round following a Poisson process.

At the end of each round, in addition to evaluating ARE of the flow size estimation, we also compute the fairness of the measurement workload distributed among the switches with *Jain's fairness index (JFI)*, which is defined as

$$JFI(\rho_1, \cdots, \rho_n) = \frac{\left(\sum_{i=1}^{n} \rho_i\right)^2}{n \sum_{i=1}^{n} \rho_i^2}$$

where ρ_i is the cell occupation ratio of the switch s_i, and n is the total number of the switches in the network. After obtaining ARE and JFI, we then compute a new bucket mapping interval $[\alpha_j, \beta_j]@p$ using (5) with the current workload distribution, report and clear the sketch cells, and proceed to the next round with the new mapping intervals.

Since in (5), we employ a parameter γ for controlling the adjusting aggressiveness, we experiment with two γ values: 0.8 and 1.2, and present the evolvements of ARE and JFI over rounds in Fig. 12. From the figure, we can see that at the end of round 0, the system has relatively a lower accuracy in terms of a larger ARE and a lower level of workload distribution fairness in terms of a smaller JFI, due to the random initial flow assignment. However, after round 1, both accuracy and fairness are improved, and the improvements are preserved during the subsequent rounds under the dynamic arrivals and departures of the network flows. Another observation is that with a less aggressive γ value of 0.8, we can achieve a higher accuracy and a better fairness, while $\gamma = 1.2$ is over aggressive under the network dynamics in this experiment.

We summarize our results as the following.

- The methods based on Distributed Sketch, i.e., **Distributed-software** and **Distributed-hardware**, are lightweight regarding per-packet measurement overhead in terms of memory accesses and hash computation.
- **Distributed-software** and **Distributed-hardware** are significantly more accurate than all the solutions using the classical CM sketch, i.e., **CM-at-core**, **CM-at-edge**, and **CM-everywhere**, and they also outperform the ideal **CM-flow-divide** method, which equally divides flows among the switches along a path.
- Comparing with the solutions based on advanced sketches, **Distributed-software** and **Distributed-hardware** outperform the TowerSketch based solutions, i.e., **Tower-at-core**, **Tower-at-edge**, and **Tower-everywhere**, as well as **Elastic-at-core** and **Elastic-at-edge** all the time, and they are more accurate than **Elastic-everywhere** in many circumstances, such as when measuring traffic in longer durations, or measuring less skewed network traffic, or measuring traffic on longer network paths.
- Our proposed workload-aware method for dynamically adjusting the bucket mapping interval assignments can effectively improve the measurement accuracy and workload distribution fairness under an environment in which network flows arrive and depart dynamically.

VI. Conclusion

In this paper, we presented *Distributed Sketch*, a novel method for per-flow network measurement. Unlike the conventional approaches that place a complete sketch data structure at one individual switch, in Distributed Sketch, each network path is associated with a logical sketch, which is collectively maintained by all the switches along the path; meanwhile, each switch multiplexes its physical sketch to participate in the constructions of the logical sketches of all the paths it belongs to. With Distributed Sketch, switches collaborate to measure network flows, and the network-wide measurement workload is fairly distributed to all the switches in the network. To overcome the problem that existing hardware switches

do not support float-point computation, we proposed an approximation method that involves only integer operations for mapping logical path sketch buckets to physical sketch cells. We implemented a system prototype with P4 on commodity hardware programmable switches, and proved in theory that Distributed Sketch has a lower error bound compared with the conventional sketch placement strategies. We also discussed the challenges in deploying Distributed Sketch in large-scale data center networks, and presented an INT-based solution.

Using a FatTree data center network testbed, we compared Distributed Sketch with conventional sketch placement strategies. We applied both the classical CM sketch as well as the recently proposed advanced sketches of TowerSketch and Elastic sketch with these strategies. We also compared Distributed Sketch with an ideal method that equally divides network flows among the switches along a path. The experiment results showed that Distributed Sketch applying both the interval-based and the approximation mappings outperforms all the alternative approaches, with the only exception when placing an Elastic sketch at every switch in the network. However, we find that when measuring traffic for longer durations, or measuring less skewed traffic, or measuring traffic on longer network paths, Distributed Sketch is still more accurate. Our findings suggested that Distributed Sketch is a lightweight and accurate per-flow measurement method, and is practical to be deployed in data center networks.

We stress that Distributed Sketch is not a replacement of the conventional sketch, as the general ideal of Distributed Sketch, i.e., splitting sketch structure for fairly distributing measurement workload to all the switches across the network, is not just for CM, but can be applied to other sketch data structures as well. For the future work, we aim to develop distributed sketching algorithms based on recently proposed advanced sketches, such as Elastic sketch, TowerSketch, etc. For example, all the switches along a network could collectively construct a logical Elastic sketch or TowerSketch, and each switch maintains a part of the global sketch structure by dividing and mapping the hash spaces.

REFERENCES

[1] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, 2006.

[2] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. SOSR'17*, Santa Clara, CA, USA, Apr. 2017.

[3] R. B. Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, 2020.

[4] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong, "Finding persistent items in data streams," *Proc. VLDB Endow.*, vol. 10, no. 4, 2016.

[5] G. Cormode and S. Muthukrishnan, "What's new: finding significant differences in network data streams," *IEEE/ACM Trans. Netw.*, vol. 13, no. 6, 2005.

[6] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. NSDI'13*, Lombard, IL, USA, Apr. 2013.

[7] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. SIGCOMM'16*, Florianopolis, Brazil, Aug. 2016.

[8] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "SketchVisor: Robust network measurement for software packet processing," in *Proc. SIGCOMM'17*, Los Angeles, CA, USA, Aug. 2017.

[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 Trans. Netw.*, vol. 26, no. 6, 2018.

[10] 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 *Proc. SIGCOMM'18*, Budapest, Hungary, Aug. 2018.

[11] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li, "Diamond sketch: Accurate per-flow measurement for big streaming data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, 2019.

[12] L. Liu, Y. Shen, Y. Yan, T. Yang, M. Shahzad, B. Cui, and G. Xie, "SF-Sketch: A two-stage sketch for data streams," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, 2020.

[13] Y. Zhang, J. Li, Y. Lei, T. Yang, Z. Li, G. Zhang, and B. Cui, "On-Off sketch: A fast and accurate sketch on persistence," *Proceedings of the VLDB Endowment*, vol. 14, no. 2, 2020.

[14] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "CocoSketch: High-performance sketch-based measurement over arbitrary partial key query," in *Proc. SIGCOMM'21*, Virtual Event, USA, Aug. 2021.

[15] K. Yang, Y. Li, Z. Liu, T. Yang, Y. Zhou, J. He, J. Xue, T. Zhao, Z. Jia, and Y. Yang, "SketchINT: Empowering INT with TowerSketch for per-flow per-switch measurement," in *Proc. ICNP'21*, Virtual Event, Nov. 2021.

[16] Y. Li, X. Yu, Y. Yang, Y. Zhou, T. Yang, Z. Ma, and S. Chen, "Pyramid family: Generic frameworks for accurate and fast flow size measurement," *IEEE/ACM Trans. Netw.*, vol. 30, no. 2, 2022.

[17] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. of Algorithms*, vol. 55, no. 1, 2005.

[18] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *ACM SIGMCOMM CCR*, vol. 32, no. 4, 2002.

[19] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. International Colloquium on Automata, Languages, and Programming (ICALP'02)*, Malaga, Spain, Jul. 2002.

[20] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC'10*, Melbourne, Australia, Nov. 2010.

[21] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via yahoo! datasets," in *Proc. IEEE INFOCOM'11*, Shanghai, China, Apr. 2011.

[22] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "FCM-Sketch: Generic network measurements with data plane support," in *Proc. CoNEXT'20*, Barcelona, Spain, Dec. 2020.

[23] "Intel tofino series," accessed on Aug. 20, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[24] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," in *Proc. NSDI'16*, Santa Clara, CA, USA, Mar. 2016.

[25] Q. Huang, S. Sheng, X. Chen, Y. Bao, R. Zhang, Y. Xu, and G. Zhang, "Toward nearly-zero-error sketching via compressive sensing," in *Proc. NSDI'21*, Apr. 2021.

[26] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, B. Tayh, and D. Raz, "Routing-oblivious network-wide measurements," *IEEE/ACM Trans. Netw.*, vol. 29, no. 6, 2021.

[27] D. Harris, A. Rinberg, and O. Rottenstreich, "Compressing distributed network sketches with traffic-aware summaries," *IEEE Trans. Netw. Service Manag.*, 2022. [Online]. Available: https://doi.org/10.1109/TNSM.2022.3172299

[28] R. B. Basat, G. Einziger, and B. Tayh, "Cooperative network-wide flow selection," in *Proc. ICNP'20*, Madrid, Spain, Nov. 2020.

[29] V. Bruschi, R. B. Basat, Z. Liu, G. Antichi, G. Bianchi, and M. Mitzenmacher, "DISCOvering the heavy hitters with disaggregated sketches," in *Proc. CoNEXT'20, poster*, Barcelona, Spain, Dec. 2020.

[30] A. Cornacchia, G. Sviridov, P. Giaccone, and A. Bianco, "A traffic-aware perspective on network disaggregated sketches," in *Proc. MedComNet'21*, Ibiza, Spain, Aug. 2021.

[31] D. Prountzos and K. Pingali, "Betweenness centrality: algorithms and implementations," in *Proc. PPoPP'13*, Shenzhen, China, Feb. 2013.

[32] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. SIGCOMM'15*, London, UK, Aug. 2015.

[33] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," IEEE SA, IEEE Standard IEEE 1588-2019, Nov. 2019. [Online]. Available: https://standards.ieee.org/ieee/1588/6825/

[34] Z. Wei, Y. Tian, W. Chen, L. Gu, and X. Zhang, "DUNE: Improving accuracy for sketch-int network measurement systems," in *Proc. IEEE INFOCOM'23*, New York, USA, May 2023.

[35] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proc. SIGCOMM'10*, New Delhi, Inida, Aug. 2010.

[36] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. SIGCOMM'15*, London, UK, Aug. 2015.

[37] P. Cui, H. Pan, Z. Li, J. Wu, S. Zhang, X. Yang, H. Guan, and G. Xie, "NetFC: Enabling accurate floating-point arithmetic on programmable switches," in *Proc. ICNP'21*, Dallas, TX, USA, Nov. 2021.

[38] Y. Yuan, O. Alama, J. Fei, J. Nelson, D. R. K. Ports, A. Sapio, M. Canini, and N. S. Kim, "Unlocking the power of inline floating-point operations on programmable switches," in *Proc. NSDI'22*, Renton, WA, USA, Apr. 2022.

[39] 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 *Proc. NSDI'21*, Apr. 2021.

[40] "$P4_{16}$ portable switch architecture (PSA)," The P4.org Architecture Working Group, Working draft, Apr. 2021. [Online]. Available: https://p4.org/p4-spec/docs/PSA.pdf

[41] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo, "Explicit path control in commodity data centers: Design and applications," in *Proc. NSDI'15*, Oakland, CA, USA, May 2015.

[42] "bmv2, the behavioral model for P4," accessed on Aug. 15, 2022. [Online]. Available: https://github.com/p4lang/behavioral-model

[43] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. SIGCOMM'08*, Seattle, WA, USA, Aug. 2008.

[44] "Mininet," accessed on Aug. 20, 2022. [Online]. Available: http://mininet.org/

[45] "MAWI working group traffic archive," accessed on Aug. 15, 2022. [Online]. Available: https://mawi.wide.ad.jp/mawi/

[46] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "SketchLib: Enabling efficient sketch-based monitoring on programmable switches," in *Proc. NSDI'22*, Renton, WA, USA, Apr. 2022.

[47] Q. Wang, Y. Tian, X. Yu, L. Ding, and X. Zhang, "Where is the traffic going? a comparative study of clouds following different designs," *IEEE Trans. Services Comput.*, 2022. [Online]. Available: https://doi.org/10.1109/TSC.2022.3182047

Liyuan Gu received the bachelor's degree in computer science from China University of Mining and Technology, Xuzhou, China, in 2021. He is currently pursuing the master's degree with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. His research interest is focused on network measurement.

Ye Tian received the bachelor's degree in electronic engineering and the master's degree in computer science from University of Science and Technology of China (USTC), Hefei, China, in 2001 and 2004, respectively, and the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2007. He joined USTC in 2008 and is currently an Associate Professor with the School of Computer Science and Technology, USTC. His research interests include programmable networks, network and Internet measurements, and network softwarization. He has published over 70 papers and co-authored a research monograph published by Springer. He is the winner of the Wilkes Best Paper Award of Oxford The Computer Journal in 2016. He is a member of the IEEE.

Wei Chen received the bachelor's degree in computer science from University of Science and Technology of China (USTC), Hefei, China, in 2020. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, USTC. His research interests include programmable networks and network measurement.

Zhongxiang Wei received the bachelor's and master's degrees in computer science from University of Science and Technology of China, Hefei, China, in 2020 and 2023 respectively. His research interest is focused on programmable networks. He will join Alibaba in July, 2023.

Cenman Wang received the bachelor's degree in computer science from University of Science and Technology of China (USTC), Hefei, China, in 2022. She is currently pursuing the master's degree with the School of Computer Science and Technology, USTC. Her research interest is focused on programmable networks.

Xinming Zhang received the BE and ME degrees in electrical engineering from China University of Mining and Technology, Xuzhou, China, in 1985 and 1988, respectively, and the PhD degree in computer science and technology from the University of Science and Technology of China (USTC), Hefei, China, in 2001. Since 2002, he has been with the faculty of USTC, where he is currently a Professor with the School of Computer Science and Technology. From September 2005 to August 2006, he was a visiting Professor with the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea. His research interest includes wireless networks, deep learning, and intelligent transportation. He has published more than 100 papers. He won the second prize of Science and Technology Award of Anhui Province of China in Natural Sciences in 2017. He won the awards of Top reviewers (1%) in Computer Science & Cross Field by Publons in 2019. He is a senior member of the IEEE.