Raising the Level of Abstraction for Sketch-Based Network Telemetry with SketchPlan

Milind Srivastava*
Carnegie Mellon University
Pittsburgh, PA, USA

Shao-Tse Hung
National Yang-Ming Chiao Tung University
Taipei, Taiwan

Hun Namkung
Carnegie Mellon University
Pittsburgh, PA, USA

Kate Ching-Ju Lin
National Yang-Ming Chiao Tung University
Taipei, Taiwan

Zaoxing Liu
University of Maryland
College Park, MD, USA

Vyas Sekar
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

While sketch-based network telemetry is attractive, realizing its potential benefits has been elusive in practice. Existing sketch solutions offer low-level interfaces and impose *high effort* on operators to satisfy telemetry intents with required accuracies. Extending these approaches to reduce effort results in *inefficient* deployments with poor accuracy-resource tradeoffs. We present SketchPlan, an abstraction layer for sketch-based telemetry to reduce effort and achieve high efficiency. SketchPlan takes an *ensemble view across telemetry intents and sketches*, instead of existing approaches that consider each intent-sketch pair in isolation. We show that SketchPlan improves accuracy-resource tradeoffs by **up-to 12x** and **up-to 60x** vs. baselines, in single-node and network-wide settings. SketchPlan is open-sourced at: https://github.com/milindsrivastava1997/SketchPlan.

CCS Concepts

• **Networks** → **Network measurement**; **Programmable networks**; **Network monitoring**; **In-network processing**.

Keywords

Sketching algorithms; Network measurement; Network telemetry

ACM Reference Format:
Milind Srivastava, Shao-Tse Hung, Hun Namkung, Kate Ching-Ju Lin, Zaoxing Liu, and Vyas Sekar. 2024. Raising the Level of Abstraction for Sketch-Based Network Telemetry with SketchPlan. In *Proceedings of the 2024 ACM Internet Measurement Conference (IMC '24), November 4–6, 2024, Madrid, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3646547.3689016

1 Introduction

Network telemetry is essential to observe traffic and drive downstream management tasks such as traffic engineering and anomaly detection. Operators specify telemetry intents to measure statistical properties of network traffic in different dimensions such as "the number of unique 5-tuples in the network" or "top 10 source IPs sending traffic to the network".

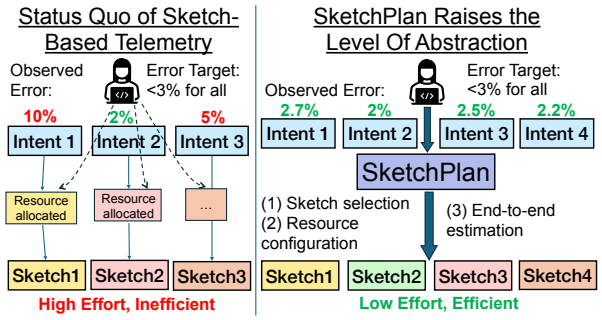


Figure 1: SketchPlan raises the level of abstraction for sketch-based telemetry.

While sketches [7, 10, 15] are a promising solution to reduce resource usage and thus telemetry costs, practically realizing the benefits of sketches is challenging on two fronts:

- *High operator effort:* To meet intents' accuracy needs, existing solutions require *manual effort* to understand algorithms, map each high-level intent to suitable sketches, and configure sketch resources appropriately. Even recent novel sketches [20, 27, 31, 33], sketch optimizations [19, 23, 32], and data plane frameworks (e.g., Sketchovsky [24], HeteroSketch [4]) lack high-level interfaces.
- *Inefficient configuration:* Extending existing approaches to provide a high-level interface, results in *inefficient* accuracy-resource tradeoffs. A key reason is that existing solutions [4, 23, 24, 28] view each intent in isolation. Thus, these are fundamentally *inefficient* for practical use where operators care about measuring and optimizing multiple intents jointly, not just one intent.

We argue that reducing operator effort requires *raising the level of abstraction* of sketch-based telemetry. Instead of requiring operators to manually map their telemetry intents and configure sketches, we envision an abstraction layer called SketchPlan. In SketchPlan, the operator specifies high-level intents of interest with accuracy targets, such as measuring the entropy and cardinality of 5-tuples with 90% and 95% accuracy, respectively. SketchPlan automatically maps these intents to sketches with appropriate resource configurations.

By raising the level of abstraction, SketchPlan also enables novel opportunities for *efficiency* via an *ensemble view* of all intents and

*Corresponding author: milindsr@andrew.cmu.edu

IMC '24, November 4–6, 2024, Madrid, Spain

ACM ISBN 979-8-4007-0592-2/24/11
https://doi.org/10.1145/3646547.3689016

sketches. Instead of viewing each intent-sketch combination in isolation, SketchPlan leverages inherent opportunities for cross-intent and cross-sketch optimization to offer significantly better resource-accuracy trade-offs. For instance, MRAC [17] and CountSketch [7] may be optimized to measure entropy and heavy hitters in isolation; but given both intents, a single UnivMon [20] instance may be optimal.

We address key technical challenges in realizing SketchPlan's vision. We provide the first practical *formulation* for mapping an ensemble of intents to sketches. SketchPlan implements an extensible *sketch coverage map* capturing the coverage capability of sketches for various queries. To achieve desirable accuracy-resource tradeoffs for the intent ensemble, SketchPlan uses offline *error-resource profiles* that characterize different query-sketch maps. We show a practical and robust mechanism for generating such profiles. Finally, we show how SketchPlan provides benefits in practical single-node and network-wide settings.

We evaluate SketchPlan on diverse operator intents and deployment regimes. SketchPlan improves accuracy-resource tradeoffs over strawman solutions by **up-to 12x** on a single node and **up-to 60x** in network-wide settings; successfully providing a *high-level and efficient* sketch abstraction layer. We show that SketchPlan's benefits are robust in the face of progressively-older error-resource profiles and that SketchPlan's design is crucial to providing efficient sketch-based telemetry.

2 Motivation and Related Work

We illustrate how the status quo in sketch-based telemetry imposes significant effort and inefficiency on operators, and why prior work does not address these problems.

2.1 Background on Sketch-based Telemetry

Operators want to specify high-level telemetry intents to measure statistical properties that drive their monitoring use cases. Each intent specifies a query to measure and an optional accuracy target. Each query specifies a statistic/metric to measure on a certain flow-key. Operators may also specify bounds on data plane resources to be used for telemetry. For example, an operator may want to measure the entropy and cardinality of traffic with 90% and 95% accuracy, respectively, with only 512KB memory resources. Accuracy targets are driven by requirements from downstream use cases.

Since processing all traffic is expensive and traditional sampling techniques can be ineffective, sketches emerged as a promising telemetry solution. A sketch is a randomized approximation algorithm to summarize data streams such as network traffic, and generate approximate estimates of various statistics. For instance, CountMin [10] can identify large network flows (heavy hitter) while MRAC [17] can estimate entropy.

2.2 Illustrative Scenario

To understand the limitations of the status quo, consider a simplified operational use case. An operator in a Network Operations Center [3] wants to monitor the network for traffic engineering and anomaly detection tasks. To support these, they are interested in measuring the *cardinality*, *entropy* and top 50 *heavy hitter* flows with some target accuracies.

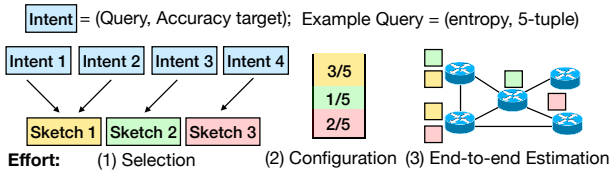


Figure 2: Practically leveraging sketches to estimate intents requires high operator effort.

Prior work	Selection	Configuration	End-to-end
Novel sketches [20, 27, 31]	X	X	X
Optimized data planes [19, 23, 24, 32]	X	X	X
Network-wide orchestration [4]	X	X	✔
Sketch-based compilers [18, 28]	X	✔	X
SketchPlan	✔	✔	✔

Table 1: Prior work in sketches provides operators with low-level interfaces that entail high effort.

Supporting these intents using sketches is easier said than done. Let us conceptually walk through the effort that an operator must exert today to estimate these intents [21] (Figure 2).

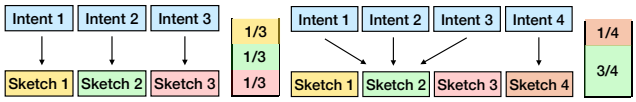
- *Sketch selection:* The operator must manually search the sketch literature to find specific sketches that support each intent. The design space here is large and complicated – should they pick a query-specific sketch like MRAC [17] for entropy, or should they pick a "general" sketch like UnivMon [20]. Either way, there are multiple query-specific and general sketches that the operator must pick from. With no guidelines to follow, the operator is forced to speculate on the sketches to deploy. Poor sketch selection can undermine or even nullify the value of using sketches (see §4).
- *Configuration:* They must speculate about which sketch is efficient in their deployment, and guess the amount of data plane resources to allocate to each sketch. Again, there is little support in existing literature to guide such configurations, and how theoretical error guarantees may translate into workload-specific empirical behavior. Poor configuration can lead to estimation errors that cause cascading failures in the operator's downstream tasks.
- *End-to-end estimation:* The operator needs end-to-end estimates of their intents across the network and across various traffic sub-populations of interest such as specific ports and origin-destination pairs. While there are some recent efforts on this front [4], they do not address the first two requirements, entail high effort and are inefficient.

2.3 Related Work and Limitations

Existing work in sketch-based telemetry can be classified into four broad themes that we discuss below.

Novel sketches: There are several novel sketches for supporting multiple metrics [20, 27, 31], or multiple flow-keys with the same data plane implementation [6, 9, 33]. While valuable, these works provide low-level interfaces that do not address the selection and configuration tasks discussed above.

Optimized data plane: Recent efforts [11, 19, 23, 24, 32] have proposed optimizations to existing sketch implementations or collections of sketches, e.g., Sketchovsky [24] re-uses data plane resources



(a) ⟨*AnyFeasible, Uniform*⟩ **(b)** ⟨*Greedy, Proportional*⟩

Figure 3: Strawman solutions for sketch selection and configuration offer poor accuracy-resource tradeoffs.

Approach	Sketches	Resource Usage	Errors
⟨*AnyFeasible, Uniform*⟩	HLL: 256KB MRAC: 256KB	512KB	0.7%, 9.7%
⟨*Greedy, Proportional*⟩	UnivMon: 512KB	512KB	18.4%, 2.7%
SketchPlan	UnivMon: 320KB MRB: 128KB	**448KB**	**2.8%, 0.4%**

Table 2: Toy scenario showing inefficiency of strawman solutions.

across multiple sketches to reduce resource overhead. These works too require the operator to manually select and configure sketches.

Network wide orchestration: HeteroSketch [4] orchestrates sketches in a network-wide fashion, but provides a low-level interface, requiring operators to specify the sketches to deploy and their resource configurations.

Sketch compilers: AutoSketch [28] compiles a stateful operation like *reduce* or *distinct* to a sketch. However, it does not address operator effort in selection and end-to-end estimation, and considers each intent in isolation, leading to poor accuracy-resource tradeoffs. MAFIA [18] requires operators to select and configure sketches manually.

3 SketchPlan's Design

Case for an Abstraction Layer: In the previous section, we saw that while sketches are a powerful capability, current sketch-based solutions offer low-level interfaces that require high operator effort. To bridge the significant disconnect between an operator's high-level intents and the existing low-level sketch-based offerings, we argue for a *sketch abstraction layer* for operators to specify high-level intents. Here, each intent specifies a query to measure and an accuracy target.

3.1 Strawman Solutions and Limitations

Building this abstraction layer entails two parts: (1) *Intent-to-sketch Selection:* Determining sketches to support an ensemble of intents; and (2) *Resource Configuration:* For each selected sketch, we need to allocate data plane resources.

We consider a few seemingly-natural solutions for sketch selection and configuration. Intent-to-sketch selection can be done by (1) ⟨*AnyFeasible*⟩: randomly picking a sketch capable of estimating an intent, or (2) ⟨*Greedy*⟩: using a general sketch like UnivMon [20] to cover as many intents as possible and ⟨*AnyFeasible*⟩ for other intents. Resource configuration can be done by (1) ⟨*Uniform*⟩: assigning equal resources to each sketch, or (2) ⟨*Proportional*⟩: assigning resources proportional to the number of intents estimated by a sketch. We can combine these to create four strawman approaches. Figure 3 shows two illustrative strawman combinations.

Consider a scenario where the operator wants to measure *cardinality* and *entropy* with 512KB of memory. Table 2 shows the sketch selection and configuration of Figure 3's strawman approaches, and their poor accuracy in measuring these intents. SketchPlan achieves both better average accuracy and lower resource usage.

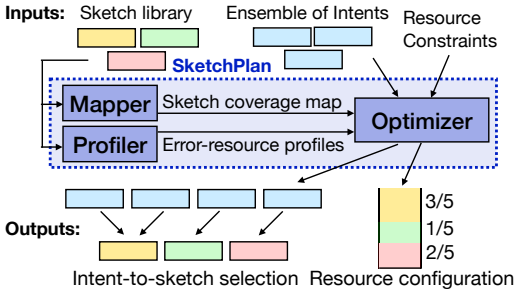


Figure 4: SketchPlan's three modules take an ensemble view of intents and sketches to estimate intents optimally.

3.2 SketchPlan's Key Insight: Ensemble view

Instead of myopically optimizing for each intent in isolation, our approach takes *an ensemble view across intents and sketches.* This allows SketchPlan to leverage cross-intent and cross-sketch interactions, and achieve high efficiency for the entire ensemble. We argue for an ensemble view along two dimensions:

- *Query coverage:* Our first insight is to explicitly map the queries that can be estimated by each sketch. This allows SketchPlan to explore different sketch selections for a given ensemble of intents.
- *Accuracy-resource tradeoffs:* Our second insight is to empirically capture the accuracy-resource tradeoffs for different sketch-query pairs, allowing SketchPlan to explore the space of resource configurations for a set of sketches to deploy.

An ensemble view is beneficial due to three reasons. (1) First, even sketches not designed to be "general" can estimate multiple queries; e.g., CountSketch [7] can estimate heavy hitters, change detection, and entropy. (2) Next, while a sketch could estimate multiple queries, it may not be optimal at estimating all of them. For e.g., UnivMon [20] can estimate both heavy hitters and cardinality but is outperformed by HyperLogLog [15] for cardinality estimation, intuitively explaining why ⟨*Greedy*⟩ strawmen perform poorly. (3) Last, accuracy-resource tradeoffs vary for different queries, and exhibit different shapes; e.g., doubling the resources of a sketch may halve the error of one query, but provide only 10% gain for another. Thus, optimal sketch selection and configuration depends on the queries to measure and available resources; explaining why resource-agnostic strawmen like ⟨*Uniform*⟩ and ⟨*Proportional*⟩ perform poorly.

3.3 Detailed Design

Figure 4 provides an overview of SketchPlan. SketchPlan takes three inputs: (1) a library of sketches, (2) an ensemble of intents, and (3) data plane resource constraints. Each intent specifies a query and an optional accuracy target. Each query specifies a metric to measure, the flow-key of traffic to measure it on, and an OD-pair to measure traffic between. Similar to prior work [4, 20], each OD-pair specifies an origin and destination network node and represents traffic flowing between these. SketchPlan's output is a selected sketch for each intent and resource configurations for each sketch, on each network node.

Notation	Definition
	Inputs
I	Set of intents. Each intent i has a query q and an optional error bound err.
Q	Set of queries. Each query q is a metric, flow-key, and OD-pair tuple (m, f, p).
p_{path}	Set of nodes on the path of OD-pair p.
S	Library of available sketches.
n_B	Resource bound on node n.
obj	Objective function to minimize.
	Outputs
D	Set of sketch deployments. Each sketch deployment $d_{(s,f,n,r)}$ represents the deployment of sketch s configured with flow-key f and resources r on node n.
$map(i)$	Function that maps each intent i to a sketch deployment d.
	Internal functions
$cov(s, q)$	Boolean function representing if sketch s can estimate query q.
$err(q, d)$	Function representing the error in estimating query q by sketch deployment d.

Table 3: Notation for optimization problem formulation.

To produce this output configuration, SketchPlan leverages the three modules described below:

Mapper: This module allows SketchPlan to have an *ensemble view* of query coverage. Given a library of sketches, this module examines the coverage capability of each sketch and generates a sketch-metric coverage map (represented by a bipartite graph similar to Figure 3). By selecting different edges in the map, SketchPlan can explore diverse sketch ensembles to estimate an intent ensemble. §3.5 specifies the entire set of sketches and metrics we implement.

Profiler: This module allows SketchPlan to have an *ensemble view* of accuracy-resource tradeoffs. Given a library of sketches and historical traces, this module generates error-resource profiles for each sketch-metric pair. An error-resource profile captures the estimation error of a sketch for a given metric with changes in resource configuration. To generate these profiles, the Profiler runs sketches offline on the historical traces and considers the median error across traces. Empirical profiling allows SketchPlan's Optimizer (discussed below) to quantitatively compare different intent-to-sketch selections and configurations, and identify the optimal solution.

Optimizer: The Optimizer takes an *ensemble view* of the operator's intents, resource constraints, sketch coverage capabilities, and error-resource tradeoffs. It formulates a network-wide optimization problem that explores deployments where each intent can be estimated by any candidate sketch (as given by the Mapper) and any resource configuration that satisfies its accuracy target (as given by the Profiler). For network-wide intents, the selected sketch can be deployed on any node on its OD-pair's path. The Optimizer's output (Figure 4) is the optimal intent-to-sketch selection and resource configurations for each sketch, on each network node.

Scope: We scope SketchPlan to static configuration of sketches for a given ensemble of intents. SketchPlan only needs to run the Mapper and Profiler once. For a new set of intents, we run the Optimizer to produce an optimal configuration. §6 discusses our future work to dynamically configure sketches in response to rapidly changing intents and traffic.

3.4 Optimization Problem Formulation

Next, we formulate the optimization problem that maps intents to sketches and selects resource configurations. Table 3 defines the symbols used in this formulation. Figure 5 shows the constraints and objective function.

Minimize **O1:** $avg_{i \in I} err(i.q, map(i))$ or **O2:** $\sum_{d \in D} d.r$

subject to:

$$\forall i \in I \mid cov(map(i).s, i.q) == 1 \quad (1)$$

$$\forall i \in I \mid map(i).f == i.q.f \quad (2)$$

$$\forall i \in I \, \exists n \in (i.q.p)_{path} \mid map(i).n == n \quad (3)$$

$$\forall n \in N \mid \Big(\sum_{d \in D \mid d.n == n} d.r \Big) \leq n_B \quad (4)$$

$$\forall i \in I \mid err(i.q, map(i)) \leq i.err \quad (5)$$

Figure 5: Optimization problem solved by SketchPlan.

Inputs: Each intent $i \in I$ has a query q and an optional error bound err. Each query q is a metric, flow-key and OD-pair tuple (m, f, p). Each OD-pair p is specified using a set of devices p_{path}. Additionally, the operator can specify (1) resource bounds n_B for each node n, and (2) the objective function obj to minimize (details below).

Outputs: The output is (1) a set of sketch deployments D, and (2) the function map. Each sketch deployment $d_{(s,f,n,r)} \in D$ denotes a sketch s configured with flow-key f and resources r to be deployed on node n. The map function maps each intent i to a sketch deployment d.

Internal functions: The boolean function $cov(s, q)$, given by the Mapper's output, represents whether sketch s can estimate the query q. The function $err(q, d)$, corresponding to the Profiler, captures the *error-resource profile* i.e. the error in estimating a query q given a sketch deployment $d_{(s,f,n,r)} \in D$. Note that different resource configurations of the same sketch are considered different sketch deployments.

Constraints: (1) Each intent i must be mapped to a sketch deployment that "covers" the intent's query q (Eq. 1). (2) The sketch deployment used to estimate an intent should be configured with the intent's query's flow-key $i.q.f$ (Eq. 2). (3) The sketch deployment should be deployed on a node that is on its OD-pair's path (Eq. 3). (4) The resource usage of sketch deployments on a node must not violate the node's resource bound (Eq. 4). (5) The error for an intent i should be no more than the specified bound $i.err$ (Eq. 5).

Objective function: We consider two goals: the operator may choose to minimize the average error across all intents (**O1**) or the resource usage of all sketch deployments (**O2**).

3.5 Implementation

We implement 5 metrics– heavy hitter, entropy, cardinality, change detection, and flow size distribution; and support any subset of the 5-tuple (src/dst IP/port, protocol) as a valid flow-key. We implement 8 sketches– CountMin [10], CountSketch [7], MRAC [17], LinearCounting [29], MRB [14], LogLog [13], HyperLogLog [15], and UnivMon [20]. Unless mentioned, the Profiler uses 18 traces of 30 sec each, from the CAIDA dataset [1] to empirically measure the error for each sketch-metric pair. It considers the median error across all epochs and derives a single error-resource profile for each sketch-metric. We implement the Optimizer's ILP in Python and use Gurobi's Python binding [2] to solve it.

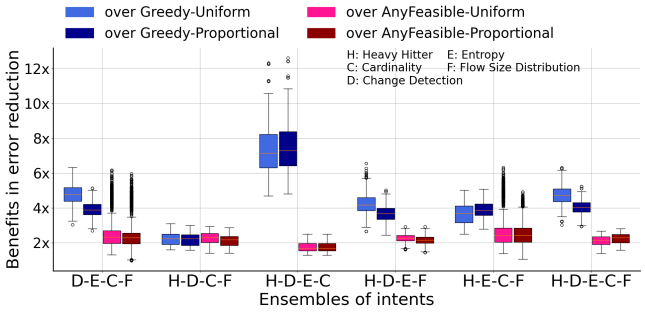


Figure 6: SketchPlan achieves up-to 12x lower error than strawman approaches for various intents.

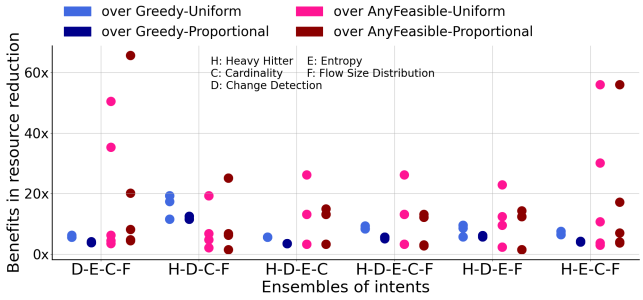


Figure 7: SketchPlan uses up-to 60x lower resources than strawman approaches for various network-wide intent ensembles.

4 Results

Since there are no existing abstraction layers for sketch-based telemetry, we demonstrate SketchPlan's benefits by comparing against the four strawman solutions from §3.1. We show that: (1) Across various intent ensembles, SketchPlan provides **up-to 12x** lower error compared to strawman solutions; (2) SketchPlan's benefits extend to network-wide settings, providing accurate telemetry with **up-to 60x** lower resources than strawman solutions, across various intent ensembles; (3) SketchPlan's benefits are robust to stale error-resource profiles; and (4) SketchPlan's design choices are crucial to achieving these benefits.

Evaluation setup: For each ensemble of intents, we use SketchPlan to select sketches and configure them optimally. We perform experiments on Cloudlab [12] using the `c220g2` node (Intel Xeon Processor E5-2660 v3, 20 2.6 GHz cores). For each strawman approach, we use five different seeds for random sketch selection. Unless mentioned, we evaluate each solution by replaying unseen traces (different from those used by the Profiler) from the CAIDA dataset [1]. We consider 18 epochs, each of 30 sec, with around 20-30M packets and 1M unique flows (based on destination IP and destination port). For each trace, we collect query estimates based on the sketch selection and configuration done by SketchPlan or strawman solutions. We measure query error compared to ground truth,[1] based on standard error measures [31].

Single-node benefits: We compare SketchPlan's benefits over strawman solutions for different ensembles of intents. We associate five metrics from §3.5 with the same flow-key *(dstIP, dstPort)* to generate five intents. We consider all intent ensembles of size 4 and the single ensemble of size 5. For each ensemble, we set a resource bound of 128kB. We run SketchPlan with the objective function to minimize the average error of the intent ensemble (**O1** from §3.4).

Figure 6 shows SketchPlan's benefits measured as the ratio of strawman's average error to SketchPlan's average error for a given ensemble. SketchPlan achieves up-to 12x benefits over strawman solutions for a diverse set of intents with the median benefit varying from 2x to 7x depending on the exact ensemble. While this result is for a 128KB resource budget, we see similar results for other budgets as well. Interestingly, SketchPlan's benefits over ⟨*Greedy*⟩ strawmen are almost always higher than ⟨*AnyFeasible*⟩ strawman; implying that a general sketch may not always be more optimal.

[1]For brevity, for heavy hitters and change detection, we only show results for estimating flow sizes, and not precision/recall of flow-keys

Experiment	Experimental setting	Optimizer runtime
Single-node	Num. of intents = 4	2.8 sec (128KB budget) 5.1 sec (1MB budget)
	Num. of intents = 5	12.9 sec (128KB budget) 21.4 sec (1MB budget)
Network-wide	Num. of intents = 4	128.7 sec
	Num. of intents = 5	1090.5 sec

Table 4: Runtime for SketchPlan's Optimizer.

Network-wide benefits: To illustrate SketchPlan's benefits in a network-wide setting, we consider a Fat-tree network topology [5] of degree 6 and generate 50 random OD-pairs. We consider six intent ensembles as before and associate each intent in each ensemble with all 50 OD-pairs, yielding one ensemble of 250 network-wide intents and five ensembles of 200 network-wide intents.

We evaluate different ensembles with varying accuracy targets and compare SketchPlan's resource usage to that of strawman approaches in Figure 7 (**O2** from §3.4). For the loosest accuracy target, SketchPlan achieves **up-to 60x** benefit in reducing network-wide resource usage compared to strawman solutions[2]. The x-axis shows different intent ensembles. The y-axis quantifies benefit by dividing the strawman's resource usage by SketchPlan's resource usage. Each scatter point denotes a particular strawman solution and seed. Appendix B provides exact numbers for the accuracy targets and shows similar trends for stricter targets as well.

Table 4 shows the runtime of SketchPlan's Optimizer for the single-node and network-wide settings. Since we envision SketchPlan being used in an offline manner by operators, this does not introduce any significant overhead in obtaining telemetry. Even so, insights from prior work [4] could accelerate network-wide planning.

SketchPlan's robustness: A natural question is if SketchPlan's benefits are robust to stale resource-accuracy profiles. To answer this, we train the Profiler on six progressively older CAIDA traces and learn error-resource profiles for each trace. P_1 denotes the profile from the oldest trace, P_6 from the newest. For each profile, we evaluate SketchPlan on the newest trace, corresponding to P_6 for the ensemble with 5 intents. We do this for three different sets of six traces. In Figure 8, the left-most point on the x-axis denotes the oldest profile (P_1 with "age" 5); the right-most point (P_6 with "age" 0) shows the profile trained and tested on the same trace.

[2]We use offline profiles to decide accuracy target compliance. In practice, SketchPlan's errors on unseen traces are similar to those on offline profiles.

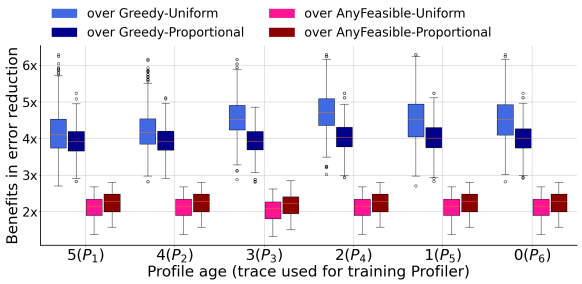


Figure 8: SketchPlan's benefits over strawman solutions are robust in the face of stale error-resource profiles.

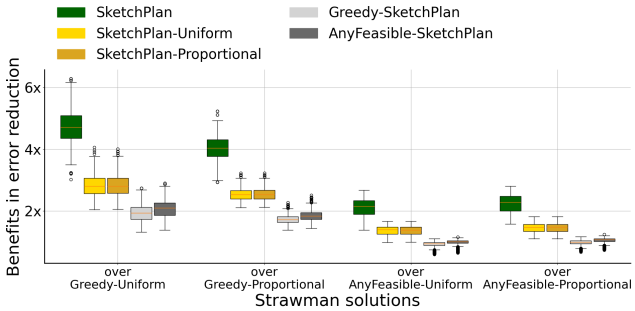


Figure 9: SketchPlan's design is crucial to its benefits.

These results show that SketchPlan's offline Profiler can still offer significant benefits robust to workload changes over small time scales. We defer SketchPlan's evaluation on, and robustness to, longitudinal changes to future work (§6).

Design benefits: To examine the benefits of SketchPlan's Mapper and Profiler, we enable one of these modules at a time and replace the other with naïve strawmen. We compare SketchPlan with *i)* ⟨*AnyFeasible*, *SketchPlan*⟩, *ii)* ⟨*Greedy*, *SketchPlan*⟩, *iii)* ⟨*SketchPlan*, *Uniform*⟩, and *iv)* ⟨*SketchPlan*, *Proportional*⟩. *i)* and *ii)* replace the Mapper with naïve strawmen. *iii)* and *iv)* replace the Profiler with naïve strawmen. Figure 9 compares SketchPlan vs. intermediate versions of SketchPlan. Versions with only the Mapper enabled (⟨*SketchPlan*, *Uniform*⟩, ⟨*SketchPlan*, *Proportional*⟩) outperform versions with only the Profiler enabled (⟨*AnyFeasible*, *SketchPlan*⟩, ⟨*Greedy*, *SketchPlan*⟩), implying that gains from the Mapper are greater. In fact, versions with only the Profiler enabled sometimes perform even worse than strawman solutions (benefits < 1). Thus, the Mapper's systematic mapping of the coverage capability of sketches is crucial to providing benefits.

5 Other Related Work

Non-sketch intent-based telemetry: These frameworks [16, 25, 30] translate intents to data plane primitives. Unfortunately, most of these consider intents in isolation and support exact telemetry primitives that are prohibitively expensive.

Sketch error estimation: Multiple works [8, 22] estimate errors based on statistical properties of sketch counters. Their estimation logic can be integrated with SketchPlan's abstraction layer for dynamic configuration capabilities.

Dynamic query workloads: FlyMon [34] and DynaMap [26] allow dynamic workloads with sketches and map-reduce queries, respectively. These frameworks are orthogonal to, and can be integrated with, SketchPlan.

6 Limitations and Future Work

Before we conclude, we discuss some key limitations and directions for future work to realize the benefits of SketchPlan in practice.

Long-term robustness: While SketchPlan's error-resource profiles are robust to small-scale network changes, they may not be immune to longer changes in network traffic. Our future work includes extending SketchPlan to be robust to such long-term drifts. One direction is to build a model that can predict a sketch's accuracy based on properties of network traffic such as cardinality and entropy. Given this, one could actively measure incoming traffic (e.g., using sketches) and use this to predict the accuracy of other sketches in the data plane. If a sketch's accuracy falls below the operator's requirement, SketchPlan could re-generate error-resource profiles using the Profiler and re-run the Optimizer to generate a sketch deployment. Theoretical techniques [8] that model a sketch's error using its counters at runtime can also be a promising solution.

Dynamic reconfiguration: While SketchPlan's high-level interface allows the operator to easily specify new intents, SketchPlan only supports offline planning and sketch deployment to satisfy these intents. Future work includes runtime data plane reconfiguration to satisfy new intents. For e.g., given unusually high traffic volume, the operator may deploy new intents for DDoS detection. While the operator could re-run the Optimizer offline and re-deploy sketches, this could be prohibitively expensive for a large intent ensemble. Ideally, SketchPlan should support new intents at runtime. A possible solution is to run the Optimizer online incrementally for the new intents while considering the reduced data plane resources from the already-deployed sketches. While this may not be globally optimal, it will reduce latency and allow dynamic sketch deployment for new intents at runtime. Simultaneously, the Optimizer can be run in background for the entire set of intents to generate a globally optimal sketch deployment. Prior works [34] can be used to reduce downtime during data plane reconfiguration.

7 Conclusions

While sketches offer powerful capabilities, operators have been missing a high-level interface to use them efficiently with low effort, stymieing practical adoption. SketchPlan addresses this key missing piece by building an abstraction layer on sketches using an ensemble view, thereby reducing operator effort and improving efficiency in sketch-based telemetry.

Acknowledgments

We thank our anonymous shepherd and the reviewers for their suggestions; and Sanjeev Sridhar for his help in implementation. This work was funded in part by NSF awards CNS-2132639, CNS-2111751, CNS-2106214, CNS-2415758, and CNS-2431093; and was partially supported by Juniper Networks. The views contained in this article are those of the authors and not of the funding agencies.

References

[1] The CAIDA UCSD Anonymized Internet Traces - 2018. https://www.caida.org/catalog/datasets/passive_dataset, 2018.

[2] Gurobi Optimization: Python API Overview. https://www.gurobi.com/documentation/current/refman/py_python_api_overview.html, 2024.

[3] What is a network operations center (NOC)? https://www.ibm.com/topics/network-operations-center, 2024.

[4] Anup Agarwal, Zaoxing Liu, and Srinivasan Seshan. Heterosketch: Coordinating network-wide monitoring in hetero-geneous and dynamic networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 2022.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, page 63–74, New York, NY, USA, 2008. Association for Computing Machinery.

[6] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo C Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. In *ACM SIGCOMM*, 2017.

[7] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, 2002.

[8] 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.

[9] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. In *ACM SIGCOMM*, 2020.

[10] 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.

[11] Rui Ding, Shibo Yang, Xiang Chen, and Qun Huang. Bitsense: Universal and nearly zero-error optimization for sketch counters with compressive sensing. In *ACM SIGCOMM*, 2023.

[12] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[13] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, 2003.

[14] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *ACM IMC*, 2003.

[15] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, 2007.

[16] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, 2018.

[17] Abhishek Kumar, Minho Sung, Jun (Jim) Xu, and Jia Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *ACM SIGMETRICS*, 2004.

[18] Paolo Laffranchini, Luis Rodrigues, Marco Canini, and Balachander Krishnamurthy. Measurements as first-class artifacts. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 415–423, 2019.

[19] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19. Association for Computing Machinery, 2019.

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

[21] Zaoxing Liu, Hun Namkung, Anup Agarwal, Antonis Manousis, Peter Steenkiste, Srinivasan Seshan, and Vyas Sekar. Sketchy with a chance of adoption: Can sketch-based telemetry be ready for prime time? In *IEEE International Conference on Network Softwarization (NetSoft)*, 2021.

[22] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. SCREAM: Sketch resource allocation for software-defined measurement. In *ACM CoNEXT*, 2015.

[23] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. Sketchlib: Enabling efficient sketch-based monitoring on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[24] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. Sketchovsky: Enabling Ensembles of Sketches on Programmable Switches. In *USENIX NSDI*, 2023.

[25] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *ACM SIGCOMM*, 2017.

[26] Chaofan Shou, Rohan Bhatia, Arpit Gupta, Rob Harrison, Daniel Lokshtanov, and Walter Willinger. Query planning for robust and scalable hybrid network telemetry systems. *Proceedings of the ACM on Networking*, 2(CoNEXT1):1–27, 2024.

[27] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. FCM-sketch: generic network measurements with data plane support. In *ACM CoNEXT*, 2020.

[28] Haifeng Sun, Qun Huang, Jinbo Sun, Wei Wang, Jiaheng Li, Fuliang Li, Yungang Bao, Xin Yao, and Gong Zhang. Autosketch: Automatic sketch-oriented compiler for query-driven network telemetry. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, 2024.

[29] Kyu-Young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Trans. Database Syst.*, 15(2):208–229, jun 1990.

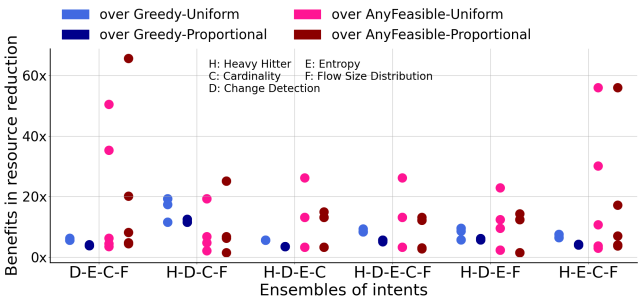
[30] Zhaowei Xi, Yu Zhou, Dai Zhang, Kai Gao, Chen Sun, Jiamin Cao, Yangyang Wang, Mingwei Xu, and Jianping Wu. Newton: Intent-driven network traffic monitoring. *IEEE/ACM Transactions on Networking*, 30(2):939–952, 2022.

[31] 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 *ACM SIGCOMM*, 2018.

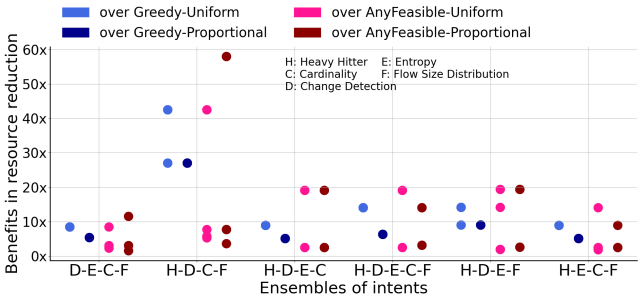
[32] Yinda Zhang, Peiqing Chen, and Zaoxing Liu. OctoSketch: Enabling Real-Time, continuous network monitoring over multiple cores. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1621–1639. USENIX Association, 2024.

[33] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. CocoSketch: high-performance sketch-based measurement over arbitrary partial key query. In *ACM SIGCOMM*, 2021.

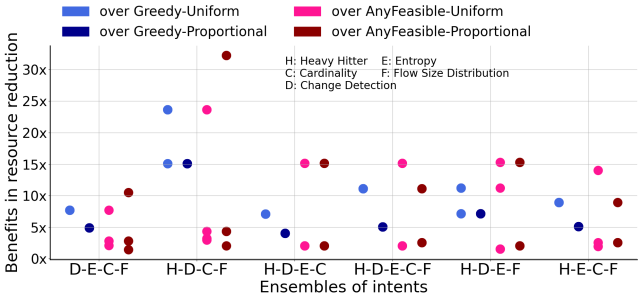
[34] Hao Zheng, Chen Tian, Tong Yang, Huiping Lin, Chang Liu, Zhaochen Zhang, Wanchun Dou, and Guihai Chen. FlyMon: Enabling on-the-fly task reconfiguration for network measurement. In *ACM SIGCOMM*, 2022.



(a) Loose targets



(b) Medium targets



(c) Strict targets

Figure 10: SketchPlan's benefit in resource reduction over strawman solutions for three progressively-stricter accuracy targets

A Ethics

This work does not raise any ethical issues.

B Network-wide results for other policies

	heavy hitter	entropy	cardinality	flow size distribution	change detection
Loose	7%	20%	10%	75%	10%
Medium	5%	15%	7%	75%	10%
Strict	5%	15%	7%	70%	7%

Table 5: Three progressively-stricter accuracy targets

Note that FSD errors are high due to its error metric, which matches flow sizes *exactly* between the true and estimated distributions. Computing the error on binned flow sizes yields much smaller errors.