Sketchy With a Chance of Adoption: Can Sketch-Based Telemetry Be Ready for Prime Time?

Zaoxing Liu*, Hun Namkung, Anup Agarwal, Antonis Manousis,
Peter Steenkiste, Srinivasan Seshan, Vyas Sekar
**Boston University, Carnegie Mellon University*

2021 IEEE 7th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0522-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/NetSoft51509.2021.9492582

***Abstract*—Sketching algorithms or sketches have emerged as a promising alternative to the traditional packet sampling-based network telemetry solutions. At a high level, they are attractive because of their high resource efficiency and provable accuracy guarantees. While there have been significant recent advances in various aspects of sketching for networking tasks, many fundamental challenges remain unsolved that are likely stumbling blocks for adoption. Our contribution in this paper is in identifying and formulating these research challenges across the ecosystem encompassing network operators, platform vendors/developers, and algorithm designers. We hope that these serve as a necessary fillip for the community to enable the broader adoption of sketch-based telemetry.**

I. Introduction

At the core of managing networks, network telemetry plays a crucial role in understanding what is happening in the network and informing management decisions. For example, to improve cloud security, telemetry enables operators to detect network anomalies and attacks in a timely fashion. Similarly, in order to optimize traffic engineering and ensure that service-level agreements (SLAs) for applications are met, operators commonly rely on telemetry to monitor network flow distributions. Traditionally, flow-based telemetry is done via offline analysis or some form of packet or flow sampling (e.g., NetFlow [1] and sFlow [2]). However, given the need for timely results using constrained compute/memory resources, offline analysis is not a practical option. Moreover, sampling only provides coarse-grained flow size distributions, and cannot provide accurate results for more fine-grained key telemetry tasks such as entropy estimation, distinct count, and change detection [3]–[5].

To address the drawbacks of packet sampling approaches, sketching algorithms (or sketches for short) have been extensively studied in recent years (e.g., [5]–[21]). In light of increasing network traffic and ever-evolving application dynamics, sketches have emerged as a promising solution for real-time network telemetry.

This paper is a reflection on the current state of sketch-based telemetry to examine not just what sketch-based systems *can* do but what *should* be done to enable broader adoption. To this end, we look at the state of the sketch-based telemetry ecosystem from the perspective of three key stakeholders in Figure 1: (1) *Network Operators (NO)* who are the users/consumers of telemetry capabilities; (2) *Algorithm Designers (AD)* who design and analyze sketching algorithms;

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

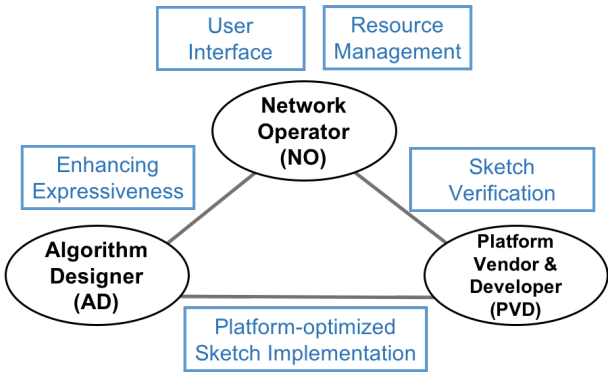


Fig. 1: Overview of the problems from the stakeholders in sketch-based telemetry.

and (3) *Platform Vendors and Developers (PVD)* who provide hardware/software primitives and APIs in various platforms (e.g., Intel DPDK [22], Barefoot Tofino [23], Broadcom Trident [24], Mellanox [25], among others) and use these APIs to develop and implement sketch-based functions. Developers are supposed to be familiar with the software and hardware primitives from vendors.

By taking this ecosystem-level view, we identify four areas of gaps between stakeholder and interaction requirements and existing research (blue boxes in Figure 1):

- **NO-Centric:** While NOs are the users of the telemetry systems, most existing efforts make unrealistic assumptions that they have extensive knowledge about the algorithms and the underlying data structures. There are few, if any, efforts to help operators translate high-level intents into sketches. This requires both high-level interfaces as well as precise resource management. While NO's intents may involve different sketches and devices, current solutions (e.g., [11], [14], [15], [26]) do not consider the composition of multiple types of sketches and the heterogeneity of network devices.
- **Between NO/AD:** Prior theoretical work in sketching algorithms covers many common telemetry tasks, and more recent work on general sketches can cover a broad portfolio of tasks [14]. Despite these advances, many common NO intents fall outside the scope of the literature. For instance, for attack detection, operators are interested in obtaining statistics from not only one dimension of data (e.g., SrcIP) but multiple dimensions (e.g., any subset of the combinations in 5-tuple). Conversely, we find that the

theory community has many rich capabilities and streaming models (e.g., turnstile [27], [28], sliding-window [29]–[31], and distributed functional monitoring [32], [33]) that are yet to find practical adoption in networking.

- **Between AD/PVD:** While sketching algorithms are theoretically resource-efficient, existing algorithms may not be efficiently realizable across diverse platforms as highlighted by recent efforts [10], [15]–[17], [34]. Similarly, while existing languages and APIs [35]–[37] are sufficiently expressive to specify different sketch algorithms, naïve implementations are often resource intensive, thus nullifying any potential benefits [10], [17]. This suggests the need for new sketch-centric APIs, language support, and best practices.
- **Between NO/PVD:** Given that the success of the operator's policies depends crucially on how accurately telemetry reflects current network conditions, verifying the practical accuracy and correctness of sketches at post-deployment is a major priority for the NO. In addition, while platform vendors have designed and delivered trusted hardware capabilities (e.g., Intel SGX [38], AMD SEV [39], and ARM TrustZone [40]) to ensure the integrity of the program running on the device, the integrity of sketch-based telemetry logic has yet to be protected.

Our contribution in this paper is to identify and formulate challenges that need to be addressed to enable sketch-based telemetry to be more widely adopted. While this list of challenges is by no means exhaustive, our goal is to start the conversation regarding the ecosystem's missing pieces. We hope that our work will inspire the community to tackle these as-yet-unsolved issues, eventually enabling the practical adoption of sketch-based telemetry.

II. Background

In this section, we first provide some background on sketches and their use in network telemetry. We then introduce the key stakeholders in sketch-based telemetry to set the context for the research challenges.

A. Sketching Algorithms

Sketching algorithms (sketches) can process data streams accurately and efficiently in an online fashion. Sketches are attractive for network monitoring precisely because they typically require small memory footprints to estimate traffic statistics with provable accuracy guarantees. In addition to network telemetry [7]–[11], [14]–[19], [41], sketch-based approaches have also been applied in databases [42], [43], streaming analytics [44], security [45], and machine learning [46]–[48].

Sketches draw on rich theoretical foundations starting from the foundational "AMS" paper [49]. At a high level, the problem they address is as follows: Given an input stream of <*key, value*> pairs (e.g., <5-tuple, packet size> pairs in network traffic), a sketching algorithm is allowed to make a single pass over the data stream to compute statistics while using sub-linear (usually poly-logarithmic) memory space compared to the total size of the dataset and the number of distinct keys. When processing each item in the stream, a sketch typically maintains a table of counters in the memory and computes multiple independent hashes to update a small random set of counters in the table. These algorithms are backed by rigorous theoretical analysis on bounded accuracy-memory tradeoffs for arbitrary workload patterns.

Sketch-based network telemetry. Sketches are useful approaches for key network telemetry tasks, such as (1) Heavy-Hitter detection to discover large flows [8], [9], [14]–[16], [41]; (2) Entropy Estimation to analyze traffic distributions for anomaly detection [14], [19], [50]; (3) Change Detection to identify significant traffic shifts over time [5], [14], [18]; (4) Cardinality Estimates to detect the number of distinct items/flows in the network traffic [6], [10], [14], [51]; (5) Performance Monitoring to identify flows with high packet loss, large latency, and high out-of-order or retransmitted packets [20]; (6) Superspreader Detection to identify sources that contact many different destinations [18], among others.

Sketches are promising in a network-wide measurement setting due to their mergeability. Independent sketch instances deployed at different parts of the network can be merged to obtain network-wide or partial network aggregated results with the same bounded errors.

B. Stakeholders for Telemetry Deployment

We identify three key players in the ecosystem that drive and influence the adoption of the above sketch-based telemetry.

Network operator: Network operators rely on real time telemetry to make timely management decisions that ensure network reliability, performance, and security. To this end, they may want network-wide information such as global heavy hitter flows, distinct flows, and entropy changes on various traffic distributions. Ideally, network operators want to express high-level telemetry objectives without having to worry about low-level algorithmic and implementation details about sketches.

Q1: Return 5-tuple 0.005-heavy hitters from an OD path

```
FlowKey = (SrcIP,SrcPort,DstIP,DstPort,Proto)
Windows = 10
C= "Select HeavyHitter(p.FlowKey,0.05) From OD-1
Where Not p.DstIP = 1.2.3.4"
return C
```

Q2: Return distinct DstIP count that a host connects to

```
C="Select Distinct(p.DstIP) From *
Where p.SrcIP=1.2.3.4"
return C
```

Fig. 2: Examples of "envisioned" telemetry queries.

For example, operators may specify queries like Q1 and Q2 depicted in Figure 2. A telemetry system should provide an interface to write queries, identify if the queries can be supported by existing primitives, and distribute the monitoring responsibilities efficiently across a network. If better or new sketches are needed in the telemetry system, operators should pass these information to algorithm designers described below.

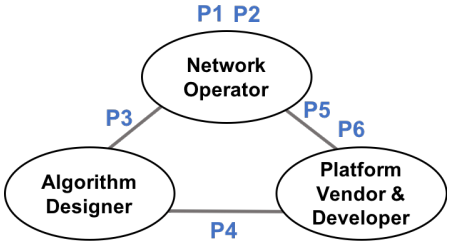


Fig. 3: Open problems between stakeholders.

Algorithm designers: We envision an active community of algorithm designers developing new sketching algorithms to estimate different telemetry metrics. They would like to understand the requirements of network operators to design improved or new sketching algorithms for needed metrics. In practice, however, it requires significant systems efforts to translate theoretical algorithms into optimized implementations on diverse platforms. As richer primitives need to be designed and new platforms emerge (e.g., Barefoot Tofino [23] and Micro-engine SmartNIC [37]), algorithm designers increasingly find themselves in need of a mature sketch-based framework allowing them to develop and evaluate algorithmic tools along with platform vendors/developers described below.

Platform vendors and developers: Platform vendors offer specialized capabilities that implement and optimize sketches on various hardware and software platforms. For instance, we have already seen programmable switches, SmartNICs, FPGAs, DPUs, and software switches established in today's networks, and we envision future deployments with richer and more diverse platform capabilities. Ideally, platform vendors should provide primitives for these developers to optimally support sketch-based telemetry. However, recent efforts suggest it is non-trivial to efficiently implement sketches [16], [17]. In this respect, we envision the need for these two stakeholders to jointly contribute their domain expertise to achieve optimized sketch implementations.

III. Research Challenges

Next, we formulate a broad (but non-exhaustive) list of open research problems P1 to P6 and some of their extensions for a sketch-based telemetry ecosystem. As depicted in Figure 3, we conceptually cluster these challenges according to each stakeholder's needs and considerations.

Preliminaries: We introduce some terms and notations to formulate the problems (summarized in Table I).

- The constants represent the inputs to the telemetry system. Specifically, network operators can define their telemetry needs by a list of input constants: (1) Queries $\mathcal{Q}$ consisting of a set of k (potentially infinite) query definitions $\{q_1, \ldots, q_k\}$; (2) Requirements $\mathcal{R}_\mathcal{A} = \{{r_a}^1, \ldots, {r_a}^k\}$ defining a set of accuracy requirements (e.g., accuracy target 95% with 0.99 confidence) for queries $\{q_1, \ldots, q_k\}$ and similarly $\mathcal{R}_\mathcal{P} = \{{r_p}^1, \ldots, {r_p}^k\}$ as the packet rate requirements; (3) Network characteristics including topology information $\mathcal{T}$, device information with resource capabilities $\mathcal{D}$, and traffic workload characteristics $\mathcal{W}_\nabla$.

Constants	Definition
$\mathcal{Q}$	Set of telemetry queries
$\mathcal{R}_\mathcal{A}$	Set of accuracy requirements, e.g., accuracy target and confidence level
$\mathcal{R}_\mathcal{P}$	Set of performance requirements, e.g., packet rate
$\mathcal{T}$	Topology information, e.g., links and devices
$\mathcal{D}$	Set of device instances with resource constraints, e.g., SmartNIC w/ 4 engines and 10MB SRAM
$\mathcal{W}_\nabla$	Traffic workload characteristics, e.g., distribution
Variables	Definition
S	Set of sketch definitions with configurations
$r_{s,d}$	Resource config. for sketch s on device d
$l_{s,d}$	Processing latency for sketch s on device d
$c_{s,d}$	Implementation of sketch s on device d
c_d	Implementation of all sketches on device d
r_d	Actual resource usage of device d from c_d
l_d	Actual processing latency of device d from c_d

TABLE I: Summary of notations in problem definitions.

- The variables are the notations for the intermediate or final outputs of the telemetry system: (1) S is a set of sketch definitions with appropriate memory and flow-key/OD-pair configurations (e.g., a Count-Min sketch tracking 5-tuple flows with 5×2048 32-bit counters); (2) $r_{s,d}$ is the resource configuration of sketch instance s on device d (e.g., assigning 200KB and 2 cores for s on CPU) and $l_{s,d}$ is the processing latency of s on d (e.g., $1\mu s$ on CPU); (3) $c_{s,d}$ is the implementation (binary code) of sketch instance s on device d. When there are multiple sketch instances in d, c_d represents the implementation of all instances combined; (4) r_d is the actual resource usage of c_d and l_d is the actual processing latency of c_d.

A. Network Operator-Centric

***Problem 1**: [Query Language] Is there a high-level declarative language that can precisely define sketch-based telemetry queries $\mathcal{Q}$?*

Sketch-based telemetry is traditionally designed under a narrow scope in the queries it supports. Specifically, existing frameworks are either designed to support one type of queries [43] or assume that the operators determine at query time the appropriate (available) sketch for each query. For example, to detect Superspreaders (i.e., SrcIPs that connect to many distinct DstIPs), the operators need to make a choice between Count-Min + HLL and CountSketch + UnivMon whereas to conduct change detection they need to choose between K-ary and Count-Min. As a result, developing a unified front-end for such sketch-based telemetry systems was, to the best of our knowledge, never seen as a key design priority. Specifically, the operators should be able to conceptually describe the characteristics of a query to execute (e.g., type of metrics, appropriate aggregation of data, accuracy constraints) without explicitly specifying the execution mechanism.

Existing efforts have proposed several query languages for network telemetry [52]–[54], streaming database [55], [56], and traffic analysis [57]. These efforts are self-contained for their systems but may not be an ideal fit for sketch-based

telemetry. Specifically, they did not consider sketches as their primitives and may overly complicate the query definitions for sketches. For instance, Sonata [52] provides a target-agnostic query interface for network operators by specifying the detailed packet-level queries with dataflow operators (e.g., map, filter, reduce) but it is unclear how to describe sketches. For example, NetQRE [53] extends from quantitative regular expressions [58] to define flow-level and application-level statistics and polices. In addition, the telemetry tool Marple [54] is designed to support a particular set of performance metrics only. Similarly, streaming databases such as Gigascope [56] support continuous queries over packet headers or counts via a SQL-alike language but do not support other metrics such as network performance and traffic patterns.

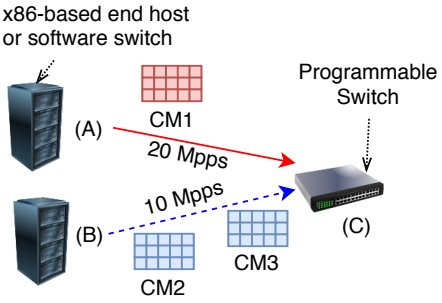
***Problem 2**: [Network-wide Optimization] Given a set of queries $\mathcal{Q}$ with accuracy requirements $\mathcal{R}_\mathcal{A}$ and performance requirements $\mathcal{R}_\mathcal{P}$, traffic workload characteristics $\mathcal{W}_\nabla$, topology $\mathcal{T}$, and device instances $\mathcal{D}$, generate resource configuration $r_{s,d}$ $\forall s, d$ within a time budget such that $\sum_s \sum_d r_{s,d}$ is minimized and $\forall s \in S$ meets $\mathcal{R}_\mathcal{A}$ and $\mathcal{R}_\mathcal{P}$*

We consider an example network-wide scenario that operators want to optimize the resource usage of the sketch-based telemetry while meeting their accuracy and performance requirements. Since the telemetry capabilities share the same infrastructure with other network services [59]–[61], it is beneficial to save resources for these concurrent services. Further, operators can also choose to optimize towards other objectives; e.g., minimizing the computation overhead of the deployed telemetry algorithms as discussed in the Extension of Problem 2, or maximize the accuracy guarantees over allowable resources and performance requirements.

Specifically, given a set of queries $\mathcal{Q}$, each with associated accuracy and performance requirements, traffic workload characteristics and a network topology, the operator's high-level goal is to deploy appropriate sketches across the deployment such that SLAs are met while minimizing overall resource usage. As a non-sketch example, recent efforts [62] focus on reusing data collected for one query for other queries. The operator ideally wants to view their deployment under as "one-big-switch" without worrying about manually distributing sketches across the various devices in the deployment to ensure appropriate correctness and coverage.

Realizing this conceptual goal in an end-to-end workflow is challenging as it requires addressing a number of sub-challenges. We briefly mention these sub-challenges now and discuss in more detail in the following subsections:

- **Problem 3:** Translate each $q \in \mathcal{Q}$ to appropriate sketch definitions with conservative (traffic-oblivious) memory configurations S to meet accuracy requirements $\mathcal{R}_\mathcal{A}$. We need to solve this problem to construct the appropriate input for the optimization in Problem 2.
- **Problem 4:** Given a heterogeneous network deployment, develop optimal device-specific sketch implementations, given sketch definitions and configurations $s \in S$. As the next step, we need to deploy sketch implementations to heterogeneous devices for telemetry in the network based on the output of Problem 2.
- **Problem 5:** Given traffic workload characteristics $\mathcal{W}_\nabla$, optimize the memory configuration of each sketch to provide a better memory-accuracy tradeoff and further reduce resource usage. This problem is useful in achieving optimized network-wide resource usage in Problem 2.
- **Problem 6:** Once sketches are deployed on device d, verify their correctness to ensure the expected accuracy requirements $\mathcal{R}_\mathcal{A}$ are met. After sketches are deployed, it is important for operators to verify that their telemetry tools can provide accurate results.



Strategy		Device			Total Resources
		A	B	C	
Network-wide UnivMon	Placement	CM1	CM2	CM3	
	Resources required	4 cores, 100 KB	2 cores, 100 KB	100 KB	6 cores, 300 KB
Better Strategy	Placement	-	CM2, CM3	CM1	
	Resources required	2 cores, 0 KB	2 cores, 200 KB	100 KB	4 cores, 300 KB

Fig. 4: Example of network-wide UnivMon not optimally placing the sketches.

Limitations of existing network-wide solutions: While prior work presented an early version of a network-wide solution [14] for resource optimization, it does not take traffic workload characteristics, different types of sketches, and the heterogeneity of the devices into account, and can converge to a sub-optimal or even infeasible sketch placement and resource allocation. Figure 4 shows a simple scenario where network-wide UnivMon [14] does not optimally place three Count-Min sketch instances in a topology of three programmable devices. Specifically, in this example, the operator wants to know the 5-tuple heavy hitters over traffic between devices A and C (CM1) and the heavy hitters over traffic between devices B and C separately for (SrcIP, SrcPort) and (DstIP, DstPort) flow keys (CM2 and CM3). Resource optimization approaches will decide which sketch will be placed on which device while being aware of the resources required for these sketches given different performance requirements for different devices: (1) UnivMon, which is unaware of the interaction between performance requirements and resource usage, tries to balance memory usage by placing a sketch on each device. This results in placing a sketch on device A which sees 20Mpps traffic.

In order to accommodate a sketch and support this forwarding rate, device A requires 4 cores. (2) A better strategy shifts telemetry load towards device B[1] which sees less traffic and can accommodate 2 sketches while meeting the 10Mpps requirement. Device A in this strategy does not maintain a sketch and only needs 2 cores to maintain 20 Mpps traffic forwarding. Note: Device C's compute resources are the same in both strategies and hence are not shown.

Extension of Problem 2: *[Maximum Performance] Given a set of queries $\mathcal{Q}$ with requirements $\mathcal{R}_\mathcal{A}$, topology $\mathcal{T}$, and devices $\mathcal{D}$, output resource configuration $r_{s,d}$ for all s, d such that $\sum_s \sum_d l_{s,d}$ is minimized and $\forall s \in S$ meets $\mathcal{R}_\mathcal{A}$*

This extension aims at providing optimized network-wide sketch placement and resource allocation that meets the device resource constraints and minimizes total packet processing overhead. In this optimization, we aim at deploying a telemetry solution to handle the largest possible volume of traffic for given queries, which potentially offers us the ability to monitor bursty traffic. Meanwhile, this type of optimization is useful for operators to control the maximum volume of traffic that goes into the telemetry infrastructure.

B. Network Operator & Algorithm Designer

Problem 3*: [Queries to Sketch Definitions] Design a compiler that translates queries $\mathcal{Q}$ into sketch definitions and configurations S that meet accuracy requirements $\mathcal{R}_\mathcal{A}$*

Our focus is on translating telemetry queries into a set of *practical* sketch definitions with memory configurations satisfying the accuracy requirements from the queries, irrespective of traffic workload characteristics and hardware platforms. This is possible because the accuracy guarantees of sketches are hardware agnostic and only depend on the memory configuration. Thus, one can potentially leverage the theoretical analysis from algorithm designers to provide traffic-oblivious sketch resource configurations. For example, if a query specifies a heavy hitter task with 98% accuracy and 0.99 confidence level, we envision a compiler to generate a platform-agnostic sketch configuration (e.g., Count-Min Sketch with $r \times d$ counters) that maintains errors $\leq 2\%$ with 0.99 probability under any workload distribution.

This is the first step towards network-wide device-aware resource management, which requires target-agnostic memory configurations treating the network-wide topology as a "one-big-switch" and corresponding performance characteristics on each hardware target as input.

Extension of Problem 3: *[Expressiveness] If the network operator's telemetry queries $\mathcal{Q}$ cannot be compiled to S, can algorithm designers develop new sketching algorithms to address the failures?*

[1]Device B runs in a CPU polling mode.

While there have been significant advances in developing sketches for various telemetry tasks, the intents of network operators may still fall outside those of existing sketching algorithms, or the theoretical design of a sketch is infeasible in hardware targets. For instance, existing sketches do not support to measure the packet loss inside a switch. Sliding window-based sketches [13], [31], [63] are infeasible in current RMT programmable switches [64]. We need algorithm designers to step in and come up with improved or new sketches. Meanwhile, the theory community has already developed a rich pool of sketching tools that may be relevant to the operator's needs. The challenge lies in how to effectively collect and formulate these requirements to motivate algorithm designers to develop new algorithms or disprove the feasibility. We expect telemetry systems to be a great channel to collect these telemetry query failures and report back to operators and relevant algorithm designers.

C. Algorithm Designer & Platform Vendor/Developer

Problem 4*: [Sketch Implementation] Given a sketch configuration $s \in S$ with device d, generate a sketch implementation $c_{s,d}$ to minimize the actual resource usages $r_{s,d}$ and $l_{s,d}$*

Ideally, we want to generate optimized platform-specific sketch implementations for any sketch configuration and device type. Today, this requires significant effort from both platform vendors/developers and algorithm designers to deliver optimized sketch implementation per hardware target [10], [14], [16], [34]. What is missing today are tools (e.g., optimizing compilers) to take as input an algorithm definition and configuration defined in a high-level language, and automatically output an implementation that is optimized for a particular hardware target. With such a tool, algorithm designers will not need to worry about how to implement a current or future sketching algorithm into the hardware architecture and platform developers will not worry about understanding the algorithmic details in order to implement the sketches. Existing efforts on P4 language and its target-specific compilers are expected to contributing in this direction. Unfortunately, our benchmark demonstrates that existing sketch implementations on programmable switches using P4 are far from resource-efficient (Table II)[2]. Compared to a fully functional switch implementation (`switch.p4`), existing sketches use excessive switch hardware resources (e.g., up to $15\times$ more hash function calls and $17\times$ more stateful ALUs).

Recent efforts have focused on performance bottlenecks of sketching algorithms run inside virtual software switches [10], [15], [17]. While they address the compute/memory bottlenecks in various software sketch implementations, their ideas do not directly transfer to other hardware platforms. For instance, NitroSketch [17] increases the memory footprint to

[2]Sketch configurations in the table, R:rows, C:columns, and L:levels. CountSketch(R=5, C=2048), UnivMon(L=16, R=5, C=2048), R-HHH(L=25, R=5, C=2048), SketchLearn(L=112, R=1, C=2048).

Resource	CountSketch	UnivMon	R-HHH	SketchLearn
Match Crossbar	10.0%	177.2%	476.9%	347.7%
SRAM	3.5%	56.3%	88.4%	78.9%
Hash Bits	3.7%	59.6%	91.6%	82.1%
Hash Calls	62.5%	1100.0%	1562.5%	700.0%
Stateful ALUs	71.4%	1142.9%	1785.7%	1600.0%

TABLE II: Additional H/W resource usage in Barefoot Tofino by existing sketch implementations. The numbers are normalized by the usage of baseline `switch.p4`

reduce CPU consumption, but the key resource constraints in hardware context are different (e.g., processing stages, ALU, and hash function calls) [23]. SketchVisor [10] and ElasticSketch [15] split a sketch into a fast path and a slow path, and use the fast path to accelerate the packet processing. This type of idea is not particularly useful in hardware switches where all packet operations should stay in the fast path [64]. In addition, other efforts such as PRECISION [65] and HashPipe [16] propose heavy hitter algorithms for programmable switches but they are either trading off packet processing performance or measurement accuracy for data plane compatibility.

Extension of Problem 4: *[Multi-Sketch–Implementation] Given all sketch configurations $s \in S$ and device instance d, generate a consolidated sketch implementation c_d for device d such that the actual device resource usage r_d and performance latency l_d are minimized?*

This extension is about optimizing the sketch implementation on a device when multiple sketch instances are present. Our observation is that many sketches share common primitive operations (hash computation, counter updates, etc.), and we expect that the actual resource usage and packet processing performance on a device d^* can be further optimized to less than $\sum_s r_{s,d^*}$ and $\sum_s l_{s,d^*}$.

A recent proposal [66] shows the promises of using program synthesis to auto-generate fast processing hardware implementations on programmable switches using fewer hardware resources. While this direction is promising in general for Problem 4 and its extension, this work demonstrates benefits in one particular hardware architecture and we would like to see if a similar approach can be designed for other platforms and how many more resources it can save.

D. Network Operator & Platform Vendor/Developer

Problem 5*: [Sketch Configuration] Given a set of traffic workload characteristics $\mathcal{W}_\nabla$ and traffic-oblivious sketch configurations S that meet accuracy requirements $\mathcal{R}_\mathcal{A}$, output a minimal platform-agnostic memory configuration for $\forall s \in S$ that meets the accuracy requirement*

This problem entails finding a minimal memory configuration that meets a certain accuracy requirement for a sketch and a given type of traffic workload characteristics (e.g., skewness, number of flows). Problem 3 attempts to provide a traffic-oblivious memory configuration for the sketch to meet the accuracy requirement under any workloads. For platform vendors, it is of importance to fully understand the resource-accuracy usage of the user functions running atop their platforms and to continue improving cost-efficiency of their architecture. In practice, network operators shall have basic understanding and expectation about the workloads such as skewness and distribution, and the traffic-oblivious configuration may not be tight anymore. For example, Count Sketch can achieve better memory-accuracy tradeoff if the workload is skewed following some Zipfian distribution [8].

SketchLearn [11] leverages automated statistical inference to actively "learn" the traffic workload characteristics to configure its sketch on the fly, relieving the user burdens in the sketch memory configuration. While a learning-based approach is promising in resolving this problem, SketchLearn did not tackle the configurations of other types of sketches and it is difficult to fit its data structure inside a programmable switch target. We are unsure whether the model inference used in SketchLearn is an optimal choice.

Problem 6*: [Verification] Given sketch implementation c_d on device d, ensure that c_d will correctly meet the accuracy requirements when running on d?*

Once sketch implementations have been deployed to various devices, one question is that whether the on-device sketch instances will work as expected. Specifically, when an adversary is present, network operators want to verify the integrity of the sketch instances such that the output is correctly reflecting the network traffic conditions. We can think of this verification in two aspects: (1) Operators can naturally verify the accuracy of sketches if the integrity of the on-device sketch instance is guaranteed. (2) If such integrity is not guaranteed, operators need to identify the occurrences when sketches failed to meet the accuracy requirements. Current platform vendors have been on an active race to offer secure enclave primitives such as Intel SGX [38], AMD SEV [39], and ARM TrustZone [40] for mapping arbitrary functions to trusted memory. It remains an open challenge on how to leverage secure hardware capabilities as the "root-of-the-trust" for sketch-based telemetry. However, if secure hardware primitives are not present, the problem becomes how to add redundancy in the sketch deployment to maintain sufficient measurement accuracy even when some devices are offline.

Existing efforts [67]–[69] demonstrate the promises of protecting network functions with hardware enclaves (e.g., Intel SGX). However, those efforts are not capable of sketch-based telemetry because (1) sketches require high throughput guarantees while existing frameworks such as SafeBricks [68] and SGX-Box [67] incur high processing overhead, and (2) these efforts are designed for general-purpose network functions where redundant modules and complexities are included.

E. A Campus Network Telemetry Example

We demonstrate a sketch-based telemetry system workflow using university campus network as an example and then generalize it to a future roadmap in the next section.

(1) Campus IT department is the network operator, who specifies a range of application-specific queries (e.g., heavy

network users, DDoS detection, and anomalies) with accuracy/resource requirements using a high-level language. The corresponding telemetry capabilities are deployed across the campus and continuously report trustworthy results to the IT even when some switch software has been hacked. The IT can tell both researcher A (algorithm designer) in the university and hardware vendors about missing supported queries.

(2) Researcher A receives the requirements and finds the new queries interesting to pursuit. Then, Researcher A and collaborators design a new sketch and add it to the telemetry library. They deploy their algorithm in the campus testbed to get feedback about the accuracy and performance, and discover the missing of a hardware feature. They inform the IT and the vendor about this missing feature.

(3) Vendor B receives some feedback and may agree to add new software or hardware primitives to support new telemetry algorithm implementations.

IV. A Future Roadmap

We envision a sketch-based telemetry framework as depicted in Figure 5, assuming that research challenges P1-P6 and others have been properly addressed by the community. In this framework, we expect a *management interface* that has an expressive front-end/API to interact with network operators, algorithm designers, and platform vendors/developers. Some key components in the interface are 1) query compiler to translate operator intents into sketch configurations, and 2) sketch library to maintain state-of-the-art sketching definitions/implementations. In the *control plane*, there will be a network-wide resource manager taking input from the management interface and computing an optimized sketch placement and resource allocation based on the requirements. In the *data plane*, the optimized and verified sketch instances will be initialized across a network of heterogeneous devices based on the resource management decisions from the control plane.

We expect network operators, algorithm designers, and platform vendors/developers will have a way interacting with each other and the telemetry framework as follows:

- **Network operators:** Operators can specify telemetry needs via the management interface and receive the intended telemetry metrics via API or return "infeasible" for further negotiation. In the back-end, operator queries are translated to sketch configurations and their related device-level implementations to be deployed. In addition, operators can also describe their intents to algorithm designers and platform vendors/developers to cover unsupported telemetry tasks.
- **Algorithm designers:** Algorithm designers can obtain new telemetry capability requests from operators and design new algorithms based on the requests. They can then add their new algorithms to the sketch ecosystem and get feedback about their implemented and evaluated algorithms in real-world scenarios.
- **Platform vendors and developers:** Platform vendors can receive new hardware capabilities requests, deliver new hardware capabilities, and update the device specifications accordingly. Platform developers can explore the sketch algorithm definitions and hardware capabilities in the sketching ecosystem, and deliver improved or new implementations to the sketch library.

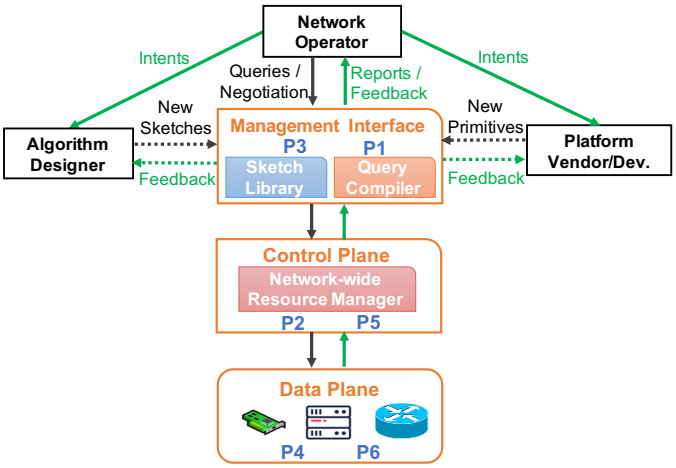


Fig. 5: Sketch-based telemetry framework and stakeholder interactions.

Prior efforts have laid the groundwork for designing sketches and making them transition from a theoretical curiosity to a promising start for network telemetry. We hope that our vision, research challenges, and collaborative efforts from the stakeholders taken together can help transition sketch-based telemetry into "prime time" deployment in the era of programmable networks.

Acknowledgements: This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the NSF/VMware Partnership on Software Defined Infrastructure as a Foundation for Clean-Slate Computing Security (SDI-CSCS) program under Award No. CNS-1700521, and NSF award CNS-1565343.

References

[1] Cisco, "Introduction to cisco ios netflow," 2012.
[2] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *Proc. of IEEE ICDCS*, 2004.
[3] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," in *Proc. of ACM SIGCOMM*, 2003.
[4] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. of ACM SIGCOMM*, 2002.
[5] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. of ACM IMC*, 2003.
[6] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *RANDOM/APPROX*. Springer, 2002.
[7] R. B. Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard, "Constant time updates in hierarchical heavy hitters," in *Proc. of ACM SIGCOMM and CoRR/1707.06778*, 2017.
[8] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. of ICALP*, 2002.
[9] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications," *J. Algorithms*, 2005.
[10] 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. of ACM SIGCOMM*, 2017.

[11] Q. Huang, P. P. Lee, and Y. Bao, "Sketchlearn: Relieving user burdens in approximatemeasurement with automated statistical inference," in *Proc. of ACM SIGCOMM*, 2018.

[12] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *POMACS (ACM SIGMETRICS)*, 2019.

[13] N. Ivkin, R. B. Basat, Z. Liu, G. Einziger, R. Friedman, and V. Braverman, "I know what you did last summer: Network monitoring using interval queries," *POMACS (ACM SIGMETRICS)*, 2019.

[14] 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. of ACM SIGCOMM*, 2016.

[15] 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. of ACM SIGCOMM*, 2018.

[16] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. of ACM SOSR*, 2017.

[17] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. of ACM SIGCOMM*, 2019.

[18] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. of USENIX NSDI*, 2013.

[19] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang, "An empirical evaluation of entropy-based traffic anomaly detection," in *Proc. of ACM IMC*, 2008.

[20] Z. Liu, S. Zhou, O. Rottenstreich, V. Braverman, and J. Rexford, "Memory-efficient performance monitoring on programmable switches with lean algorithms," *Proc. of SIAM/ACM APoCS*, 2019.

[21] R. Schweller, A. Gupta, E. Parsons, and Y. Chen, "Reversible sketches for efficient and accurate change detection over network data streams," in *Proc. of ACM IMC*, 2004.

[22] "Data plane developer kit (dpdk)," https://software.intel.com/en-us/networking/dpdk.

[23] "Barefoot Tofino," https://barefootnetworks.com/products/brief-tofino/.

[24] "Broadcom Trident 3," https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series/.

[25] "Mellanox SmartNIC," https://www.mellanox.com/products/smartnic.

[26] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," in *Proc. of ACM SIGCOMM*, 2014.

[27] Y. Li, H. L. Nguyen, and D. P. Woodruff, "Turnstile streaming algorithms might as well be linear sketches," in *Proc. of ACM STOC*, 2014.

[28] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Practical sketching algorithms for low-rank matrix approximation," *SIAM Journal on Matrix Analysis and Applications*, 2017.

[29] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM journal on computing*, 2002.

[30] V. Braverman and R. Ostrovsky, "Smooth histograms for sliding windows," in *Proc. of IEEE FOCS*, 2007.

[31] V. Braverman, R. Ostrovsky, and A. Roytman, "Zero-one laws for sliding windows and universal sketches," in *Proc. of APPROX/RANDOM*, 2015.

[32] G. Cormode, "The continuous distributed monitoring model," *ACM SIGMOD Record*, 2013.

[33] D. P. Woodruff and Q. Zhang, "Tight bounds for distributed functional monitoring," in *Proc. of ACM STOC*, 2012.

[34] M. Yang, J. Zhang, A. Gadre, Z. Liu, S. Kumar, and V. Sekar, "Joltik: enabling energy-efficient" future-proof" analytics on low-power wide-area networks," in *Proc. of ACM MobiCom*, 2020.

[35] "Barefoot P4 Studio," https://www.barefootnetworks.com/products/brief-p4-studio/.

[36] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, 2014.

[37] "Netronome: Agilio SmartNICs and Software," https://fd.io/technology/.

[38] Intel SGX, https://software.intel.com/en-us/sgx.

[39] AMD, "Secure Encrypted Virtualization," https://developer.amd.com/.

[40] ARM TrustZone, https://developer.arm.com/ip-products/security-ip/trustzone.

[41] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proc. of ICDT*, 2005.

[42] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi, "Mergeable summaries," in *Proc. of ACM SIGMOD*, 2012.

[43] E. Gan, J. Ding, K. S. Tai, V. Sharan, and P. Bailis, "Moment-based quantile sketches for efficient high cardinality aggregation queries," *arXiv preprint arXiv:1803.01969*, 2018.

[44] Apache, "Apache druid," https://druid.apache.org/.

[45] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches," in *Proc. of USENIX Security*, 2021.

[46] B. Ghazi, R. Panigrahy, and J. R. Wang, "Recursive sketches for modular deep learning," in *Proc. of ICML*, 2019.

[47] N. Ivkin, D. Rothchild, E. Ullah, I. Stoica, R. Arora *et al.*, "Communication-efficient distributed sgd with sketching," in *Proc. of NeurIPS*, 2019.

[48] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proc. of ACM SIGMOD*, 2018.

[49] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. of ACM STOC*, 1996.

[50] P. Clifford and I. Cosma, "A simple sketching algorithm for entropy estimation over streaming data," in *Proc. of AISTATS*, 2013.

[51] P. Flajolet, ric Fusy, O. Gandouet, and et al., "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *AOFA*, 2007.

[52] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proc. of ACM SIGCOMM*, 2018.

[53] Y. Yuan, D. Lin, A. Mishra, S. Marwaha, R. Alur, and B. T. Loo, "Quantitative network monitoring with netqre," in *Proc. of ACM SIGCOMM*, 2017.

[54] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proc. of ACM SIGCOMM*, 2017.

[55] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: Continuous dataflow processing," in *Proc. of ACM SIGMOD*, 2003.

[56] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: a stream database for network applications," in *Proc. of ACM SIGMOD*, 2003.

[57] K. Borders, J. Springer, and M. Burnside, "Chimera: A declarative language for streaming network traffic analysis," in *Proc. of USENIX Security*, 2012.

[58] R. Alur, D. Fisman, and M. Raghothaman, "Regular programming for quantitative properties of data streams," in *European Symposium on Programming*, 2016.

[59] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "Distcache: Provable load balancing for large-scale storage systems with distributed caching," in *Proc. of USENIX FAST*, 2019.

[60] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "Tea: Enabling state-intensive network functions on programmable switches," in *Proc. of ACM SIGCOMM*, 2020.

[61] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proc. of ACM SIGCOMM*, 2017.

[62] S. J. Saidi, A. Maghsoudlou, D. Foucard, G. Smaragdakis, I. Poese, and A. Feldmann, "Exploring network-wide flow data with flowyager," *IEEE Transactions on Network and Service Management*, 2020.

[63] M. Gabel, D. Keren, and A. Schuster, "Anarchists, unite: Practical entropy approximation for distributed streams," in *ACM KDD*, 2017.

[64] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proc. of ACM SIGCOMM*, 2013.

[65] R. Ben-Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Efficient measurement on programmable switches using probabilistic recirculation," in *Proc. of IEEE ICNP*, 2018.

[66] X. Gao, T. Kim, M. D. Wong, D. Raghunathan, A. K. Varma, P. G. Kannan, A. Sivaraman, S. Narayana, and A. Gupta, "Switch code generation using program synthesis," in *Proc. of ACM SIGCOMM*, 2020.

[67] J. Han, S. Kim, J. Ha, and D. Han, "Sgx-box: Enabling visibility on encrypted traffic using a secure middlebox module," in *APNet*, 2017.

[68] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "Safebricks: Shielding network functions in the cloud," in *USENIX NSDI*, 2018.

[69] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Shieldbox: Secure middleboxes using shielded execution," in *Proc. of ACM SOSR*, 2018.