Spatiotemporal Sketch Disaggregation: Streaming Analytics with Heterogeneous Resources

Jonatan Langlet
Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
jlanglet@kth.se

Peiqing Chen
Department of Computer Science
University of Maryland, College Park
College Park, MD, USA
pqchen99@umd.edu

Michael Mitzenmacher
Engineering and Applied Sciences
Harvard University
Allston, MA, USA
michaelm@eecs.harvard.edu

Zaoxing Liu
Department of Computer Science
University of Maryland, College Park
College Park, MD, USA
zaoxing@cs.umd.edu

Ran Ben Basat
Computer Science department
University College London
London, United Kingdom
r.benbasat@cs.ucl.ac.uk

Gianni Antichi
Dept. of Electronics, Information and Bioengineering
Politecnico di Milano
Milano, Italy
gianni.antichi@polimi.it

***Abstract*—Streaming analytics are essential in a large range of applications, including databases, networking, and machine learning. To optimize performance, practitioners are increasingly offloading such analytics to network nodes such as switches. However, resources such as fast SRAM memory available at switches are limited, not uniform, and may serve other functionalities as well (e.g., firewall). Moreover, resource availability changes over time due to the dynamic demands of in-network applications.**

In this paper, we propose a new approach to disaggregating data structures, leveraging any residual resources available at network nodes. We focus on sketches, which are fundamental for summarizing data for streaming analytics while providing beneficial space-accuracy tradeoffs. Our idea is to break sketches into multiple 'fragments' that are placed at different network nodes. The fragments cover different time periods and vary in size, and are combined to form a network-wide view of the underlying traffic. We apply our solution to three popular sketches (namely, Count Sketch, Count-Min Sketch, and UnivMon) and demonstrate that we can achieve approximately a 75% memory size reduction for the same error for many queries, or a near order-of-magnitude error reduction if memory is kept unchanged. Further, we demonstrate real-world feasibility through a hardware pipeline for high-speed commodity switches.

***Index Terms*—Distributed data structures, Network monitoring, Computations on discrete structures, Pipeline implementation**

I. Introduction

Streaming analytics tools for tracking and analyzing system behavior, where requests are made at run-time, are important for many applications in databases [32], [52], [73] and networking [35], [71], [72]. For example, in databases, one may wish to know the entries that are most frequently being accessed over a time window [52]; similarly, in networking, one may wish to find the heaviest flow (connection) in the network over a time period [5].

To reduce costs and improve performance, practitioners offload functionalities to network devices, such as programmable network switches [12], [59]. A fundamental challenge is that these have little available memory and limited computation capabilities [13], [14], [58]. The total amount of memory available for line-rate statefulness and packet buffering is as little as O(10MB) [58]. Further, these resources need to be shared across different applications, including streaming analytics [7], [53], [64], security [54], [69], [70], machine learning aggregation [45], [49], [62], storage for database systems [47], [52], [65], and various network functionalities [38], [51], [55], [60]. As different switches may run different applications, and their requirements change dynamically (e.g., based on the traffic or query patterns), the amount of free resources varies greatly and constantly changes, making it hard to deploy any analytics solution that requires a fixed amount of memory.

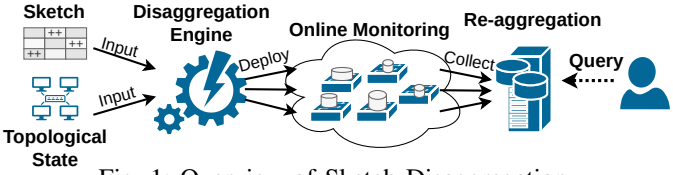


Fig. 1: Overview of Sketch Disaggregation.

In this paper, we propose utilizing switches' current *residual resources* as 'fragments' of a disaggregated data structure. Focusing on sketches, as they are fundamental building blocks for streaming analytics, we present a principled way to disaggregate a single data structure into multiple, dynamically sized pieces that can change in real time (Figure 1). Those fragments can be centrally collected and used together to answer queries similarly to a single data structure.

While there has been a lot of research focused on minimizing the sketch size while optimizing the size-accuracy tradeoff [17], [19], [23], [53], [57], these measure flow statistics at a single node, and their accuracy is thus restricted by the resources available there. However, this overlooks the potential benefits of leveraging residual resources at other nodes. Our key observation is that in networks, the same packet typically traverses multiple nodes (e.g., switches), allowing us to leverage the varying memory across these nodes to im-

prove the accuracy, even when different packets pass through different switches. Although we focus on network analytics, our principles apply to similar settings that naturally appear in distributed databases. For instance, in distributed indexes, where key searches traverse multiple nodes with different searches following different paths, the resources along the path can be effectively harnessed [1], [67].

More concretely, each node fragment provides an estimate, and our methods combine these to produce a single, accurate, estimate from the ensemble along a flow's path. While this is conceptually straightforward, several challenges arise due to inequalities in sketch accuracy caused by heterogeneity: (1) nodes vary in resource availability, (2) nodes experience different traffic volumes, and (3) flows traverse paths of different lengths, resulting in varying numbers of fragments forming estimations. Furthermore, we must address switches' highly restricted computational capabilities, where even basic operations like multiplication and division may be unsupported in the hardware, and minimal computation can be performed per packet due to fast line rates.

In our solution, we consider the time divided into consecutive 'epochs' that provide the granularity at which the user can express its queries. The main innovation is to make epochs divisible, consisting of smaller 'subepochs' after which the fragment is collected. Each flow is measured during a single subepoch per epoch, allowing us to utilize the available resources at a switch more efficiently. This approach trades memory for reporting frequency: when memory is constrained, more subepochs are used to ensure the error is within acceptable bounds; when memory is abundant, fewer subepochs are needed, thereby reducing the collection frequency.

To answer queries, we combine the fragments that measure the flow across the nodes along its path. Estimates from fragments covering shorter subepochs are scaled proportionally to ensure comparability across measurements. The underlying assumption is that, since epochs are short, a flow's rate remains relatively uniform within an epoch, allowing accurate estimations to be inferred from a single subepoch.

We realize this technique in DiSketch – a system that efficiently disaggregates sketches to leverage all residual resources while optimizing accuracy. We demonstrate the generality of DiSketch, we apply it to three popular sketches: Count Sketch [17], Count-Min Sketch [23], and UnivMon [53]. We compare DiSketch against traditional sketch deployments (which we refer to as "aggregated") as well as versus DISCO [15], a recent sketch disaggregation technique. Extensive experiments show that our solution is highly memory-efficient, achieving comparable accuracy to DISCO while using only 25% of the memory, or reducing error by nearly an order of magnitude under the same memory constraints.

II. Motivation

Recent advancements in streaming analytics have led to the development of compact sketch-based solutions, as described in several works [27], [39]–[41], [53], [57], [68], [74]. These studies have demonstrated the feasibility of implementing these structures within the constraints of modern switches,

Application	Examples	Memory
Basic Packet Processing	switch.p4 [56]	30%
Security	Ripple [70], Jaqen [54], Bedrock [69]	+10-50%
Machine Learning	SwitchML [62], ATP [45], THC [49]	+10-40%
Storage/Database	DistCache [52], NETACCEL [47], Cheetah [65]	+20-30%
Networking	SilkRoad [55], HPCC [51], SwRL [38], Sailfish [60]	+5-40%

TABLE I: The on-switch memory cost of network functions.

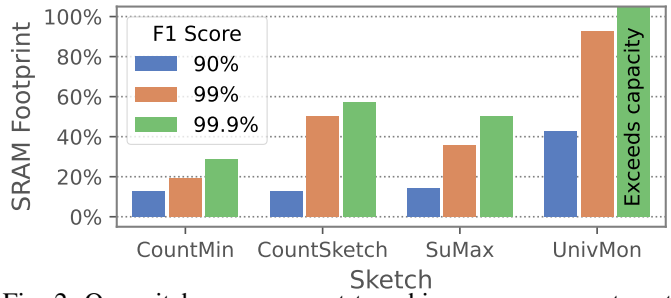


Fig. 2: On-switch memory cost to achieve an accuracy target while monitoring 30s of real-world backbone traffic.

achieving minimal estimation errors [40], [41], [57], [74]. However, sketches are memory-intensive data structures, and their accuracy directly depends on the amount of memory dedicated to them. As we show later, deploying sketches on switches alongside other functionality competing for limited memory can result in significant accuracy degradation.

Our survey of recent literature on in-network computing, which encompasses applications ranging from security to machine learning acceleration and network functions, reveals a significant demand for switch resources. For instance, basic packet processing capabilities alone, such as L2/L3 forwarding, may consume up to 30% of a switch's memory, as shown in Table I. Including additional functionalities further reduces the available memory for sketches.

To quantify the impact of this resource competition, we analyzed the SRAM memory requirements of both established (i.e., Count Sketch [17], Count-Min-Sketch [23]) and recently proposed sketches (i.e., UnivMon [53], SuMax [74]) for heavy hitter detection over a 30-second window (similar to previous works [53], [68], [71]), using real-world traffic traces from an Internet backbone [16]. Our findings, depicted in Figure 2, show that to achieve a 99% F1 score, the memory demand of sketches ranges from 20% to nearly 90% of a programmable Tofino switch's SRAM [58], underscoring the challenge of maintaining high monitoring accuracy while co-locating sketches with other functions. A high monitoring accuracy is essential as the base of responsive diagnosis [50], [63], and a 99% accuracy already indicates a significant amount of incorrect flow classification. Improved classification accuracy requires even more memory and can sometimes exceed the switch's memory capacity, even in isolation.

Moreover, even if a sketch is not co-located with any other in-network function, it is worth noting that the amount of traffic observed by each switch significantly differs, and there can be orders of magnitude different volumes even for switches with the same logical purpose (e.g., edge switches in a datacenter) [8], [26], [61]. As a consequence, deploying the measurement on a cut in the network topology (e.g., all

edge switches [36]) results in varying degrees of accuracy even when all switches have the same memory allocated.

Acknowledging these challenges, disaggregating sketches across multiple switches emerges as a promising solution for utilizing network-wide resources.

III. Sketch Disaggregation

We consider disaggregating *sketches*, which, for our purposes, are viewed as a matrix structure in which each cell is an identical copy of a simpler data structure, usually a counter. We further assume that when an element (e.g., a packet) is inserted into the data structure, its key (e.g., flow ID) is mapped via hashes into one or more cells in each row, and the cells are updated appropriately. In what follows we assume one cell is updated in each row, and that cell is chosen uniformly in each row by the hash function. Examples of such sketches include ones for frequency estimation [17], [24], [28], [41], [74], set membership [9], [27], [29], sparse recovery [31], [42], frequency moments estimation [4], [18], [53], entropy estimation [20], [37], [53], and ℓ_p samplers [21], [22]. These sketches' cells are exported and reset periodically, to prevent the buildup of stale data and subsequently reduced estimation accuracy. The period between resets is referred to as the *epoch*.

Understanding the challenges of sketch disaggregation across multiple nodes begins with a depiction of a datacenter network's architecture. A classic Fat-Tree topology, commonly referenced in literature and employed in real-world deployments, is illustrated in Figure 4. This topology highlights the existence of numerous paths between any two end-nodes, with the number of hops varying based on the nodes' locations. For example, flows between (A) and (B) traverse just a single switch, while any path between (A) and (D) contains five switches. Sketch disaggregation is then the process of dividing a sketch across network paths, where each network node hosts a fragment of the sketch. As in traditional deployments, we assume that at the end of each epoch, per-node data structures are sent for analysis to a central server called the controller. After collection and aggregation, per-path fragments can be queried together to answer queries similarly to traditional aggregated sketches. With this in mind, there are two natural approaches to disaggregating sketches: *per-column* and *per-row* disaggregation.

In per-column disaggregation (Figure 3a), each network hop would host all sketch rows of the sketch matrix, but only a part of the columns. Keys are still mapped to one cell in each row, selected uniformly at random by a hash function. This requires that both the path length and fragment widths have to be known by all sketch nodes when a packet is traversing the network, which is costly. That is because a switch needs to know the indices of columns it holds, as well as the total number of on-path columns, since the hash functions need to output a column index. Previous work on sketch disaggregation attempts to solve this through lookup tables in each fragment containing entries for every network path going through a switch [34], posing a scaling issue for large networks. For example, on a k-ary Fat-Tree, each edge switch needs to store information about $k^3 \cdot (k-1)/8$ paths (it has $k/2$ options for each of the aggregate and core switches, $k-1$ for the aggregate switch on the way down and another $k/2$ for the last edge switch). This means that even for moderately sized networks such as $k = 28$, we would require information for about 74 thousand paths, leaving less memory for the sketch itself and undermining the original purpose of the disaggregation.

In contrast, in per-row disaggregation (Figure 3b), each node hosts a single row in each fragment, occupying all sketch-allocated memory. Each fragment is then equivalent to an independent sketch row, so they can function in isolation, and there is no need for a lookup table as with per-column aggregation. Note that, unlike standard sketches, the differences in row sizes due to memory lead to an "irregular" shaped matrix, which makes it harder to aggregate the results into a single accurate estimate.

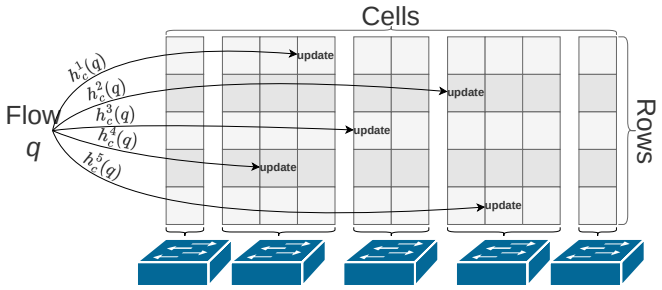
To illustrate the disaggregation overheads, we implemented state-of-the-art per-row (DISCO [15]) and per-column (Distributed Sketch [34], lookup table excluded from cost) disaggregated sketches on a programmable Tofino switch [58]. Figure 5 presents these overheads, comparing the total resources required along a path with those of a traditional Count-Sketch.

While per-row disaggregation generally offers better resource efficiency than per-column, producing accurate estimates from this kind of data structure is particularly challenging when the switches differ significantly in the memory amount and traffic volume.

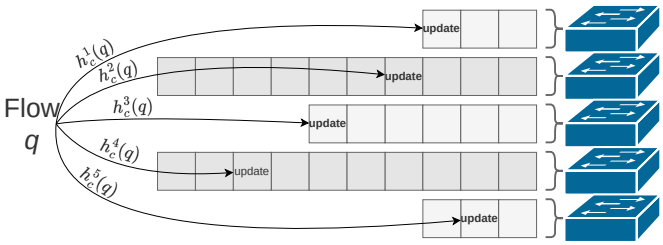
Before we describe our solution, we discuss the main challenges of disaggregation.

Challenge 1: Nodes across the network can have varying resources, a result of deploying distinct in-network functions at different switches and possibly other causes of switch heterogeneity. Some functions, such as security mechanisms, may be more suitably deployed at switches near endpoints [62], [65], [69], while others fit better within the network core [60]. This diversity leads to a heterogeneous use of memory, ruling out per-column disaggregation due to the high computational overhead and substantial memory requirements for stateful per-flow counter allocation. Per-row disaggregation in highly heterogeneous deployments can experience accuracy degradation, where tiny fragments introduce significant errors to the composite sketch. Alternatively, these fragments become essentially negligible when assigned an importance proportional to their relatively high error.

Challenge 2: Traffic volume can vary significantly across nodes, stemming from the design of datacenter networks and their traffic patterns. For example, the Fat-Tree topology facilitates massive multi-path routing [2], yet despite load-balancing efforts, imbalances persist [3], [33], [43], [44], [66]. Studies have shown that much traffic remains local, with only a fraction traversing the entire datacenter [61]. Consequently, switches near end-hosts experience higher traffic volumes than those at the core, affecting the accuracy of sketch fragments under heavy loads.



(a) Per-column disaggregation. Fragments host full-depth sketches.



(b) Per-row disaggregation. Fragments host one row each.

Fig. 3: Visualization of sketch disaggregation directions.

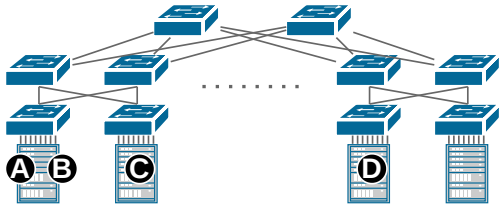


Fig. 4: A Fat-Tree Topology. Packets traverse 1 (e.g., A-B), 3 (e.g., A.C), or 5 (e.g., A-D) switches.

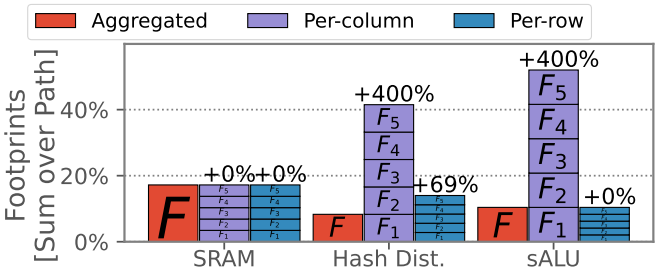


Fig. 5: The disaggregation direction has a significant impact on computational resources. Shown here are full-path footprints in a Tofino programmable switch, split per switch.

Challenge 3: The path lengths of different flows vary, influenced by the datacenter's network topology and traffic distribution. While some traffic remains local, affecting only a few nodes, other flows span across the network. This variance means that some traffic benefits from more extensive observation by multiple fragments, whereas others do not. For flows traversing only a single fragment, the accuracy degradation is akin to using a one-row sketch, which can yield significant inaccuracy. To demonstrate this effect, we deploy DISCO, a per-row disaggregated count sketch, in a Fat-Tree topology using the same experimental parameters as further down in Section VI-A. We show a breakdown of the per-path-length's impact on heavy hitter detection in Figure 6. Notice the significant impact that the path length has on the estimation accuracy, with queries regarding single-hop flows being highly inaccurate.

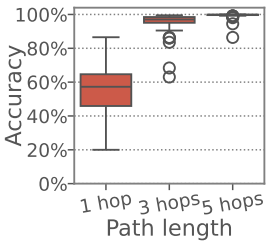


Fig. 6: Path-lengths' impact on per-row disaggregation.

In the following section, we introduce a general technique that can be used to deploy per-row disaggregated versions of sketches to achieve highly accurate estimations.

IV. Spatiotemporal Disaggregation

Here, we introduce our spatiotemporal disaggregation solution for resource-scarce sketch disaggregation.

In heterogeneous network environments, where nodes differ in resource capacity and traffic load, sketch accuracy can be significantly compromised due to smaller and/or overloaded fragments that introduce substantial estimation errors. In some cases, it may be more effective to entirely disregard these less reliable fragments and instead rely on the fewer, larger fragments along the network path. This is contrary to our goal of utilizing resource gaps and hints at a deeper issue with inefficient resource leverage.

An appealing solution is flow-level sampling, where fragments track a subset of flows whose size is determined by the fragment's memory and expected traffic load. Larger or less loaded fragments can handle more flows, while 'congested' fragments require more restrictive sampling. This approach allows even small fragments to provide useful insights on the flows that they track while larger fragments provide a more complete flow coverage. However, naive sampling risks leaving some flows untracked when all their on-path fragments fail to sample them. Additionally, it introduces inconsistent measurement inaccuracies, with some flows favored over others even when they traverse the same path. This inconsistency can undermine the overall reliability of the sketch.

To mitigate this unreliability, we propose a novel temporal sampling technique where a subset of flows are actively monitored at a time during dynamically defined periods. This way, we optimize resource utilization and accuracy while ensuring each fragment provides equal coverage for all flows.

We start with a brief overview of our solution, followed by a more detailed description in subsequent sections. Please refer to Figure 7 for a visual overview of our terminology.

a) Subepoching (§IV-A): : Each fragment divides its sketching epoch into n subepochs, where n is chosen per fragment. A flow is only monitored during one subepoch in each on-path fragment, hence a fragment only tracks approximately $\frac{1}{n}$ of the flows at a time. Counters are exported and reset after each subepoch to allow for central querying. We elaborate on subepoching in Section IV-A.

b) Error Equalization (§IV-B): : A network-wide error target is chosen, which is used to homogenize errors across fragments to facilitate network-wide querying. Fragments accomplish this by estimating their local errors at the end of

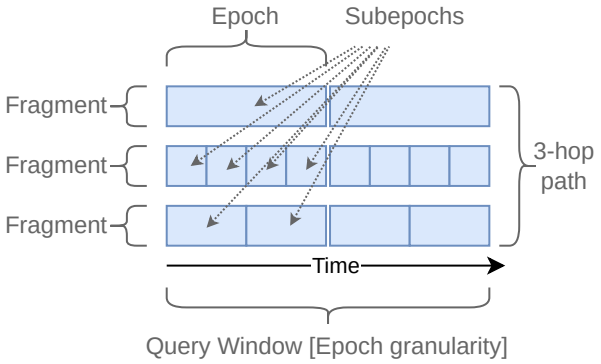


Fig. 7: Sketching epochs are divided into subepochs. Fragments are single-row sketches residing on different switches. Queries are executed against composite sketches, comprising all relevant subepoch records.

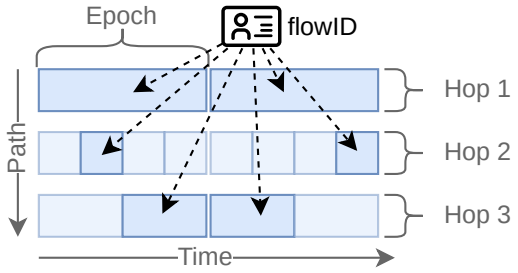


Fig. 8: Flows map into one subepoch per epoch in each hop, leading to temporal sampling.

each epoch and using this estimation to adjust the number of subepochs for the upcoming epoch. This way, all fragments aim to deliver similar error bounds in the upcoming epoch. We elaborate on error equalization in Section IV-B.

c) Central Queries (§IV-C): : We consider queries that analyze the traffic in one or several adjacent sketching epochs. These queries are executed against a specific key (e.g., a network flow, port, or host) or the aggregate network (e.g., in entropy estimation). The exact supported queries depend on the capabilities of the deployed sketch that is being disaggregated. At query time, collected data from relevant fragments are used to compile a virtual composite sketch that covers the queried data streams. We elaborate on composite sketch compilation and querying in Section IV-C.

A. Subepoching

We now provide a detailed description of subepoching, which is the foundation of our temporal sampling technique. For ease of exposition, we first assume that all flows follow the same path length; in Section IV-D, we discuss optimization that improves the accuracy when the path lengths differ.

Sketch fragments dynamically divide their epochs into a power-of-two number of subepochs (n) so that $n = 2^x$ for some $x \in \mathbb{N}$. Specifically, during epoch E, each fragment F divides the global epoch duration into n_E^F equal-sized subepochs. To simplify the notation, we will from now on omit the F superscript from variables when there is no fragment ambiguity. For simplicity and efficiency of implementation, we hereafter assume that n is a power of two.

Subepoching allows fragments to perform temporal flow-level sampling while simultaneously guaranteeing flow queryability within each query window. This is achieved by mapping each flow to one subepoch per fragment in each epoch, resulting in each subepoch monitoring a distinct subset of flows. We visualize this in Figure 8. Subepoch mapping for epoch E is achieved through a fragment-specific hash function $s_E : \mathbf{q} \to \{0, 1 \ldots, n_E - 1\}$, where $\mathbf{q}$ is the set of flows that traverse the fragment during a subepoch. That is, each flow q is monitored within subepoch $s_E(q)$.

A subepoch record is generated and exported at the end of each subepoch, containing all information required to centrally query the data stream (see details in Section IV-C).

While initiating a new epoch, fragments individually replace their hash functions to prevent persistent collisions. Additionally, the number of subepochs n is recomputed based on fragments' estimated performances, to equalize the network-wide error bounds.

B. Error Equalization

In most cases, sketches provide accuracy guarantees based on an analysis in which a basic data structure (e.g., a sketch row in count min and count sketches) provides the desired precision with a constant probability (e.g., 3/4). Merging the estimates of independent repetition of this structure (e.g., using min in the Count-Min sketch or median in the Count Sketch) then amplifies the success probability as desired. For simplicity, and because we find it effective, we follow the same rationale – we aim for different fragments to yield estimates with similar errors, allowing us to amplify the success probability through merging.

With this in mind, we consider the Count-Min and Count Sketches as two examples. The standard analysis of the Count-Min sketch looks at the expected noise that other flows impose onto the queried flow's counter. Assuming that the hash function is pairwise independent, any other flow increases the counter with probability $1/w$, i.e., the expected noise is bounded by $\frac{\sum_{k=1}^{|\mathbf{q}|} f_k}{w}$. By Markov's inequality, this gives

$$\Pr\left[\hat{f}_k - f_k \geq 4\frac{\sum_{k=1}^{|\mathbf{q}|} f_k}{w}\right] \leq \frac{1}{4}\,. \tag{1}$$

Similarly, following the standard analysis of Count Sketch [17], [46], each fragment of width w has a frequency estimate $\hat{f}_k$ for every flow $q_k \in \mathbf{q}$ with expectation f_k and variance at most $\frac{\sum_{k=1}^{|\mathbf{q}|} f_k^2}{w}$. Using Chebyshev's inequality, this implies that:

$$\Pr\left[|\hat{f}_k - f_k| \geq 2\sqrt{\frac{\sum_{k=1}^{|\mathbf{q}|} f_k^2}{w}}\right] \leq \frac{1}{4}\,. \tag{2}$$

Therefore, based on the switch's available space, we aim to size the subepochs such that the noise bound roughly matches a target quantity ρ, which we loosely refer to as a fragment's probabilistic error bound (PEB). The magnitude of

a fragment's estimation errors is linked to ρ, and based on the above, we set:

$$\rho = \begin{cases} \sqrt{\frac{\sum_{k=1}^{|\mathbf{q}|} f_k^2}{w}}, & \text{if CS} \\ \frac{\sum_{k=1}^{|\mathbf{q}|} f_k}{w}, & \text{if CMS} \end{cases} . \quad (3)$$

To roughly equalize the error across fragments, we define a network-wide *target PEB* ρ_{target}, representing the desired PEB in all fragments' subepoch records. Each switch, knowing its space constraints, then attempts to meet this target.

For our purposes, we can think of sketch fragments as singe-row sketches. Given that sketches aim to *estimate* the underlying frequency vector $\mathbf{f}$, we can use the fragment's counters $c_i \in \mathbf{c}$ as an approximation of the frequency vector. We can then modify Equation 3 to calculate an estimated subepoch PEB $\widehat{\rho} \approx \rho$ from the sketch counters:

$$\widehat{\rho} = \begin{cases} \sqrt{\frac{\sum_{i=1}^{w} c_i^2}{w}}, & \text{if CS} \\ \frac{\sum_{i=1}^{w} c_i}{w}, & \text{if CMS} \end{cases} . \quad (4)$$

Let $\widehat{\rho}_{E,s}$ be the estimated PEB from subepoch s in epoch E. An epoch's average error bound is then estimated as:

$$\overline{\widehat{\rho_E}} = \frac{\sum_{s=0}^{n_E-1} \widehat{\rho}_{E,s}}{n_E} . \quad (5)$$

We aim to equalize $\overline{\widehat{\rho_E}}$ across the network, so that $\overline{\widehat{\rho_E}} \approx \rho_{target}$ for all fragments and epochs. Recall that $n \approx FlowSamplingRate^{-1}$ within each subepoch, which leads to $\rho \propto \frac{1}{n}$. Further, we are assuming that traffic patterns are relatively stable between consecutive epochs, hence $\overline{\rho_{E+1}} \approx \overline{\widehat{\rho_E}}$ for each epoch E. Following these assumptions, fragments autonomously adjust n to approach ρ_{target}:

$$n_{E+1} = \begin{cases} 2n_E, & \text{if } \overline{\widehat{\rho_E}} > 2\rho_{target} \\ \max(1, \frac{n_E}{2}), & \text{if } \overline{\widehat{\rho_E}} < \frac{\rho_{target}}{2} \\ n_E, & \text{otherwise} \end{cases} . \quad (6)$$

ρ_{-1} is not defined, so we initiate the first epoch with a likely suboptimal $n_0 = 1$. We recommend computing a moving n_{E+1} as we do in Equation 6 instead of independently calculating it (using the equation $n_{E+1} = 2^{\lfloor \max\left(0, \log_2 \frac{\overline{\widehat{\rho_E}}}{\rho_{target}}\right) \rceil}$) to reduce the effect of $\widehat{\rho}$ outliers. Calculating a moving n showed a slight empirical advantage in our simulations.

a) ρ_{target} Selection: One might be tempted to set an extremely low ρ_{target} for the network, to increase the estimation accuracies. Unfortunately, ρ_{target} only states the error bound for queries *within a single subepoch*, but we want to optimize for the estimation accuracy during a query window, i.e., a set of contiguous epochs. The epoch estimation is, in cases of temporal blind spots, an extrapolation from the mean subepoch estimation and is therefore highly dependent on how representative the tracked flow pattern is to the full epoch. Therefore, we want to minimize the final estimation error $\epsilon = \epsilon_{subepoch} + \epsilon_{extrapolation} \approx \rho + \epsilon_{extrapolation}$. Unfortunately, $n \propto \frac{1}{\rho_{target}}$, i.e., decreasing ρ_{target} generally increases fragments' n, leading to shorter flow-tracking time windows and subsequently increased extrapolation errors $\epsilon_{extrapolation}$.

The extrapolation accuracy is a function of n and the distribution of inter-packet arrival times for flows (i.e., flow burstiness patterns). For instance, the frequency estimations for highly bursty flows are unlikely to be accurately estimated from a sampled time window, while uniformly transmitting flows can be accurately extrapolated. ρ_{target} should be selected following an analysis of a network's typical traffic patterns.

This error equalization results in all fragments' subepoch records having similar expected error bounds, regardless of size or traffic load, facilitating accurate queries.

b) The UnivMon Sketch: UnivMon has a data structure that consist of multiple count sketches called 'levels'. According to the above, we set each fragment to contain all levels, all of which have the same width (as in the paper [53]) and the same subepoch hash.

C. Central Querying

Each fragment exports a subepoch record R at the end of each subepoch, which is sent for central collection.

A record is defined as $R = (F, E, S, n, \mathbf{c}, \mathbf{h})$ where:

- F is the fragment that the record is from,
- E is the epoch number,
- S is the subepoch number,
- n is the number of subepochs that F used in E,
- $\mathbf{c}$ are the counters from the end of the subepoch,
- $\mathbf{h}$ are the hash functions used during sketching.

To refer to a field, e.g., F, inside of a record R, we use $R.F$. Each of these records R is added to the set of collected records $\mathbf{R}$ which forms the basis for network-wide queries.

A key-based query is defined as $Q = (\theta, \tau, \kappa)$ where:

- θ is the query type (e.g., flow frequency or data volume),
- $\tau = [T_{start}, T_{end})$ is the query time window,
- κ is the queried key (e.g., a flow).

The set of supported query types and keys depends on the capabilities of the deployed sketches.

Queried time windows are at the granularity of sketching epochs so that $Q.\tau$ aligns with a set of contiguous epochs $\mathbf{E}_Q$. Each epoch is queried individually to yield a list of per-epoch query outputs $O_E \in \mathbf{O}$ for each $E \in \mathbf{E}_Q$. These per-epoch outputs are merged to craft a final (per-key) query output O_Q, for instance through $O_Q = Sum(\mathbf{O})$ or $O_Q = Average(\mathbf{O})$ depending on the query.

Each epoch E is queried as follows:

a) Step 1 - Retrieve relevant records:: Let $\mathbf{R}_Q^E$ be the set of subepoch records that form the base for query Q in epoch E. The specific set depends on the query, but all records have E as their epoch number.

Many queries, such as queries against a specific flow $Q.\kappa$ require knowledge of the network path of the flow $\mathbf{P}_\kappa$. We assume that the path for flows is known or computable. This assumption is standard and can be achieved, e.g., if ECMP-based (hash-based) load balancing is used since we can recompute the hashes [30]. This path is used to restrict $\mathbf{R}_Q^E$ so that $R.F \in \mathbf{P}_\kappa$.

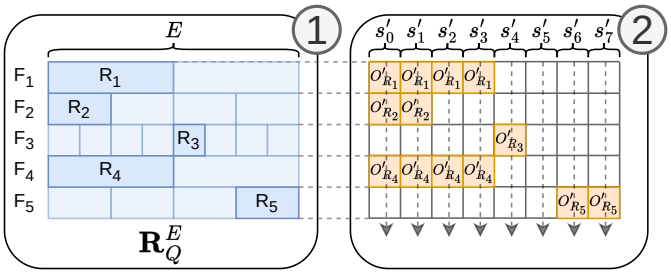


Fig. 9: Centralized querying process. (1) Relevant records from on-path fragments are retrieved, with subepochs that sampled the target key during the query window selected. (2) Subepochs and estimations are normalized and merged to generate the query result.

Further, only records from subepochs that sampled κ are selected. This is done through the key-to-subepoch mapping hash function $s_E^F(\kappa)$, which is computed for each fragment $F \in \mathbf{P}_\kappa$, so that $R.S = s_E^F(\kappa)$.

We now have a set $\mathbf{R}_Q^E$ containing all records within E that form the basis for query Q.

b) Step 2 - Query the Records:: All records are normalized into equal-length subepochs to facilitate querying of varying-length subepoch records. For this, we find n_m, the largest n for any record in $\mathbf{R}_Q^E$, which is the number of subepochs that records should normalize into.

Each record $R \in \mathbf{R}_Q^E$ is queried individually as single-row sketches to retrieve their raw estimations O_R. For instance, in CMS, $O_R = c_i \in R.\mathbf{c}$, where $i = h_c(\kappa)$ and $h_c \in R.\mathbf{h}$ is the indexing hash function used at sketching-time. These estimations are divided into $N_R = \frac{n_m}{R.n}$ smaller estimations $O'_R = \frac{O_R}{N_R}$, one per normalized subepoch that overlap with the record's subepoch.

To estimate the frequency for a key (e.g., flow) within an epoch, we first estimate its statistic within each normalized subepoch. Namely, for a given normalized subepoch S', we consider all records the key was mapped to that include S'.

For example, consider a case where the queried key traversed five fragments and is recorded in subepoch records $R_1, \ldots, R_5$ as shown in Figure 9 Step 1. In particular, this means that $n_m = 8$ and we thus have eight normalized subepochs. The estimate in the first two normalized subepochs is then based on the normalized estimations $O'_{R_1}, O'_{R_2}, O'_{R_4}$, the third is based on just O'_{R_1} and O'_{R_4}, etc. The method for combining the estimates from each normalized subepoch depends on the sketch itself (e.g., the minimum of their estimates in Count-Min, or the median in Count Sketch).

We note that there can be some normalized subepochs without any estimates (that is, a temporal 'blind spot' for this key). In this case, we use the mean of the estimates of the other normalized subepochs. For example, in Figure 9 Step 2, since we have no measurements of the key size during the sixth normalized subepoch, we use the average of the other seven normalized subepochs' estimates. These temporal blind spots are the cause of extrapolation errors when the mean is a poor estimator for the key's true statistic during this time window.

Finally, the sum of the normalized subepoch estimates serves as a cumulative estimate over the entire epoch and is the epoch's output.

D. Enhanced Single-hop Sketching

As previously discussed in Section III and shown in Figure 6, the accuracy of flow estimations is closely tied to the number of fragments traversed (i.e., to the path length). Flows that traverse a single fragment, notably, have much worse performance, as they do not obtain the benefit from multiple estimates, and this is especially damaging when that fragment contains relatively few counters. We, therefore, offer a mitigation strategy that aims to enhance the accuracy of flows that traverse just a single network hop, with only mild costs in accuracy for other flows. For this, we assume that fragments can identify single-hop flows at sketching-time[1].

We arrange that these flows are monitored not in one, but two subepochs within each epoch. This is done by modifying the key-to-subepoch mapping to compute the two subepochs $S_{E,0} = s_E(\kappa)$ and $S_{E,1} = s_E(q) + \frac{n_E}{2} \bmod n_E$. Thus, for $n_E \geq 2$, single-hop flows are monitored during two subepochs, one in the first half and one in the second half of the epoch. If $n_E = 1$, then this mitigation technique is not applicable but also suggests that the single-hop fragment is large enough to deliver suitably accurate estimations on its own. Note that this technique requires a query to use two subepoch records from each queried epoch for such flows.

Although this mitigation technique enhances the accuracy of single-hop flows, it simultaneously imposes (approximately) twice as many counter increments for those same flows, which increases the fragment's ρ. Therefore, we expect slightly increased errors in queries against other flows. We evaluate this effect in Section VI-D.

V. Implementation

We provide two open-source implementations of DiSketch: a modular software simulator and a hardware prototype. The simulator supports flexible experimentation with disaggregated sketches across diverse topologies and configurations. The hardware prototype targets P4-programmable switches on commodity hardware, demonstrating feasibility at line rates of multiple terabits per second with minimal resource overheads. Spatiotemporal disaggregation is optimized for the constraints of the Protocol-Independent Switch Architecture (PISA) [10], [11], enabling all packet processing to remain in the high-speed data plane.

A. Hardware Prototype

To demonstrate the hardware efficiency of our approach, we implement DiSketch on a P4-programmable switch. This implementation is verified on a Tofino2 switch, capable of 6.4Tbps network traffic. Our prototype is lightweight, comprising 850 lines of code split between the P4 data plane and the Python control plane.

Subepoch membership is computed entirely in the data plane using a bitslice of the switch-internal timestamp.

[1]Single-hop flows can be identified, for example, when a switch is neither receiving nor sending a packet to another switch, in cases where all network switches contain a sketching fragment.

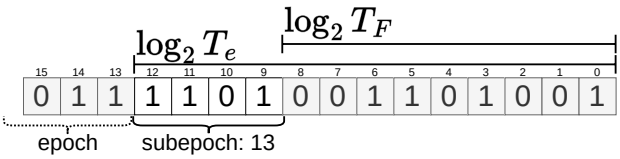


Fig. 10: Inferring the subepoch from a timestamp.

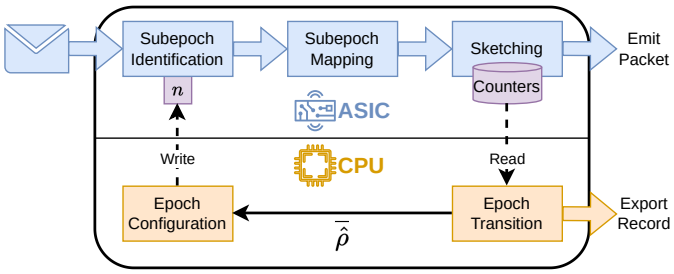


Fig. 11: An overview of the on-switch placement of DiSketch components. The high-speed ASIC does online sketching, while the slower on-switch CPU handles epoch transitions.

Assuming epoch lengths and subepoch counts are powers of two, the intra-epoch offset is encoded in the lower $\log_2(\texttt{epoch\_length})$ bits of the timestamp (Figure 10). The currently active subepoch corresponds to the highest $\log_2(n)$ bits of this intra-epoch window, where n is the number of subepochs in the fragment.

To determine whether a flow is active in the current subepoch, we hash the flow ID using a native CRC8 primitive and XOR the result with the intra-epoch timestamp. If the top $\log_2(n)$ bits of the result are all zero, the flow is selected for tracking. This bitwise match is implemented using a single-entry ternary match table, where the bitmask encodes the current value of n through its number of leading ones. This design supports low-latency reconfiguration and incurs minimal table space.

Epoch configuration is too complex to adhere to the strict PISA requirements and is deferred to the local control plane and invoked only at epoch boundaries, as shown in Figure 11. To avoid counter resets, we track deltas: the controller queries raw counter values and subtracts the previously recorded state to infer per-subepoch activity. The only controller-ASIC communications required are counter exports and occasional bitmask updates when the number of subepochs changes.

Figure 12 shows the data plane resource overheads of DiSketch when applied to a single-row disaggregated Count-Min Sketch with 131K counters. The dominant resource, SRAM, sees a negligible increase of just 0.1%. Most of the added cost comes from the hash-based subepoch selection logic, which introduces modest overheads to hash distribution units, TCAM, and ternary result buses. Overall, the impact on hardware resources remains low and is well justified by the substantial accuracy improvements enabled by spatiotemporal sampling.

VI. Evaluation

Topologies. We simulate two smaller data center topologies: a $k = 2$ Fat-Tree network with four core switches (20 switches

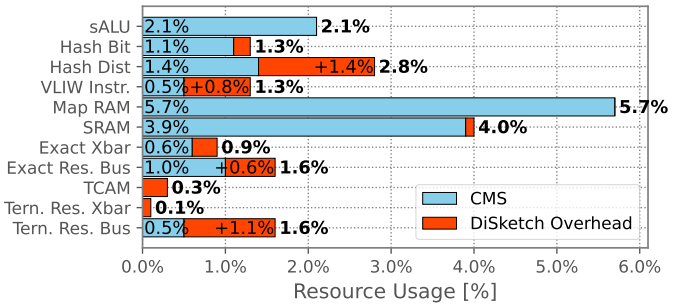


Fig. 12: Hardware resource cost of a single-row disaggregated CMS with and without DiSketch enhancements.

in total), and a SpineLeaf network (12 switches in total)[2].

In these networks, we evaluate two switch memory size distributions: *Homogeneous* and *Heterogeneous*.
The heterogeneous scenario allocates pseudo-random memory sizes for per-switch sketches. These memories are generated with a pre-defined average memory size and inter-switch heterogeneity, designed to correspond with a selected heterogeneity level[3]. The generated memories are randomly distributed to switches, not considering their topological position. As a default, we use an arbitrary heterogeneity level of gini $= 0.4$, resulting in notable memory size variation between switches. An example memory distribution for five switches with gini $= 0.4$ can be: [10%, 30%, 100%, 160%, 200%], i.e., fragment-sizes ranging from 10% to 200% of the base memory size.
The homogeneous scenario allocates exactly the base memory size for all switches, resulting in a heterogeneity level of 0.

Sketches. We are evaluating spatiotemporally disaggregated instances of Count Sketch (CS), Count-Min Sketch (CMS), and UnivMon (UM)[4]. To simplify the discussion, we name applied versions of our solution as *DiSketches* (Disaggregated Sketches), i.e., DiSketch-CS, DiSketch-CMS, and DiSketch-UM. For comparison, we also evaluate aggregated sketches on core switches in the network, and DISCO [15] deployed network-wide. All sketches/fragments utilize all available memory of their respective switches.

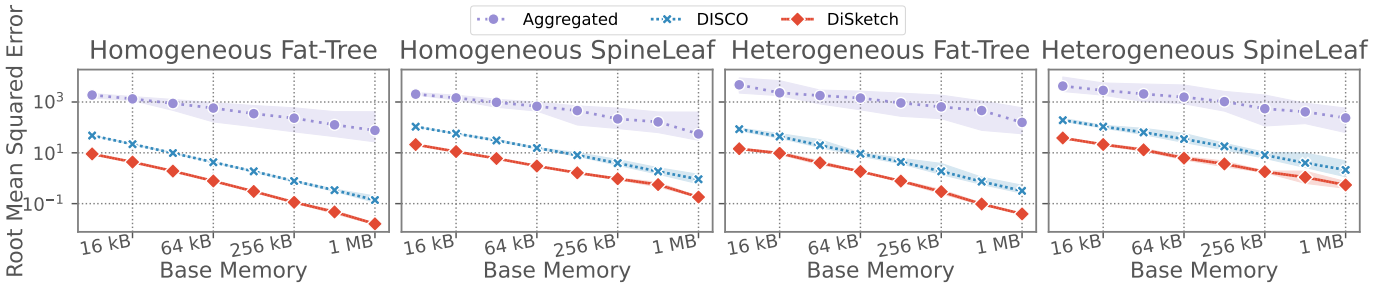
Traces. In this evaluation, we use the real-world CAIDA-NYC Equinix packet trace [16], recorded at a backbone link in 2019. If nothing else is stated, then we replay ∼5 seconds of traffic (∼2M packets, covering ∼200K flows). We only have access to traffic recorded at a single network link, and we map IP addresses in traffic uniformly at random to hosts in our network to simulate network-wide communication, while omitting flows where both the source and destination map to the same host.

DiSketch converges on an optimal epoch configuration over time, and we split the query window into arbitrary 32 epochs to allow DiSketch to converge. We do, however, include the

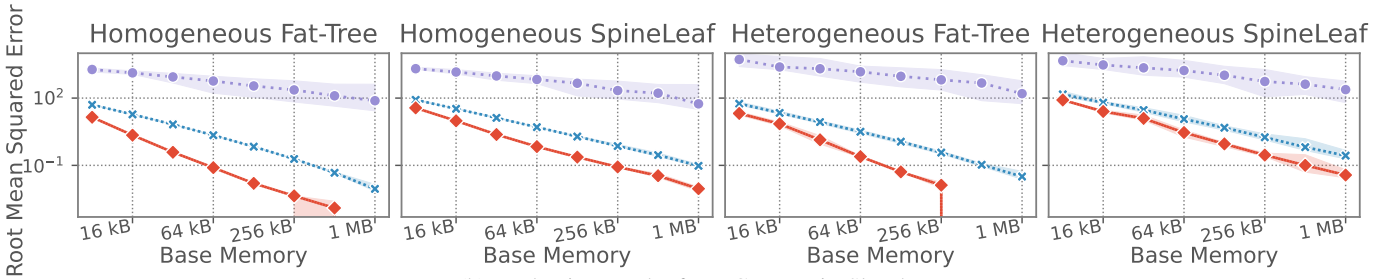
[2]The simulated networks are small in size due to computational restrictions. For context, a single data point in our experiments requires nearly a full hour of simulation time. We have simulated topologies up to approximately double these sizes, without any noticeable impact on the results.

[3]The gini inequality index is used to generate random memory distributions.

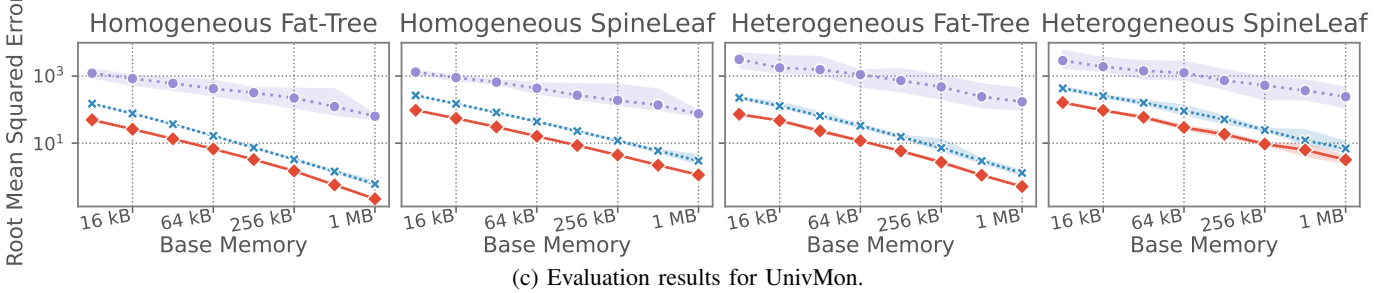
[4]We deploy UnivMon with 16 levels, which is approximately the second logarithm of the expected number of flows.



(a) Evaluation results for a Count Sketch.



(b) Evaluation results for a Count-Min Sketch.



(c) Evaluation results for UnivMon.

Fig. 13: Frequency estimation errors of sketches for 5-hop flows at various memory sizes. Scenarios are combinations of the Fat-Tree and SpineLeaf topologies, either with homogeneous or heterogeneous switch memory sizes.

suboptimal initial epochs in the query window.

A. *Frequency Estimation Errors*

In Figure 13, we present the accuracy of flow frequency estimation, a common sketch query [17], [23], [53]. Here, we query all flows that were tracked during the experiment in terms of their number of transmitted packets and define the query error as the absolute difference from the ground truth number of packets. We use the aforementioned experimental setup and replay traffic across the entire network topology. In this figure, only flows with full-length network paths are queried, to ensure that all evaluated flows would have traversed an aggregated sketch residing on core switches. We evaluate per-path-length performance later in Section VI-D.

This evaluation demonstrates a clear pattern, where our solution consistently outperforms both aggregated deployments and DISCO-disaggregated deployments of all three example sketches. This pattern holds for homogeneous and heterogeneous deployments in both of the evaluated network topologies. As an example, in a heterogeneous Fat-Tree deployment, DiSketch-CS delivers $RMSE < 1$ using on average 128KB of per-switch memory, which is approximately 25% of the 512KB that DISCO requires. All aggregated sketches failed to deliver this accuracy under the evaluated memory constraints. Alternatively, using the same 512KB of average per-switch memory, the aggregated CS, DISCO-CS, and DiSketch-CS deliver $RMSE$ of 460, 0.8, and 0.09 respectively. These patterns hold when the sketching epoch, or the underlying traffic volumes, increase. We see similar patterns for all evaluated sketches and network topologies. This does, however, only investigate flows with full-length network paths, at fixed heterogeneity levels. We will demonstrate the effect of these parameters in sections VI-C and VI-D.

Takeaway: DiSketch consistently and significantly enhances the memory-vs-accuracy tradeoff for frequency estimation, either increasing the accuracy by nearly an order of magnitude or reducing the required memory by approximately 75% depending on the sketching environment.

B. *Entropy Estimation Errors*

In Figure 14, we present the experimental results of estimating the network-wide entropy of IP addresses. This is done through UnivMon, according to the algorithm outlined in their paper [53]. Given that disaggregated sketches far outperform aggregated sketches, we here compare DISCO only with DiSketch. We therefore also remove the full-path-length limitation, and base these evaluations on all network traffic. We present only the heterogeneous Fat-Tree scenario here, but the results are similar in all evaluated scenarios.

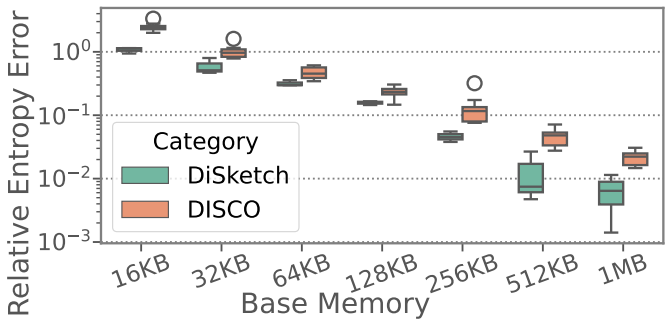


Fig. 14: The error in entropy estimation of UnivMon in a Fat-Tree network with a random memory distribution.

Similar to the previous section, we note a consistent and significant error reduction through our solution, and DiSketch-UM delivers a similar entropy estimation accuracy as DISCO-UM using half as much memory. Alternatively, DiSketch-UM lowers the errors by approximately 50% across tests.

Takeaway: Our solution halves the amount of memory required for entropy estimation, or halves the errors while keeping the memory unchanged.

C. Heterogeneity Effects

Here, we evaluate DiSketch in a wide range of heterogeneous environments (i.e., where the traffic load and memory sizes vary between switches). For simplicity, we again focus on the frequency estimation accuracy of DISCO-CS and DiSketch-CS.

Experimental Setup. As opposed to the prior experiment, this experiment simulates a single 5-hop network path as shown in Figure 16. The network load heterogeneity depends on various factors, including the network topology, load balancing, node positions, and the traffic patterns of connected hosts. By simulating a single path, we gain precise control of the per-switch traffic volumes and can set the heterogeneity levels freely. Traffic on other semi-overlapping paths is emulated as per-switch background traffic.

We vary the memory and traffic load heterogeneity of the hops, according to the coefficient of variation (CoV, i.e., the relative standard deviation). Heterogeneity levels range from perfectly homogeneous (CoV $= 0$) to significantly heterogeneous (CoV $= 1.8$). The total amount of background traffic ($259K$ packets) and the number of on-path counters (5120) is fixed across tests[5]. Two pseudo-random lists of 5 integers are generated for each test, one with the per-hop load, and the other with the per-fragment memory size, so that both lists adhere to the experimental parameters. For instance, an example background traffic distribution with CoV ≈ 1.5 is [$204189(78.7\%)$, $18364(7.1\%)$, $29(0.01\%)$, $2265(0.9\%)$, $34675(13.4\%)$]. The generated loads and widths are independent and are randomly distributed to the nodes.

There are six different packet streams: five background traffic streams, each passing through one switch, and one evaluation stream that traverses the entire path. The aforementioned CAIDA packet trace is used as the basis of all network traffic, and we map IP addresses at random into the traffic streams. Packets in each stream are replayed chronologically in the order they appear in the packet trace. Approximately 300 flows comprising approximately 1% of the total traffic are replayed across the full path, from sender to receiver, and are used for evaluation. The remaining 99% of traffic only *crosses* the individual nodes (see Load in Figure 16). The experimental results are only based on flows traversing all five of these fragments.

The experimental results are provided in terms of the Normalized Root Mean Squared Error (NRMSE) [6] between the estimated frequencies and the known ground truth, and is normalized by the total number of packets to provide a dimensionless measure of error. Smaller values indicate better performance.

Results. The average results following multiple simulations are presented in Figure 15 for all heterogeneity combinations. To further help illustrate the effect of DiSketch, we present the difference in NRMSE between DiSketch and DISCO in the rightmost heatmap.

There are several patterns in this data, including an apparent direct link between the width/load heterogeneity and the sketching accuracy. The width heterogeneity pattern appears valid, in that we expect increased heterogeneity in per-switch memory to reduce the overall sketching accuracy in the network. This effect is intuitively explained by the theory behind sketches, in that a single row on its own might be inaccurate, but combining several rows boosts the accuracy super-linearly. A high width heterogeneity means that a few switches have most of the memory, leading to fewer suitably accurate rows, degrading performance when the total memory remains fixed. However, the apparent beneficial effect of load heterogeneity on the sketching accuracy is a likely experimental artifact. Recall that we are only evaluating the accuracy of flows traversing the full path, and the total amount of background traffic remains constant. Therefore, as we increase the load heterogeneity of the background traffic, we condense it into fewer switches, leading to an accuracy increase for most on-path fragments. This effect might not hold if those background flows were accounted for in the evaluation.

We highlight the right-most heatmap, which shows the accuracy improvement as you replace DISCO with DiSketch. There is a consistent accuracy enhancement in every heterogeneity combination, with greater gains as the heterogeneity levels increase. This is expected, as DISCO is heterogeneity unaware, while DiSketch fragments mitigate the heterogeneity penalties by autonomously reconfiguring themselves to equalize the network-wide PEB. Note that the improvement is presented in terms of the absolute change in $\log_{10}$ NRMSE, so a value of 1 would signal an order of magnitude reduction in NRMSE.

Takeaway: DiSketch outperforms DISCO in all heterogeneity combinations tested. This is especially evident as the heterogeneity levels increase, demonstrating the effectiveness of network-wide PEB equalization.

[5]The small scale in these simulations allowed us to evaluate numerous heterogeneity settings within a reasonable time, and yields the same general pattern as full-scale heterogeneity simulations.

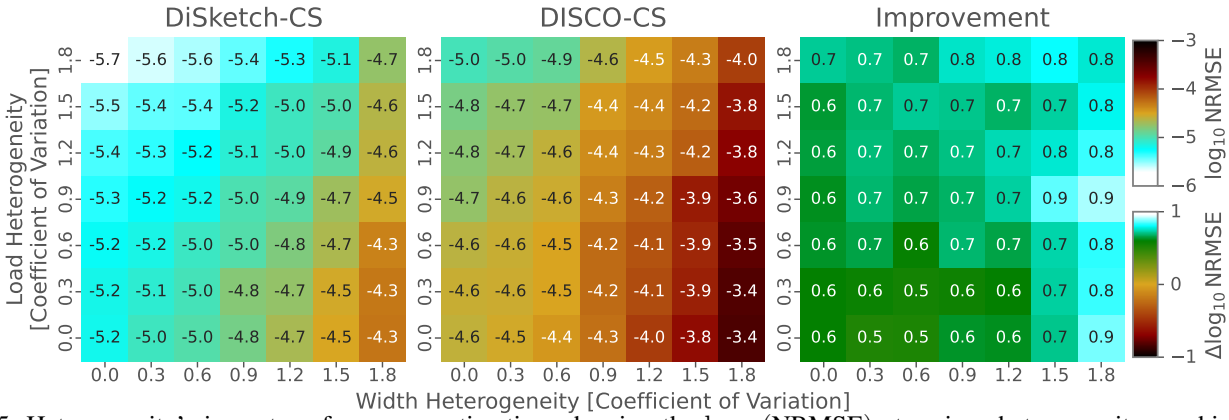


Fig. 15: Heterogeneity's impact on frequency estimation, showing the $\log_{10}$(NRMSE) at various heterogeneity combinations. The right-most heatmap shows the $\log_{10}$(NRMSE) improvement over DISCO gained through spatiotemporal disaggregation.

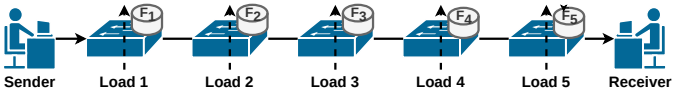


Fig. 16: Experimental Setup in Heterogeneity Tests.

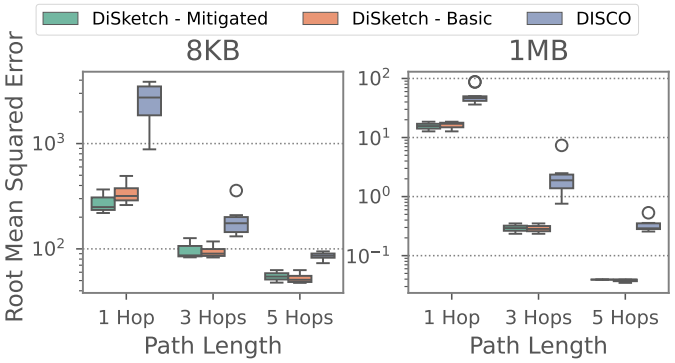


Fig. 17: The path lengths' impact on frequency estimation accuracy for Count Sketch. DiSketch is evaluated both with and without the single-hop mitigation strategy.

D. Path Length Effects

Here, we evaluate how the path length impacts the accuracy of DiSketch-CS and DISCO-CS. The heterogeneous Fat-Tree scenario from previous sections is reused, and the query results are grouped according to the path lengths of the queried flows. For ease of presentation, we only present the experimental results with the smallest base memory (8KB) and the largest base memory (1MB) in Figure 17.

As expected, increasing the number of traversed sketching fragments (i.e., the path length) improves query accuracy across all disaggregated sketches. This effect is most significant when more memory is allocated to sketching, with single-hop queries experiencing approximately a 50x increase in errors over 3-hop flows in the 1MB experiments, compared to a more reasonable 2.8x increase in the 8KB experiments. When applying the mitigation technique from Section IV-D, errors for single-hop flows decreases by approximately 24% and 13% in the 8KB and 1MB tests, respectively. Further, DiSketch demonstrates a consistent accuracy improvement over DISCO for all evaluated path lengths. The mitigation results in a slight error increase for queries of multi-hop flows, due to the increase in increments from single-hop flows, although the effect is within one standard deviation in these experiments.

Takeaway: The query accuracy is greatly impacted by the path length of the underlying flow, and the single-hop mitigation results in a slight accuracy enhancement for single-hop flows and a slight accuracy decrease for multi-hop flows. Accordingly, whether mitigation is worthwhile may depend on the setting.

VII. Discussion

This section provides a brief discussion of spatiotemporal disaggregation and proposes future research into the techniques presented in this paper.

Other data structures. This paper presented spatiotemporal disaggregation when coupled with Count Sketch, Count-Min Sketch, and UnivMon. However, we expect these ideas to be useful beyond that and envision that they can be applied to other sketches (e.g., Bloom filters, and HyperLogLog), as well as non-sketch data structures. The exact technical requirements for spatiotemporal disaggregatability, as well as structure modifications required in each case, are left as future work. However, non-cumulative estimations likely have to modify the temporal merging at query time, while the spatial merging likely follows the row-merging logic of an aggregated version of the sketch, similar to how we spatially merge Count Sketch fragments through the median.

Path stability. Some load balancing schemes, such as flowlet switching [66], frequently alter the path of flows at incredibly short timescales. These techniques result in irregular and unstable flow paths, impacting the practicality of sketch disaggregation. For instance, if flow paths change during a sketching epoch, then portions of the flow increments within that epoch could end up in different sets of fragments. Without awareness of these path changes, the estimation accuracies of the analysis engine could suffer. If the analysis engine is unaware of such paths, then the estimation accuracies would suffer. However, it is possible to design disaggregation techniques that accommodate path changes. Assuming fine-grained path tracing is already implemented, the analysis

engine could incorporate all fragments traversed during the epoch into the composite sketch output. Weighting could be employed based on the duration during which a flow has traversed each fragment. Developing precise techniques for sketch-based estimations under changing path conditions is left as future work.

Alternatively, one could configure the load balancing techniques to only perform re-routing at epoch transitions. This adjustment would ensure that each sketching epoch contains the same set of fragments, except in cases of re-routing triggered by failures. This method could stabilize the path data within each epoch, improving the consistency and accuracy of sketch-based monitoring.

Finding an Optimal ρ_{target}. Spatiotemporal disaggregation is built around subepoching, where the monitoring epoch is dynamically divided into briefer subepochs based on fragments' PEBs (ρ). However, one critical aspect has not been investigated: what is the optimal ρ_{target}? This choice is strongly influenced by the network characteristics where the sketch is deployed, including the expected loads, fragment sizes, and burstiness of flow traffic patterns. Determining a theoretically optimal ρ_{target} is beyond the scope of this paper and is left for future work. However, in our experience, the selection is relatively forgiving, with any value within a factor of two of the optimal ρ_{target} yielding similar performance. Regardless, this warrants an in-depth theoretical analysis, and experimental validation to quantify the impact of this choice across diverse network conditions.

VIII. Related Work

Traditionally, sketches have been deployed aggregated [17], [23], [53]. Recent interest has shifted towards network-wide deployments to enhance measurement flexibility [15], [25], [34], [48], [71], [74].

For example, Zhao et. al., 2021, [74] designed a sketch-based network-wide telemetry system named LightGuardian. In their approach, each switch hosts two *SuMax* sketches: one actively populated and one being collected. This supports new sketch measurements, including latency jitter and packet loss detection, using a probabilistic in-band collection method to reduce centralized collection costs. Nonetheless, this system does not address heterogeneous environments, incurs substantial resource costs, and employs aggregated (i.e., non-disaggregated) sketches on each switch.

To our knowledge, the first paper describing the potential of disaggregated sketches was DISCO [15], published in 2020. They argued for sketch disaggregation to simplify sketch deployments in resource-scarce environments. They presented a basic technique for per-row disaggregation of sketches for estimating flow size, showing an increased accuracy in heavy hitter detection. Although promising, they did not investigate heterogeneous environments, hardware feasibility, or more complex sketches for tasks unrelated to flow size estimation.

Cornacchia et al., 2021, [25] highlighted the detrimental effects of traffic patterns on per-row disaggregated sketches, notably increased hash collisions and accuracy degradation due to load imbalances. They proposed that sketch fragments sample a subset of traffic to process, but their algorithm assumes full in-band knowledge of per-flow paths and fragment dimensions, thus introducing considerable overheads and limiting deployment flexibility. We believe that their fundamental idea is valid, that sampling techniques would lead to efficient distributed sketching in heterogeneous environments, but that their imposed assumptions are unreasonable in many deployments. Hence, we choose not to include this solution in the evaluation. DiSketch fragments operate autonomously, without any per-flow knowledge assumptions. We discussed the issues with these assumptions in Section III.

Gu et al., 2023, [34] proposed per-column disaggregation as an alternative to per-row disaggregation and discussed how to handle load imbalances. As outlined in Section III, this approach faces the challenge of evenly distributing counters across diverse paths to ensure balanced counter incrementation relative to fragment sizes. To address this, the authors propose the use of lookup tables for counter allocation, containing one entry for every active traffic flow. However, since the number of sketch cells typically grows sub-linearly to the number of keys, the inclusion of such a lookup table contradicts this fundamental design goal. Further, per-column disaggregation is inefficient in high-performance switching architectures such as PISA, since per-row computational logic remains dedicated even for packets where it is not utilized, leading to a high resource footprint.

Li et al., 2024, [48] addresses the traffic imbalance issue by proposing a deployment and increment strategy that ensures load balancing across sketch rows. Their method selectively deploys rows across the network, supporting a variable number of rows per fragment. The ingress switch determines the number of rows each hop should process per packet, inserting this information as a new header for ingressing packets and allocating traffic based on hop capacities. However, this places high burdens on ingress switches, requiring extensive knowledge of network paths and fragment dimensions, and reduces the MTU of the network through extended packet headers, risking frame fragmentation and reduced goodput. DiSketches has none of these issues.

IX. Conclusion

We introduced spatiotemporal sketch disaggregation, a novel sketching scheme for disaggregated streaming analysis. This approach combines spatial and temporal indexing through subepoching, enabling flexible and heterogeneous deployments of multiple sketches, as demonstrated with Count Sketch, Count-Min Sketch, and UnivMon.

Our findings show that spatiotemporal disaggregation significantly enhances sketch accuracy, reducing estimation errors by nearly an order of magnitude compared to DISCO, and by several orders of magnitude over conventional aggregated sketching methods. These accuracy improvements are especially pronounced in heterogeneous environments, where network-wide equalization of probabilistic error bounds allows fragments of varying loads and sizes to be efficiently queried together.

REFERENCES

[1] Marcos K. Aguilera, Wojciech Golab, and Mehul A. Shah. A practical scalable distributed b-tree. *Proc. VLDB Endow.*, 1(1), 2008.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[3] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proc. of ACM SIGCOMM*, 2014.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996.

[5] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Volumetric hierarchical heavy hitters. In *Proc. of IEEE MASCOTS*, 2018.

[6] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. Salsa: self-adjusting lean streaming analytics. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 864–875. IEEE, 2021.

[7] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minlan Yu, and Michael Mitzenmacher. PINT: probabilistic in-band network telemetry. In *SIGCOMM '20*, pages 662–680. ACM, 2020.

[8] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proc. of SIGCOMM IMC*, 2010.

[9] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[10] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR*, 2014.

[11] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[12] Broadcom. Broadcom NetOps. https://academy.broadcom.com/blog/network-operations/dx-netops/5-steps-to-get-telemetry-data-in-dx-netops.

[13] Broadcom. Broadcom Tomahawk 5. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78900-series.

[14] Broadcom. Broadcom Trident 5. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78800.

[15] Valerio Bruschi, Ran Ben Basat, Zaoxing Liu, Gianni Antichi, Giuseppe Bianchi, and Michael Mitzenmacher. Discovering the heavy hitters with disaggregated sketches. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 536–537, 2020.

[16] CAIDA. Passive monitor: equinix-nyc, 2019.

[17] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proc. of ICALP*, 2002.

[18] Peiqing Chen, Dong Chen, Lingxiao Zheng, Jizhou Li, and Tong Yang. Out of many we are one: Measuring item batch with clock-sketch. In *Proceedings of the 2021 International Conference on Management of Data*, pages 261–273, 2021.

[19] Peiqing Chen, Yuhan Wu, Tong Yang, Junchen Jiang, and Zaoxing Liu. Precise error estimation for sketch-based flow measurement. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 113–121, 2021.

[20] Peter Clifford and Ioana Cosma. A simple sketching algorithm for entropy estimation over streaming data. In *Artificial Intelligence and Statistics*, pages 196–206. PMLR, 2013.

[21] Edith Cohen, Rasmus Pagh, and David Woodruff. Wor and p's: Sketches for l_p-sampling without replacement. *Advances in Neural Information Processing Systems*, 33:21092–21104, 2020.

[22] Michael B Cohen and Richard Peng. Lp row sampling by lewis weights. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 183–192, 2015.

[23] Graham Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. *J. Algorithms*, 2005.

[24] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[25] Alessandro Cornacchia, German Sviridov, Paolo Giaccone, and Andrea Bianco. A traffic-aware perspective on network disaggregated sketches. In *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, pages 1–4. IEEE, 2021.

[26] Christina Delimitrou, Sriram Sankar, Aman Kansal, and Christos Kozyrakis. Echo: Recreating network traffic maps for datacenters with tens of thousands of servers. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*, pages 14–24. IEEE, 2012.

[27] Fenghao Dong, Yang He, Yutong Liang, Zirui Liu, Yuhan Wu, Peiqing Chen, and Tong Yang. Simisketch: Efficiently estimating similarity of streaming multisets. *arXiv preprint arXiv:2405.19711*, 2024.

[28] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 323–336, 2002.

[29] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88, 2014.

[30] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 57–70, 2024.

[31] Sumit Ganguly. Counting distinct items over update streams. *Theoretical Computer Science*, 378(3):211–222, 2007.

[32] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proc. of ACM SOSP*, 2003.

[33] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 225–238, 2017.

[34] Liyuan Gu, Ye Tian, Wei Chen, Zhongxiang Wei, Cenman Wang, and Xinming Zhang. Per-flow network measurement with distributed sketch. *IEEE/ACM Transactions on Networking*, 2023.

[35] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proc. of ACM SIGCOMM*, 2018.

[36] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research*, pages 1–7, 2018.

[37] Nicholas JA Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 489–498. IEEE, 2008.

[38] Yongchao He, Wenfei Wu, Xuemin Wen, Haifeng Li, and Yongqiang Yang. Scalable on-switch rate limiters for the cloud. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[39] Qun Huang, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. Sketchvisor: Robust network measurement for software packet processing. In *Proc. of ACM SIGCOMM*, 2017.

[40] Qun Huang, Patrick PC Lee, and Yungang Bao. Sketchlearn: Relieving user burdens in approximatemeasurement with automated statistical inference. In *Proc. of ACM SIGCOMM*, 2018.

[41] Qun Huang, Siyuan Sheng, Xiang Chen, Yungang Bao, Rui Zhang, Yanwei Xu, and Gong Zhang. Toward nearly-zero-error sketching via compressive sensing. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 1027–1044, 2021.

[42] Piotr Indyk and Milan Ruzic. Near-optimal sparse recovery in the l1 norm. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 199–207. IEEE, 2008.

[43] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. Clove: Congestion-aware load balancing at the virtual edge. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 323–335, 2017.

[44] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[45] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael Swift. ATP: In-network aggregation for multi-tenant learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 741–761. USENIX Association, April 2021.

[46] Kasper Green Larsen, Rasmus Pagh, and Jakub Tětek. Countsketches, feature hashing and the median of three. In *International Conference on Machine Learning*, pages 6011–6020. PMLR, 2021.

[47] Alberto Lerner, Rana Hussein, Philippe Cudré-Mauroux, and U eXascale Infolab. The case for network accelerated query processing. In *CIDR*, 2019.

[48] Fuliang Li, Kejun Guo, Jiaxing Shen, and Xingwei Wang. Effective network-wide traffic measurement: A lightweight distributed sketch deployment. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, 2024.

[49] Minghao Li, Ran Ben Basat, Shay Vargaftik, ChonLam Lao, Kevin Xu, Xinran Tang, Michael Mitzenmacher, and Minlan Yu. THC: Accelerating Distributed Deep Learning Using Tensor Homomorphic Compression. In *USENIX Symposium on Networked Systems Design and Implementation*, 2024.

[50] Yuliang Li, Rui Miao, Mohammad Alizadeh, and Minlan Yu. {DETER}: Deterministic {TCP} replay for performance diagnosis. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 437–452, 2019.

[51] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58, 2019.

[52] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. Distcache: Provable load balancing for large-scale storage systems with distributed caching. In *Proc. of USENIX FAST*, 2019.

[53] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proc. of ACM SIGCOMM*, 2016.

[54] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3829–3846, 2021.

[55] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[56] Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. Fast in-network gray failure detection for isps. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 677–692, 2022.

[57] Andrea Monterubbiano, Jonatan Langlet, Stefan Walzer, Gianni Antichi, Pedro Reviriego, and Salvatore Pontarelli. Lightweight acquisition and ranging of flows in the data plane. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 7(3):1–24, 2023.

[58] Barefoot Networks. Barefoot Tofino. https://barefootnetworks.com/products/brief-tofino/.

[59] NVIDIA. NVIDIA DeepStream. https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_Overview.html.

[60] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, et al. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 194–206, 2021.

[61] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015.

[62] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701*, 2019.

[63] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. Internet performance from facebook's edge. In *Proceedings of the Internet Measurement Conference*, pages 179–194, 2019.

[64] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proc. of ACM SOSR*, 2017.

[65] Muhammad Tirmazi, Ran Ben Basat, Jiaqi Gao, and Minlan Yu. Cheetah: Accelerating database queries with switch pruning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2407–2422, 2020.

[66] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 407–420, 2017.

[67] Qing Wang, Youyou Lu, and Jiwu Shu. Sherman: A write-optimized distributed b+tree index on disaggregated memory. In *Proceedings of the 2022 International Conference on Management of Data*, New York, NY, USA, 2022. Association for Computing Machinery.

[68] Theophilus Wellem, Yu-Kuen Lai, Chao-Yuan Huang, and Wen-Yaw Chung. A flexible sketch-based network traffic monitoring infrastructure. *IEEE Access*, 7:92476–92498, 2019.

[69] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, and Ang Chen. Bedrock: Programmable network support for secure rdma systems. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2585–2600, 2022.

[70] Jiarong Xing, Wenqing Wu, and Ang Chen. Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3865–3881, 2021.

[71] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proc. of ACM SIGCOMM*, 2018.

[72] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Proc. of USENIX NSDI*, 2013.

[73] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *USENIX Workshop on Hot Cloud (HotCloud)*, 2010.

[74] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 991–1010, 2021.