SketchGuide: Reconfiguring Sketch-based Measurement on Programmable Switches

Zhengyan Zhou[†], Jingwen Lv[†], Lingfei Cheng[‡], Xiang Chen[†*], Tianzhu Zhang[◇],
Qun Huang[*], Jiayu Luo[‡], Longlong Zhu[‡], Dong Zhang[‡], Chunming Wu[†]
Zhejiang University[†], Fuzhou University[‡], Peking University[*], Nokia Bell Labs[◇]

***Abstract*—Sketches enable efficient and fine-grained network measurement results with configurable resource-performance trade-offs. While sketch configurations are guided by theories, the current theoretical guidelines are either impractical or deficient for sketch configurations on emerging programmable switches. To better configure sketches on programmable switches, we (1) systematically analyze the limitations of sketch configuration guidelines on programmable hardware switches (*i.e.,* unguided parameters, accuracy profiles, and resource budgets); (2) propose a generic and practical framework called *SketchGuide* to automate efficient sketch configurations on programmable switches; (3) implement SketchGuide on a Barefoot Tofino switch and compare SketchGuide to the state-of-the-art sketches by conducting extensive experiments. Our evaluations demonstrate that SketchGuide can automatically configure unguided parameters given resource budgets. SketchGuide reduces the hardware resource footprint by 52.92%-99.28% compared with current guidelines without impacting fidelity.**

I. Introduction

Network measurement provides indispensable information for modern network management tasks such as intrusion detection, load balancing, and traffic engineering [1–3]. In this respect, emerging programmable switches allow users to flexibly customize packet processing behaviors in the data plane for richer and real-time measurement capabilities.

With this network programmability, sketches or sketching algorithms are demonstrated as promising alternatives [4–17], given their efficient resource footprints and the fine-grained measurement results. Further, sketches provide configurable parameters, which are highly important because they determine the trade-offs between resources footprints and measurement accuracy. For instance, the count-min sketch (CM) [13] provides configurable counter numbers to trade flow frequency errors for memory footprints. Ideally, users configure these parameters with performance guarantees (*e.g., error upper bound*) according to some approaches. These approaches are given by designers to guide parameter choices backed by sound theories. We refer to these approaches as *theoretical configuration guidelines* in this paper.

Unfortunately, these theoretical guidelines are impractical and deficient in programmable switches. Specifically, users may have preferences for measurements. For example, in addition to network measurement, administrators may need to reserve a remarkable portion of switch resources to realize the network functions that prevent critical user traffic from being affected by network maintenance or failures [5]. In view of switch resource limitations [18, 19], administrators need to carefully balance fine-grained switch resources between measurement tasks and other functions [15]. However, the existing theoretical guidelines fail to give administrators instruction on how to configure switch-level resources to satisfy their preferences (e.g., preventing tasks from overusing the reserved portion of switch resources). More precisely, we analyze and identify *three key limitations* (§ III) in existing guidelines.

- *Unguided parameters:* To be hardware-compatible, some sketches are redesigned with new parameters that enable them to fit in resource-restrictive programmable switches [4–13, 15, 16]. However, none of the existing guidelines provide insights into how to configure these parameters. Thus, users have to rely on their empirical experience to configure, leading to sub-optimal implementations.
- *Accuracy profiles:* Existing guidelines only target a limited number of accuracy metrics and introduce their bounds under worst-case workloads [4–13, 15, 16]. However, administrators may have different measurement goals corresponding to a wide range of accuracy metrics [15, 20, 21]. Also, they desire to understand the accuracy under actual workloads, which highly diverge from worst-case workloads [6, 22]. In this context, existing guidelines cannot satisfy user-specified requirements.
- *Resource budgets:* Some hardware resources are more likely to be the bottleneck when implementing sketches with other functions [15]. For example, sketches need a large number of hash calls and SALUs for packet processing and counter updates respectively. Both resources are scarce in hardware and may need to be carefully budgeted to ensure the coexistence of switch functions. However, some guidelines only focus on high-level sketch parameters [4–13] so that users are unable to budget the bottleneck resources at a switch level. As a result, these bottleneck resources may exceed or exhaust the user-specified budgets, leading to configurations that are infeasible for implementation.

Consequently, current works take an *empirical* approach [5, 6, 11, 12, 15–17, 20] to configure sketches. Users must manually tune parameters while observing whether sketches conform to accuracy targets within budgets. However, the manual tuning approach entails manual efforts and even sub-optimal performance, hindering the efficiency of reconfiguring

978-1-6654-8234-9/22/$31.00 ©2022 IEEE

sketches for user preferences. SketchLearn [6] addressed similar but different problems, *i.e.,* the user burdens of sketch configurations. But it is limited in budgeting switch-level resources, internal parameter configurations, and comprehensive accuracy profile. Our experiments show that configuring the internal parameters can significantly reduce memory consumption with the same accuracy (§ VIII).

In this paper, we address these limitations and propose a generic framework called *SketchGuide* for user preferences to automatically configure sketches on programmable switches at the offline stage. Users can leverage SketchGuide to assign switch-level resource budget, configure all parameters, and profile preferred accuracy metrics. Specifically, *SketchGuide (1)* automatically chooses configurable parameters via primitive identifications, *(2)* explicitly expresses the trade-off between resources and performance by defining an expressive expression *Sketch Skyline (SS)*, *(3)* automatically solves optimized configurations for all parameters by posterior statistics and a pruning algorithm, *(4)* profiles accuracy under actual traffic workloads using machine learning. With this generic framework, users can automatically obtain efficient configurations given specified resource budgets with comprehensive accuracy profiles. Once the system is successfully built, users can easily reconfigure the sketches to meet new service-level objectives (SLOs). Furthermore, we *(5)* release the source code of SketchGuide so that it can help users automatically configure sketches for their preferences.

We conduct extensive experiments on a Tofino switch [23] to evaluate *SketchGuide* and reconfigure some state-of-the-art sketches [5, 6, 8, 13, 15, 17]. The evaluation shows that *SketchGuide* can generate configurations for optimized performance (*e.g.,* up to 99.28% resource footprint reduction) given resource budgets and accurately profiles accuracy targets.

II. Background

We start with some background on sketch-based measurements on programmable hardware switches. Next, we illustrate the configuration approaches of sketches, including theoretical guidelines and manual tuning approaches.

A. Programmable Hardware Switches

Hardware architectures. Programmable hardware switches allow users to customize the behaviors of data-plane packet processing. This paper focuses on Reconfigurable Match-Action Tables (RMT) paradigm [18] (*i.e.,* the architecture of a canonical commercial realization Intel Tofino switch chip [23]). RMT-based programmable switches maintain a pipeline of stages in the data plane for line-rate packet processing. Each stage is equipped with a match table that matches packet header fields to a specific value and executes simple instructions followed by action units. Each stage maintains an identical design with the same types and amounts of resources (*e.g.,* SRAM and SALU). A data-plane program is written in P4 language [24]. P4 compiler maps the programs into the pipeline stages and offers optimized resource allocations. A P4 program cannot be compiled if it requires excessive resources.

B. Sketch-based Measurement

Single-level/multi-level sketches. Sketch is an approximate data structure with some sketching algorithms for data streaming processing. Some single-level sketches such as CM [13] and CS [25] maintain a matrix of counters. As a canonical example, CM comprises d hash functions $h_i(1 \leq i \leq d)$ and a counter matrix $C[i][j]$, where $1 \leq i \leq w, 1 \leq j \leq d$). The $C[i][j]$ consists of d hash functions h_i each of which consists of w counters. Particularly, some sketches such as SketchLearn [6] and UnivMon [21] maintain *multi-level* counter matrices. For example, UnivMon uses multiple sketches (*e.g., L-level* sketches) and provides general monitoring capabilities.

Measurement tasks. The flow features of sketch-based measurements mainly include frequency [5, 6, 9, 13, 25], cardinality [5, 6, 13, 25], heavy hitters [5, 6, 11, 12, 16], heavy changers [5, 6, 12], entropy [5, 6, 17] and distributions [5, 6]. We illustrate two typical tasks due to the space limit. *(1)* For frequency estimations, a flow key f can be defined as any combination of packet fields (*e.g.,* 5-tuples or source-destination address pairs). Each flow key is first hashed by n hash functions to n hash value $h_i(f)$. The $h_i(f) - th$ counter at $i - th$ row is updated with the frequency v of f. Users can query the total frequency of f from the value of $h_i(f) - th$ counters at $i - th$ row. For example, CM regards the minimum of d values, $Min\ (C[i][h_i(f)])(i = 1, ..., d)$, as the frequency estimation of f. *(2)* The heavy-hitter detections report the flows whose flow frequency/size exceed a predefined threshold (*i.e.,* heavy hitter). For example, CM tracks heavy hitters by a priority queue or heaps but these data structures are infeasible or inefficient to implement in programmable switches [15].

Hardware optimizations. To make it feasible or efficient for sketch implementations, some optimizations for sketches at a hardware level are introduced in SketchLib [15]. The state-of-the-art solution (optimization#6 (O6) in SketchLib) maintains a *hash table* and an *exact table* to track these heavy flow keys. First, the heavy hitters are mapped into the hash table at the data plane. Second, the collided keys in the hash table will be reported to the control plane. The control plane appends the collided keys to the exact table. The keys stored in both tables will not be reported to reduce bandwidth consumption between both planes. Last, the heavy hitters are extracted from both tables and regarded as the final results.

C. Configuration Approaches

Theory guidelines. Sketch designers provide some guidelines for parameter choices backed by sound theories. For example, the theoretical guidelines of CM require users to assign two parameters, ϵ (*i.e.,* maximum acceptable error upper bound) and δ (*i.e.,* the probability of the bound failing), given resource budgets. Users obtain the width w and the depth d of the sketch according to $w = \lceil \frac{e}{\epsilon} \rceil$, $d = \lceil ln(\frac{1}{\delta}) \rceil$. The δ means that the error bound may fail. For example, users may expect the upper bound of flow size error is 0.001, $\epsilon = 0.001$, while the upper bound may fail with a probability 0.01, $\delta = 0.01$. Thus $w = \lceil \frac{e}{0.01} \rceil = 2719$ and $d = \lceil ln(\frac{1}{0.01}) \rceil = 5$. Next, users can write P4 code to implement CM with 5 register

TABLE I
SOME UNGUIDED/TUNABLE SKETCH PARAMETERS

Sketch Name	Parameters
SketchLearn [6]	θ: large flow extraction threshold
	ϕ: extraction threshold of the k-bit flow key
Sketches optimized by	h: hash table size for heavy hitters
SketchLib O6 [15]	e: exact table size for heavy hitters
FCM-Sketch [17]	k_f: k-ary tree branches
	t: the number of trees
	b: counter size of the tree
HashPipe [16]	d: the number of table stages
	k_h: desired heavy hitter number
UnivMon [21]	l: L_2-HH sketch level
	c: L_2-HH sketch colume
	r: L_2-HH sketch row

arrays, each of which maintains 2719 register units. Also, these parameters entail the consumption of more scarce resources, such as SALUs, hash calls, and stages.

Manual tuning. Users may manually tune the parameters for performance preferences. We summarize the reasons that users manually tune the parameters. The reasons may include *(1)* meeting specific performance requirements which may be out of the range of guidelines (*e.g.,* there are no error bounds median error should be less than 5% [15]), *(2)* better conforming to actual workloads (*e.g.,* traffic distribution) for resource-efficiency rather than reserving excessive resources for worst-case workloads [6], *(3)* directly assigning resource budgets without careful performance considerations [5].

III. MOTIVATION

Problem scope. In the context of programmable networks, the users can customize sketch-based measurements given specified user preferences of performances. For example, the users may require different service level agreements (SLAs) of measurements in different scenarios such as Clouds, Internet Service Providers (ISPs), and edge networks. Simultaneously, the users must ensure the coexistence between measurement tasks [7, 15, 26–28] and some resource-intensive tasks such as in-network caches [29]. However, the resources that a programmable switch is equipped with are scarce. For instance, a typical commercial programmable switch maintains 12 pipeline stages, each of which is equipped with 10 MB of SRAM and TCAM, 10 SALUs, and 10 hash calls [18, 19]. These require the users to carefully budget the switch-level resources for sketch-based measurements while meeting the SLAs. Specifically, users can configure the parameters of sketches to tradeoff resource usage and performances. In any of these scenarios, our goal is to configure sketch-based measurements on a programmable switch to minimize sketch resource consumption while fulfilling user preferences.

However, there are some limitations in the guidelines of configurations. Without loss of generality, we start with *heavy-hitter detections* using the typical CM optimized with O6 in SketchLib [15] to illustrate these limitations, which also exist in more tasks (*e.g.,* entropy, cardinality, etc.) and more sketches on programmable switches (*e.g.,* [5, 6, 8–12, 14, 17]).

A. Key Limitations

Limitation#1 (L1): unguided parameters. Existing theory guidelines fail to configure some parameters affecting the resource-accuracy tradeoffs in sketches. Specifically, *(1)* some unguided parameters are out of the range of the original design goals (*e.g.,* the hash/exact table slots for heavy hitters in SketchLib O6 [15]), and *(2)* some parameters are generated because the data structures of sketches are modified for efficiency or realizability purposes (*e.g.,* k in FCM-sketch [17], d in HashPipe [16]), and *(3)* the validity of some theories under the modifications on data structures may be a question. Theory guidelines do not tell how to configure these parameters. But these parameters should be satisfactorily configured because they determine the accuracy-resource tradeoff (demonstrated in § VIII). More examples are shown in Table I.

Limitation#2 (L2): accuracy profiles. A comprehensive profiler of accuracy gives users more confidence to deploy sketches. Some theory guidelines provide error bounds to profile measurement accuracy. However, the error bounds only address a part of measurement metrics (*e.g.,* flow frequency errors in CM). In practice, users may have some other performance targets (*e.g.,* the median error should be less than 5% [15]) and more measurement tasks [20, 21] (*e.g.,* entropy, cardinality estimation). The error bounds are unable to profile more accuracy metrics and more measurement tasks. Furthermore, the existing theory largely provides loose bounds (*e.g.,* ϵ and δ of CM) for only worst-case workloads. Some works show that the bounds are too loose to profile accuracy in practice [6, 22, 30]. As a result, the loose error bounds give less confidence in the measurement results so that users have to reserve more resources to guarantee accuracy [6, 22].

Limitation#3 (L3): resource budgets. Sketch is abstractly backed by the sound theory and is independent of implementation targets. So it may be unsurprising that programmable hardware resources are not taken into account in the theoretical guidelines. However, it is inevitable to carefully budget hardware resources for sketches as some scarce resources are likely to be the bottleneck. Unfortunately, users can only assign these parameters at a high level without the observability of the actual hardware resources (*e.g.,* SRAMs, SALUs, stages, etc.) before compilation. As a result, the assigned parameters may lead to the budget excess of resource allocation or a sub-optimal performance of the sketch-based measurement. Although users can attempt to compile the program to observe resource allocations, this approach is time-consuming (*e.g.,* up to 5 minutes per sketch compilation § VIII) and may lead to unnecessary resource consumption.

Summary. Theory guidelines are limited while some sketches even lack guidelines [15, 16]. Although manually tuning is an alternative but it may entail *(1)* labor-intensive efforts, *(2)* expert experience requirements, *(3)* sub-optimal performances, *(4)* comprehensive accuracy profiles, and *(5)* the implicit correlation between resource-performance tradeoffs and parameters. In summary, these limitations of theory guidelines (*L1, L2,* and *L3*) and limitations of manual tuning motivate us to better guide sketch configurations.

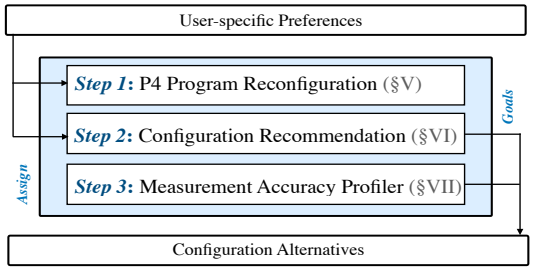

Fig. 1. The overview of configuring sketches by SketchGuide

IV. Design Goals and Challenges

Next, as is shown in Fig. 1, we illustrate our high-level ideas to address these limitations we identified earlier. Then, we highlight three key challenges that our design faces.

Design goals. Our main goal is to provide a generic framework *SketchGuide* to configure sketches given specified resource budgets. A three-step design is as followed. First, users assign switch-level resource budgets and performance targets based on preferences. Second, SketchGuide automatically configures all parameters and recommends optimized configurations, can explicitly express the tradeoff between resources and performance. Last, SketchGuide provides accurate and comprehensive accuracy profiles for each configuration.

A. Challenges and Key Insights

However, it is non-trivial to achieve these goals. In this section, we describe the main challenges we face.

C1: Generically parameter identifications and sketch configurations. It is challenging to *generically* identify sketch parameters given resource budgets. Intuitively, we can configure the original sketch parameters (*e.g.,* w and d of CM). However, the data structures/algorithms of sketches may be modified [15–17] due to the implementation differences. Thus, these original parameters may be modified or disappeared. Furthermore, it is challenging to know the resource usage and budget resources from the original parameters. Because the mapping from the original parameters to the resources usage is not straightforward (*e.g.,* the actual stage consumption can only be known by compiling the P4 program).

Insights. To be general, SketchGuide should configure parameters in a unified approach. Despite the various sketch implementations and implicit mapping, we observe that some primitives related to resources remain unchanged. Thus, rather than configuring the original parameters, we configure the amounts of resources that a sketch consumes. Specifically, we can find configurable parameters related to resources (*i.e., resource parameters*) by identifying these unchanged primitives. SketchGuide provides four types of primitives (shown in Table II) to identify resource parameters. With these primitives, users can perform code analysis to automatically identify resource parameters in a unified manner.

C2: Generically configurations for exclusive or heterogeneous targets. It is challenging to configure a sketch tradeoff between resources and performance targets. Because users may specify multiple mutually exclusive targets. For one, some performance metrics such as recall and precision are mutually exclusive. For another, some performance targets are inherently heterogeneous (*e.g.,* precision (indicating measurement accuracy) and SRAM footprints (indicating resource usage)). Thus, it is challenging to recommend the "best" configuration under multiple exclusive targets. Furthermore, it is also difficult for users to accurately measure their performance preferences among these exclusive targets.

TABLE II
Identify resource parameters with primitives

Types	Primitive	Descriptions
Register	`reg_count (1)`	The number of registers in an array
	`reg_init (2)`	The initialization of an array
	`reg_action (2)`	The action of an array
Stage	`nested_if (2)`	The number of nested if clauses
Hash	`hash_init (2)`	The initialization of a hash call
Table	`tbl_size (1)`	The size of an exact table

Approaches	Type	Descriptions
Identify params	(1)	Identifying the parameters in primitives
Count primitives	(2)	Counting the number of primitives

Insights. We tackle this challenge by defining an explicit trade-off expression, called *Sketch Skyline (SS)*, borrowed from the idea in database systems [31]. SS is a group of vectors of optimized performance targets that cannot dominate each other. "A vector a dominates another vector b" represents that all performance targets of a are superior to the ones of b respectively. For instance, for a vector [recall, precision], $a = [0.90, 0.80]$, $b = [0.79, 0.92]$, a has higher recall but has less precision than b. Thus a and b cannot dominate each other. With SS, we can explicitly express the tradeoffs of heterogeneous and potentially exclusive performance targets. Moreover, the SS that we recommend is a group of configuration vectors that target the user preferences rather than the "best" one. This allows users to inaccurately measure their preferences and find satisfactory configurations from SS.

C3: Accuracy profiling with large coverage of actual workloads. It is challenging to profile the accuracy of configurations at an offline stage with large coverage of actual workloads (*e.g.,* traffic distributions). We profile the accuracy at actual workloads for resource efficiency rather than the worst cases. A strawman solution is to inject the actual workloads captured from the real scenario into the configured sketches on real devices and profile their accuracy. But this approach is biased because not all workloads might be observable in practice. Although a uniformly distributed synthetic trace allows users to estimate worst-case workloads [32] but this approach is far from resource-efficiency.

Insights. Despite the insufficient coverage of real traces, we can first capture real trace and profile accuracy at a three-device linear topology that deploys sketches on a DUT (device under test). Next, we leverage the *generality* of machine learning to predict accuracy at other workloads. Specifically, we use a neural network to learn the mapping from the actual workloads. For simplicity, we express traffic distributions by some statistics. As a result, users can fast profile sketch performances covering various traffic distributions by feeding the trained neural network the statistics of traffic distributions.

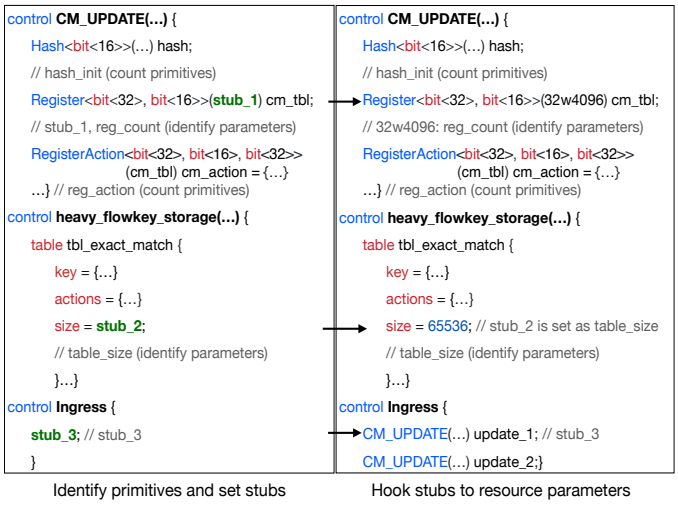

Fig. 2. Primitive identification (illustrated in the annotation) and resource parameter setting by replacing stubs with resource parameters.

B. Design overview

Next, we dive into the details of how SketchGuide addresses these challenges. Fig. 1 shows the workflows that users leverage SketchGuide for configurations. In step 1, users generate a reconfigurable P4 program where the parameters of sketches are resource-related and reconfigurable. In step 2, SketchGuide searches the optimized configurations with the reconfigurable program using a Bayesian Optimizer [33] to prune the search space. In step 3, users can profile the accuracy of these configurations and select the preferred one.

V. P4 Program Reconfiguration

Overview. Users make the P4 program reconfigurable so that SketchGuide can automate the reconfiguration process and search optimized configurations. Specifically, a reconfigurable P4 program is a template file with some stubs. The program can be reconfigured by replacing the stubs with P4 codes. To generate the reconfigurable program, users first write a P4 program to be deployed. Then, they identify the parameters related to resources (*i.e., resource parameters*) in the program, where the positions of these resource parameters are set in some stubs, to generate the reconfigurable program.

Resource types. Users identify the parameters related to resources. To answer what kinds of resources should be considered, we observe and carefully synthesize the known resources that sketches require [14, 15]. Despite the diversified sketches, the required hardware resources are similar. We summarize the main resource types, which can be further extended to more resources that maintain the same consumption (*e.g.,* Hash call can be extended to hash bit and hash dist unit).

- *SRAM* Both counters implemented as register arrays and the table to track flow keys consume SRAM blocks.
- *SALU* A table equipped with register arrays being read/written or maintained hash calls consumes a SALU.
- *Hash call* The flow keys hashed to sketches consume hash call resources, which should be pre-allocated to guarantee all possible execution paths for line rate processing [15].
- *Stage count* A sketch with multiple condition dependencies or the total required resource usage exceeding the equipped resources per stage consume multiple stages.

Resource parameter identification. Next, SketchGuide chooses resource parameters by primitive identifications. To address C1, SketchGuide identifies resource parameters that indicate the tunable resource usage that a sketch consumes. The resource usage can be tuned by changing P4 codes, which are abstracted as *primitives*. The primitives guide users to choose the resource parameters they intend to configure. It is general to identify parameters based on these primitives because P4 codes are always corresponding to specific resources regardless of sketch implementations. We define four types of primitives (shown in Table. 2) with two approaches of resource parameter identification shown in Fig. 2.

(1) Identifying the parameters *embedded* in the primitives, *e.g., reg_count*, the number of registers in an array):

```
Register<bit<32>, bit<16>>(4096) cm_tbl
Resource parameter: 4096
```

(2) Counting the number of primitives, *e.g., reg_init*, counting the number of register initialization:

```
CM_UPDATE(...) update_1;
CM_UPDATE(...) update_2;
Resource parameter: num(CM_UPDATE)=2
```

Note that sketches should be implemented in proprietary tables to ensure no identification errors. Users choose these parameters at the beginning only once.

Reconfiguring a P4 program. With these identified parameters, we obtain a P4 template file by setting stubs at the positions of these parameters, from `stub_1` to `stub_n`. For approach *(1)*, the parameters embedded in a primitive are replaced by a stub, *e.g.*:

```
Primitive:
Register<bit<32>, bit<16>>(4096) cm_tbl;
Stub:     stub_1;
```

The primitives identified using approach *(2)* are substituted by a stub *e.g.*:

```
Primitive:
CM_UPDATE(...) update_1;
CM_UPDATE(...) update_2;
Stub:     stub_2;
```

To obtain a compilable P4 program, SketchGuide replaces stubs in the P4 template file with P4 codes comprising resource parameters. As an example shown in Fig. 2, `stub_1` is replaced by a resource parameter `reg_size`, the size of a register array. The `stub_2` is replaced by another one, `exact_tbl_size`, the size of an exact table. Particularly, some primitives such as `reg_init` and `reg_action` can be combined into a code block (*e.g.,* `CM_UPDATE`) and collectively replaced by one stub `stub_3`. After replacing all the stubs, the final P4 program is generated.

VI. Configuration Recommendations

Overview. After reconfiguring the P4 program, SketchGuide searches optimized configurations using Bayesian Optimizer to prune the search space. Users assign resource budgets and objective functions for the Bayesian

Optimizer. To tackle C2 (§ IV-B), we first define an explicit expression *Sketch Skyline (SS)* to eliminate the conflicts among performance targets. Next, we design an algorithm to fast generate fewer *SS candidates* by exploiting the properties of SS and pruning the non-SS.

A. Preference Assignment

Resource budgets. Users can assign resource budgets at a switch level for sketches as some ranges (*e.g.,* the number of SALU $\in [1, 6]$). The configuration that is out of the budget will raise exceptions, which are handled by reducing resource usage in the priority of each target. SketchGuide further filters the configurations out of the budgets (L3 addressed).

Objective functions. Users can assign the weight of each performance target based on preferences and construct a monotone scoring function $f(x_1, ..., x_n)$ (*e.g.,* $f(x_1, ..., x_n)$= *0.3×recall+0.3×precision+0.3×SRAM_count*). Bayesian Optimizer maximizes the f to search for the optimal resource parameters $(p_1, ..., p_n)$. To normalize each performance target, we observe that some resources monotonically increase with the increase of parameters. Thus, we normalize the heterogeneous performance vectors by dividing the maximum of each performance value during the search process.

B. Sketch Skyline Definition

We define Sketch Skyline (SS) to express the performance targets. Formally, SS is a group of vectors v_k with performance targets, x_i. A vector is denoted as $v_k = [x_1, ..., x_n]$.

Definition 1. *v_1 dominates v_2. For two bounded vectors, $v_1 = [x_1, ..., x_n]$ and $v_2 = [y_1, ..., y_n]$, $i \in [1, n]$, x_i is better than y_i.*

Definition 2. *Sketch Skyline (SS) is a set of v_k that cannot dominate each other.*

By defining *SS*, we explicitly express the tradeoff of heterogeneous performance targets and eliminate potential conflicts. For example, for two heterogeneous performance targets recall and SRAM counts, "better" represents higher recall and fewer SRAM counts. At this point, an SS can be $SS = \{v_1, v_2\}$, where $v_1 = [0.78, 17]$, $v_2 = [0.86, 20]$. As a result, SketchGuide can provide more comprehensive and understandable configuration expressions for users.

Lemma 1. *For a set M of any monotone scoring function f, $M \rightarrow R$, if a vector $v_k = [x_1, ..., x_n]$, where $v_k \in M$ maximizes $f(x_1, ..., x_n)$, then v_k is in the SS.*

Lemma 1 indicates that the favorite configuration of users is included in the *SS* no matter how users weigh preferences.

Lemma 2. *For every vector $v_k = [x_1, ..., x_n] \in SS$, there exists a monotone scoring function $f(x_1, ..., x_n)$ such that v_k maximizes $f(x_1, ..., x_n)$.*

Lemma 2 indicates that every vector in *SS* potentially is the favorite of users. The proof of both lemmas can be referred to [31, 34, 35]. We can leverage *Lemma 1* and *Lemma 2* to optimize the SS solving process in the next section.

C. SS Candiadate Generation

SS Candidates. Although users can specify an objective function, it is difficult to precisely express their preferences using the weight. Thus, rather than selecting the configurations with the maximal scoring function, SketchGuide takes the configurations generated in the searching process as candidates.

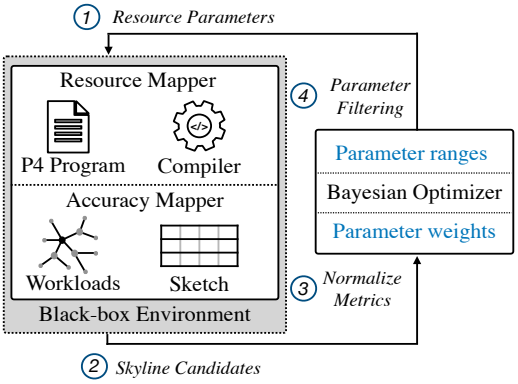

Fig. 3. The workflows of sketch skyline candidate solving

Bayesian optimizer. There are two challenges for SS candidate generation. (1) How to generate candidates fast but close to user preferences? (2) As a generic framework, SketchGuide must be scalable when the number of parameters is large (*e.g.,* users may specify $\leq$20 parameters). It is notoriously difficult to solve a high-dimensional problem promptly. We answer these questions using a Bayesian Optimizer [33]. Intuitively, Bayesian Optimizer (1) accelerates the candidate searching with an objective function; (2) approximates the function of the black-box environment by a posterior distribution of functions $d(p_1, ..., p_n)$ drawn from a Gaussian process prior. We illustrate the theory in the appendix.

Candidate searching. SketchGuide searches SS candidates using Bayesian Optimizer while mapping accuracy results and resource usage to accelerate the searching process. Fig. 3 shows that SketchGuide solves SS candidates in a four-step loop. *(1)* A black-box execution environment executes a reconfigurable P4 program where unguided parameters are combined into the unified resource parameters (L1 addressed). The execution environment hides the environment details and decouples the program into two mappers. *(2)* The black box includes a resource mapper and an accuracy mapper, both of which map parameters to resource usage and measurement accuracy respectively. The results are combined as a vector v_k (SS candidates). *(3)* SketchGuide generates the f from v_k, which is normalized to a range [0, 1]. *(4)* Bayesian Optimizer [33] searches for an SS candidate with a higher value of f. The configurations excessed the resource budgets are filtered. Next, we illustrate the details of this process.

Performance mapper. The resource usage and measurement results can be generated from the black-box executable environment. Users can trace the workloads (*i.e., traffic trace*) from the real world to obtain the measurement results under the actual workloads using an accuracy mapper. To obtain resource usage, a strawman solution is directly running sketches on the hardware. However, this solution hinders users from fast generating SS candidates and deploying sketches. For example, compiling a sketch (*e.g.,* SketchLearn in [15]) on a Tofino switch spends more than five minutes. Rather, SketchGuide maps light-weight sketches (*e.g.,* CM [13] and CS [25]) using the compiler equipped with Tofino switches

Algorithm 1 SS Solving with the nearest neighbor

Input: n: the nearest neighbor, sc: SS candidate, l: sc list, sf: scoring function
function NEAREST_NEIGHBOR_PRUNING(l)
  while l is not empty **do**
    find the n with the maximal sf
    append the n to l
    for each sc dominated by n in l **do**
      remove sc from l
    end for
  end while
  Return SS
end function

[23] due to the acceptable compiling duration (*e.g.,* less than one minute). For some complex sketches such as SketchLearn, SketchGuide maps resources using the RMT_mapper [15, 26], which is an open-source compiler mapping P4 programs to the RMT hardware. The RMT_mapper quickly exhibits similar results compared to Tofino compilers (shown in Fig. 16) with fidelity (demonstrated in [15]). We extend the RMT_mapper to map more resources such as SRAM and TCAM, keeping the same *objective* that minimizes pipeline stages as [15].

D. Sketch Skyline Solving.

To solve SS from its candidates, a strawman solution is to search all the possible configurations and leverage the classical block-nested-loops algorithm [31]. However, this brute-force approach is time-consuming. Instead, we automatically and fast solve the SS by leveraging the properties.

Property exploitation. The Bayesian Optimizer generates v_k minimizing f. However, it is challenging for users to express their expectations with an f. According to *Lemma 1*, v_k must be in the SS. Thus the weights of each performance target need not be accurately set. Because a maximum f must be in SS, which potentially is the favorite of users (*Lemma 1*). Furthermore, Bayesian Optimizer helps to fast search SS candidates with a direction (*i.e.,* minimizing the f). As a result, it is unnecessary to explore all possible candidates such that fewer vectors will be counted as SS candidates.

SS Solving algorithm. To recommend configurations, SketchGuide first sorts the SS candidates according to f value. Second, we design an algorithm to solve the final SS by leveraging *Lemma 1*. In algorithm 1, SketchGuide *(1)* finds the nearest neighbor (NN) (*i.e., the minimum value of* $f(x_1, ..., x_n)$ as a vector v_k stored in the SS), *(2)* removes all the SS candidates which are dominated by v_k, *(3)* loops until all the SS candidates are dominated each other. Users can balance performance targets and choose configurations from the SS based on their preferences. Because the vectors in SS potentially are the favorite of users (*Lemma 2*). We illustrate the intuition of this algorithm in the appendix.

VII. Measurement Accuracy Profiler

To understand the robustness of each configuration in SS, SketchGuide tests the configured sketches on a real device.

TABLE III
Performance metrics and weights

Metric	Weight	Expectation	Description
Recall	1.0	[0.8, 1]	The recall of detection results
Precision	1.0	[0.8, 1]	The precision of detection results
SALU	-1.0	[1, 6]	The SALU block consumption
Map_RAM	-1.0	[1, 20]	The map RAM consumption
Hash_call	-1.0	[1, 6]	The hash call consumption
SRAM	-1.0	[1, 50]	The SRAM block consumption

However, these configurations are generated from sampled actual workloads that may not represent all the situations. Rather than exhaustively sampling all the workloads, we profile the accuracy of each configuration in the SS and leverage the generalization of machine learning to predict the accuracy under more workloads. Specifically, SketchGuide uses a neural network as a profiler to learn the performances. Then, SketchGuide can query the performance with high coverage from the profiler without testing the DUT because the neural network is driven by data such that it hides the details of sketch models (L2 addressed).

Accuracy result generation. The configured sketches are deployed on a real programmable switch (*i.e.,* DUT) to generate accuracy results. Users can build a three-device testbed with the DUT. A sender extracts traffic traces and sends packets to the receiver across the DUT, which measures the traffic. We take both accuracy results and resource parameters under various workloads as the dataset for the neural network.

Accuracy predictor. To achieve high coverage of accuracy prediction, we leverage the generalization ability of a neural network. Given the configurations, the accuracy of sketches is fixed under the same workloads. Thus, we can predict the accuracy of sketches with a regression task. We take some traffic statistics as the input of the neural network to simplify workloads, typically including skewness in Zipf distribution [36], the standard deviation of frequency, the mean of frequency, and the total frequency. The profiler is trained using the dataset captured from the DUT. The trained profiler can quickly query the performance under various statistics. Further, users can predict more accuracy metrics such as recall and precision rather than only errors. By assigning different statistics, users can query the accuracy under more workloads.

VIII. Implementation and Evaluation

We implement and evaluate SketchGuide to indicate that:

- SketchGuide can configure resource parameters and solve the final SS (Exp#1) given resource budgets.
- SketchGuide profiles accuracy under high coverage of workloads with acceptable prediction errors (Exp#2).
- SketchGuide automatically solves the SS that reduces the resource consumption under the same performance targets compared to theoretical guidelines (Exp#3, #4).
- SketchGuide generates configurations reducing the resource consumption under the same performance targets compared to the manual tuning approaches (Exp#5, #6).
- SketchGuide fast converges to the maximum (Exp#7) and eliminates unbounded configurations (Exp#8).

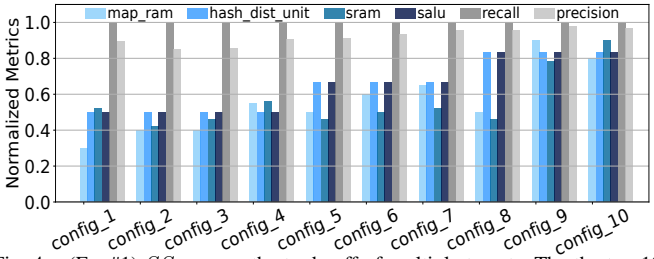

Fig. 4. (Exp#1) SS express the trade-off of multiple targets. The the top-10 scores in the SS is exhibited for users to choose preferred configurations.

- SketchGuide significantly reduces the time of resource mapping using the extended RMT_mapper compared to the Tofino compilers (Exp#9).

We illustrate Exp#8 and Exp#9 in the appendix.

A. Case Study

TABLE IV
IDENTIFIED PARAMETERS OF A CM WITH RANGES

Parameters	Range	Description
hash_cnt	[1, 7]	The counts of hash calls
reg_size	[512, 20480]	The number of units per register array
exact_tbl_size	[16, 65536]	The number of exact table entries
hash_tbl_size	[16, 65536]	The number of hash table entries

We elaborate on a case to study how to configure resource parameters and profile accuracy using SketchGuide. Another case study of a Bloom Filter configuration is illustrated in appendix. envision that an operator configures a CM [13] optimized by a SketchLib O6 for heavy-hitter detections [15] with some performance targets and SLAs. The unguided parameters are converted into resource parameters despite the diversity of CM implementations. Moreover, the unguided parameters can be configured to improve the performance.

Implementation. We implement SketchGuide in python 3.5 on Open Networking Linux (ONL) atop a Tofino switch [23]. We use a Tofino compiler as the resource mapper and a trace-driven simulation as the accuracy mapper. A real-world trace from CAIDA 20160121 Chicago [37] is used. The measurement results are captured from the PCIe channels at the ONL. The network flow is defined as a source IP address. The flows whose traffic volume exceeds a threshold in an epoch are taken as heavy hitters. An epoch is approximated as 10 thousand IPv4 packets. The threshold is set corresponding to the top 0.2 percentile of flow volume. Bayesian Optimizer is implemented using an open-source framework [38]. We implement the profiler using Keras [39]. The neural network comprises four layers, including 64, 128, 50, and 10 Relu neurons from the input layer, hidden layer, and output layer respectively. Because the dimension of accuracy prediction task is relatively small, we choose a small-scale neural network as the profiler. Generically, a simple network exhibits more robustness and generality. We set the batch size as 1500 and use dropout to prevent overfitting. We test 3216 groups of accuracy results measured by multiple accuracy metrics.

Methodology. We identify four key resource parameters using the primitives listed in Table II. Users can set slightly loose parameter ranges and resource budgets according to their requirements. The identified parameters and the assigned ranges are shown in Table IV. Next, we set stubs in a template file and generate P4 files by replacing the stubs with P4 codes. Here, we set the resource budgets and performance expectations, both of which are listed in Table III.

Exp#1: Configuration recommendations. SketchGuide can configure sketches with multiple performance targets specified by users. To show the scalability of SketchGuide, we target 6 metrics that are shown in Table III. These metrics and the corresponding weights are composed of the scoring function f. For a higher value of f, we set the weights of accuracy metrics and resource metrics as positive and negative numbers respectively. For simplicity (benefit from *Lemma 1*), we set the weights as 1.0 or -1.0 (*i.e.,* f=$recall$+$precision$-$SALU$-Map_RAM-$Hash\_call$-$SRAM$). The Bayesian Optimizer searches for the group of parameters with the maximum f. In this process, 400 SS candidates are generated (200 steps of random exploration and 200 steps of Bayesian Optimization). We filtered the configurations out of resource budgets (Table III, expectation) and solve the final SS by algorithm 1. The metrics are normalized by dividing the upper bounds of the expectations. The configurations with the top-10 highest target scores are selected. Fig. 4 shows these configurations are all optimal because their performances are unable to dominate each other. Thus, SS explicitly expresses the tradeoffs of each performance target. However, under the same objective function, the performance results may be various. Thus, we recommend all the SS to users to choose their preferred configurations based on requirements or SLAs. For example, a firewall function will be deployed in the device and consumes a number of memory footprints. Meanwhile, the SLA requires a measurement precision of no less than 80%. In this context, the user can choose config_1 due to the high precision and minimal consumption of memory footprints.

Exp#2: Accuracy profiler. To profile the accuracy of each configuration in SS, we test each configuration and generate 3216 groups of data. We predict the accuracy of each configuration using a neural network to better understand the accuracy under actual workloads at the offline stage. The workloads of traffic volume are expressed by skewness, standard deviation, mean, and total value. We leverage 5-fold cross-validation [40] to train the neural network and train 200 rounds in each fold. The profiler has a high training speed because 1500 data/batch are trained in parallel with 200 rounds of training per fold. Totally, the training time is less than one minute. We profile 4 accuracy metrics, *i.e.*, estimation errors of the flow traffic volume, recall, precision, and f1-scores. We test the profiler in the test dataset. Fig. 5 shows the profiler can predict these accuracy metrics with acceptable errors (less than 0.2 relative error of prediction in more than 80% dataset). The accuracy profiler can profile multiple accuracy targets with acceptable errors. The users can evaluate the performance and determine whether to deploy the configuration in practice.

B. Compared to the Theoretical Guidelines

We demonstrate that SketchGuide can *(1)* configure unguided parameters, *(2)* configure specified performance targets, and *(3)* reduce resource usage compared to theory.

Alternatives We conduct the flow size estimation task, where the most sketch-based measurement approaches provide bounded errors. First, the alternative sketches are configured according to their theoretical guidelines. The sketches include the typical CM [13], ElasticSketch[12] and the state-of-the-art FCM-Sketch [17], SketchLearn [6], and CocoSketch [8].

Methodology We set the same memory usage and parameters for CocoSketch and SketchLearn as the original papers. The remaining sketches follow the same configurations in FCM [17]. Their error bounds are derived from each configuration respectively. We maintain 1.5MB for CM, 1.5MB for Elastic, 1.31MB for FCM-Sketch, 500KB for CocoSketch, and 2.02MB for SketchLearn. Second, we solve configurations by SketchGuide under the same error bounds of theoretical guidelines. *(1)* We also configure the unguided/tunable parameters list in Table I. *(2)* We specify the performance targets as minimizing memory usage while eliminating the configurations with unbounded errors. *(3)* We show that the configurations solved by SketchGuide significantly reduce resource usage at the actual workloads compared to the theory.

Setup and metrics A trace from CAIDA 20160406 Chicago [37] with 2.2GB traffic is input. We show the memory reduction as a memory reduction rate (*i.e.,* the portion of memory reduction in the memory usage of original configurations) to measure memory for each sketch independently. We show the memory reduction rate as the minimum, lower quantile, upper quantile, and maximum from light to deep blue.

Exp#3: *SketchGuide configures unguided/tunable parameters and reduces resource usage.* Fig. 6 shows the memory usage of all sketches is significantly reduced. Because the unguided/tunable parameters affect the performances of sketches but theoretical guidelines are unable to configure. SketchGuide can configure because all the parameters are equivalently regarded as the inputs of the black-box environment. Bayesian Optimizer solves the configurations that maximize performance targets. The memory reduction rates are different due to the different degrees of how parameters influence performances. Although the memory usage under the actual workloads will be less than the upper bounds, SketchGuide can automatically solve the configurations without manual tuning and achieve optimized performance.

Exp#4: *SketchGuide configures parameters under the specified performance target.* We append the bandwidth target to show the scalability of SketchGuide under multiple user preferences. The performance target is minimizing memory while maximizing packet processing bandwidth. Fig. 7 shows the memory reduction rates are different from the ones in Fig. 6 due to different performance targets. Fig. 8 shows the configurations are optimized for the bandwidth target and the packet processing durations are significantly reduced. With the objective function, users can customize configurations to their preferences. Furthermore, the performance target is not limited to accuracy and memory. Other targets such as bandwidth can also be taken into consideration.

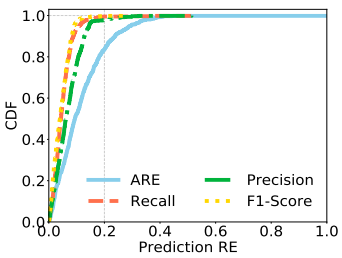

Fig. 5. (Exp#2) SketchGuide predicts multiple accuracy metrics with acceptable relative errors.

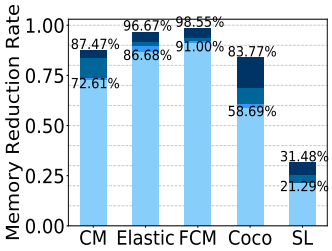

Fig. 6. (Exp#3) SketchGuide reduces memory usage compared with theoretical configuration guidelines.

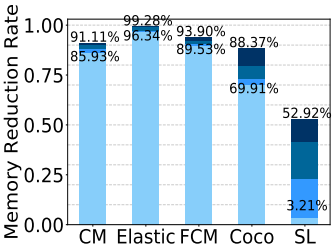

Fig. 7. (Exp#4) Memory usage compared with theoretical guidelines when minimizing update time.

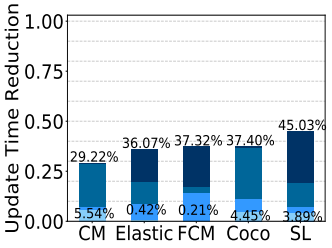

Fig. 8. (Exp#4) Update time reduction compared with theoretical guidelines when minimizing update time.

C. Compared to the Manual Tuning Approaches

We show that SketchGuide can reduce resource usage under the same targets compared to the manual tuning approaches.

Alternatives. The first manual tuning approach (M#1) configures sketches under a specified target. We follow the same methodology in [15] to configure CS [25] targeting 5% absolute relative middle errors while minimizing resource usage. We choose the minimal resource use with an accuracy of under 5% median error. Then, we compare another typical approach (M#2) [6, 8, 13, 17, 21] engaging all the resource budgets to sketches by maximizing parameters about resource usage. For example, a sketch can leverage all the memory for measurements and configure parameters by manually tuning.

Setup and metrics. The optimal configuration in [15] of CS ($w = 3$ and $d = 20000$) is taken as the baseline. We set the same target and configure CS using SketchGuide with 100 rounds of Bayesian Optimization. We measure the accuracy of the flow size estimation task by average relative error (ARE), $ARE = \frac{1}{n}\sum_{i=1}^{n}\frac{|f_i - \hat{f}_i|}{f_i}$, where n is the number of flow, f_i, and $\hat{f}_i$ are the actual and estimated flow sizes respectively.

Exp#5: *SketchGuide reduces resource consumption compared to M#1.* Fig. 9 shows that the configurations generated by SketchGuide consume fewer resources than M#1. Because users only finely tune the parameters and choose a configuration with acceptable performance. SketchGuide can further optimize the configuration due to the search directions. Moreover, SketchGuide can fast configure CS for better configurations. After iterations of 40 rounds, SketchGuide converges to the minimum because the Bayesian Optimizer automatically searches the configurations with directions.

Exp#6: *SketchGuide improves accuracy compared with M#2.* Given d from 1 to 10, we solve the configurations using SketchGuide. Fig. 10 shows that the ARE of each sketch is reduced because engaging all the memory budgets

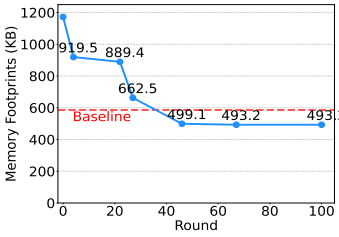

Fig. 9. (Exp#5) SketchGuide reduces memory usage of CS compared with approach M#1.

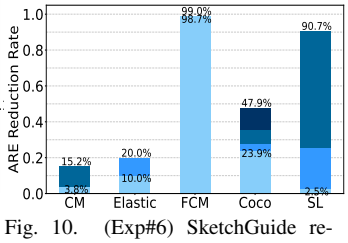

Fig. 10. (Exp#6) SketchGuide reduces ARE of each sketch compared with approach M#2.

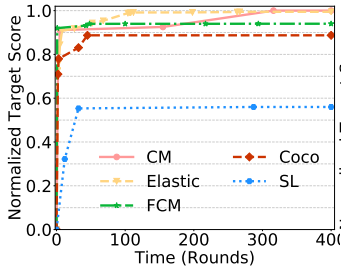

Fig. 11. (Exp#7) SketchGuide fast converges to the global maximum.

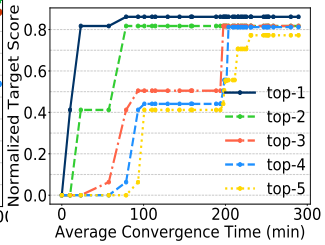

Fig. 12. (Exp#7) SketchGuide achieves the convergence point fast.

is inefficient. The accuracy can be significantly improved by SketchGuide under the same memory budgets. Because the configurations can be further optimized using the M#2.

Exp#7: *SketchGuide fast converges to the final SS.* Maintaining the same setup in Exp#1, we record the maximum normalized score during SS searching. Fig. 11 shows that it takes less than 230 minutes for SketchGuide to converge to the global maximum. We further test CM [13], FCM [17], SketchLearn [6], Elastic [5], and Coco [8]. Fig. 12 shows that it takes less than 230 minutes to converge. It means that SketchGuide searches for about 300 parameter combinations and finds the final SS without manual effort. A brute-force search may take unacceptable 2^{19} parameter combinations ($\sim$3932 hours) for CM ($d \in [1, 8]$ and $w \in [1, 65536]$).

IX. Discussion

Although SketchGuide is designed for the sketch configuration on programmable hardware switches, the same idea can be applied to software switches. One of the most important performance bottlenecks of sketches in software switches is the speed of packet processing [4, 7] rather than the memory footprints. SketchGuide can still optimize the speed of packet processing by customizing the objective function. The key difference from the sketches on hardware switches is that users can directly configure the original sketch parameters using SketchGuide without the hardware constraints. Furthermore, resource usage can be easily calculated from the parameters. In the context of software switches, the SketchGuide is motivated by the packet processing speed and will be more lightweight.

X. Related Works

A. Sketch Configuration

SketchLearn [6] aims to relieve the user burdens for sketch deployment. But it keeps some empirical parameters and inefficiently engages in all resources to configure. This approach may lead to sub-optimal performance. SketchGuide can configure all the parameters and provide optimized performance.SketchLib [15] proposes six techniques to reduce the resource consumption of the existing sketch implementation. SketchGuide can reconfigure the parameters of SketchLib for user preferences and further reduce resource consumption. SCREAM [41] dynamically allocates resources at runtime. ErrorEstimator [22] narrows the error bound of sketches on the fly. SketchGuide configures parameters before the deployment without complicated runtime scheduling. HeteroSketch [32] addresses the network-wide deployment of sketches on heterogeneous devices by manually tuning, which is labor-intensive and may lead to sub-optimal performance. UnivMon [21] configures sketches in the network-wide measurement. SketchGuide is orthogonal to these network-wide measurements and more fine-grained at a switch level. These network-wide works can also get the benefits of SketchGuide by assigning the scoring function. Some works use integer linear programming (ILP) for sketch configurations [20, 21, 32] and P4 program configurations [42] that may include sketches. But ILP needs the art of problem encoding such that the sketch performances depend on users. SketchGuide addresses the limitations of sketch configuration guidelines while providing a generic framework that can budget resources for multiple performance metrics without problem coding. Furthermore, SketchGuide only configures sketches without modifying them so that the theoretical guarantees still hold.

B. Parameter Combination Optimization

Some approaches can potentially be used to optimize sketch configurations. Grid search [43] exhaustively searches parameters but may lead to an intolerant searching time when the parameter ranges are large. Random search [44] solves the parameters by sampling but the performance is unstable. Particle swarm optimization (PSO) [45] searches for solutions with direction but is time-consuming. SketchGuide uses Bayesian Optimization [33] to fast search parameters with directions.

XI. Conclusion and Future Works

In this paper, we address three limitations of sketch configuration guidelines and propose SketchGuide, a generic framework, to automatically configure sketches for user preferences. SketchGuide recommends configurations (sketch skylines) given resource budgets and profiles the accuracy at actual workloads. Our evaluation results show that SketchGuide outperforms the current approaches of sketch configuration with orders-of-magnitude resource reduction and high configuration speed. We plan to configure more types of measurement tasks on more devices (*e.g.,* software switches) in our future works. The source code of SketchGuide is available at: https://github.com/vancasola/SketchGuide.

Acknowledgment

We thank our shepherd Prof. Chen Qian and the anonymous reviewers for their enlightening comments. Chunming Wu is the corresponding author. This work is supported by the "Pioneer" and "Leading Goose" R&D program of Zhejiang (2022C01243, the key R&D program of Zhejiang province (2021C01036)), and the Fundamental Research Funds for the Central Universities (Zhejiang University NGICS Platform).

REFERENCES

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proc. of ACM CoNext*, 2011.

[2] 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.

[3] 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.

[4] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proc. of ACM SIGCOMM*, 2017.

[5] 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.

[6] Q. Huang, P. P. Lee, and Y. Bao, "Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference," in *Proc. of ACM SIGCOMM*, 2018.

[7] 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.

[8] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "Cocosketch: high-performance sketch-based measurement over arbitrary partial key query," in *Proc. of ACM SIGCOMM*, 2021.

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

[10] L. Tang, Q. Huang, and P. P. Lee, "Spreadsketch: Toward invertible and network-wide detection of superspreaders," in *Proc. of INFOCOM'20*.

[11] ——, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *Proc. of IEEE INFOCOM*, 2019.

[12] Q. Huang and P. P. Lee, "Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams," in *Proc. of IEEE INFOCOM*, 2014.

[13] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, 2005.

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

[15] H. Namkung, Z. Liu, D. Kim, V. Sekar, P. Steenkiste, G. Liu, A. Li, C. Canel, A. A. Philip, R. Ware *et al.*, "Sketchlib: Enabling efficient sketch-based monitoring on programmable switches," in *Proc. of USENIX NSDI*, 2022.

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

[17] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "Fcm-sketch: generic network measurements with data plane support," in *Proc. of ACM CoNext*, 2020.

[18] 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," *ACM SIGCOMM CCR*, 2013.

[19] 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.

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

[21] 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.

[22] P. Chen, Y. Wu, T. Yang, J. Jiang, and Z. Liu, "Precise error estimation for sketch-based flow measurement," in *Proc. of ACM IMC*, 2021.

[23] (2022) Barefoot tofino. [Online]. Available: https://barefootnetworks.com/products/brief-tofino/

[24] 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.

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

[26] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. of USENIX NSDI*, 2015.

[27] X. Chen, Q. Huang, P. Wang, Z. Meng, H. Liu, Y. Chen, D. Zhang, H. Zhou, B. Zhou, and C. Wu, "Lightnf: Simplifying network function offloading in programmable networks," in *Proc. of IEEE IWQoS*, 2021.

[28] X. Chen, H. Liu, Q. Huang, P. Wang, D. Zhang, H. Zhou, and C. Wu, "Speed: Resource-efficient and high-performance deployment for data plane programs," in *Proc. of IEEE ICNP'20*.

[29] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proc. of USENIX NSDI*, 2017.

[30] Q. Huang, S. Sheng, X. Chen, Y. Bao, R. Zhang, Y. Xu, and G. Zhang, "Toward {Nearly-Zero-Error} sketching via compressive sensing," in *Proc. of USENIX NSDI*, 2021.

[31] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. of ICDE*, 2001.

[32] "HeteroSketch: Coordinating network-wide monitoring in heterogeneous and dynamic networks," in *Proc. of USENIX NSDI'22*.

[33] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[34] H.-T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM (JACM)*, 1975.

[35] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[36] D. M. Powers, "Applications and explanations of zipf's law," in *New methods in language processing and computational natural language learning*, 1998.

[37] (2022) The caida anonymized internet traces. [Online]. Available: http://www.caida.org/data/overview/

[38] (2022) Bayesianoptimization. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[39] (2022) Keras. [Online]. Available: https://github.com/keras-team/keras

[40] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society*, 1974.

[41] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Scream: Sketch resource allocation for software-defined measurement," in *Proc. of ACM CoNext*, 2015.

[42] M. Hogan, S. Landau-Feibish, M. T. Arashloo, J. Rexford, and D. Walker, "Modular switch programming under resource constraints," in *Proc. of USENIX NSDI*, 2022.

[43] G. H. John, "Cross-validated c4. 5: Using error estimation for automatic parameter selection," *Training*, vol. 3, 1994.

[44] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[45] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of ICNN'95 - International Conference on Neural Networks*, 1995.

APPENDIX

In this section, we illustrate the theory of the Bayesian Optimizer, the intuition of SS solving algorithm, and some experiments.

A. Bayesian Optimizer Theory

The black-box environment gives a set of observations in the form of $\{\mathbf{x}_n, y_n\}_{n=1}^{N}$, where $y_n \sim \mathcal{N}(f(x_n), \nu)$ and ν is the variance of noise of the function observations. The acquisition function denoted by $a : \mathcal{X} \to \mathbb{R}^+$ determines what point in $\mathcal{X}$ are evaluated via a proxy optimization $\mathbf{x}_{\text{next}} = \text{argmax}_{\mathbf{x}} \text{a}(\mathbf{x})$. The integrated acquisition function where the parameters (summarized by θ) is:

$$\hat{a}\left(\mathbf{x}; \{\mathbf{x}_n, y_n\}\right) = \int a\left(\mathbf{x}; \{\mathbf{x}_n, y_n\}, \theta\right) p\left(\theta \mid \{\mathbf{x}_n, y_n\}_{n=1}^{N}\right) \mathrm{d}\theta \quad (1)$$

The cumulative distribution function of the standard normal as $\Phi(\cdot)$. The expected improvement we choose is:

$$a_{\text{EI}}\left(\mathbf{x}; \{\mathbf{x}_n, y_n\}, \theta\right) = \sigma\left(\mathbf{x}; \{\mathbf{x}_n, y_n\}, \theta\right)\left(\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x}); 0, 1)\right) \quad (2)$$

As the number of observations grows, the posterior distribution d approaches the function of the black-box environment. The recommendations are more closed to user expectations.

B. The Intuition of SS solving algorithm

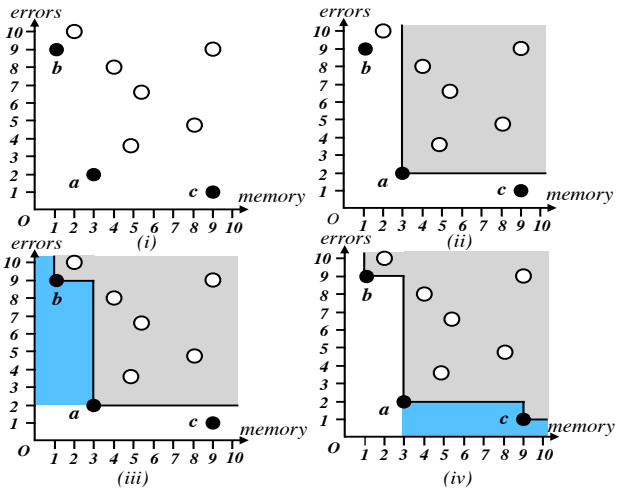

Fig. 13. Intuitions of algorithm 1: pruning search spaces with NN

We show the intuition in Fig. 13. For example, performance targets are set to be *[errors, memory]*, whose weights are one. Thus, the $f(x_1, ..., x_n) = errors + memory$ (*i.e.,* L_1 norm). SketchGuide first finds the nearest neighbor vector a in *(i)*, then a dominates some vectors in *(ii)*. Next, SketchGuide searches the blue space in *(iii)* and finds the nearest neighbor vector b. After b dominates some vectors, the next nearest neighbor c is found and dominates the residual vectors. Finally, we obtain the SS including three vectors a, b, and c. This algorithm fast solves the final SS by avoiding visiting all the vectors. In general, if the SS has k vectors, then the algorithm performs k useful NN queries and $k + 1$ empty queries.

TABLE V
IDENTIFIED PARAMETERS OF A BLOOM FILTER WITH RANGES

Parameters	Range	Description
hash_cnt	[1, 7]	The counts of hash calls
reg_size	[512, 65536]	The number of units per register array

C. Bloom Filter Configuration (Case Study).

We study a case of the SketchGuide generality by configuring a Bloom Filter for a set query task. We keep the same implementation and setup in the case study (Exp#1). Here, we update 1K flows into the Bloom Filter and test the Bloom Filter using the residual flows. We do not profile accuracy because the false negative rates of both configurations are negligible.

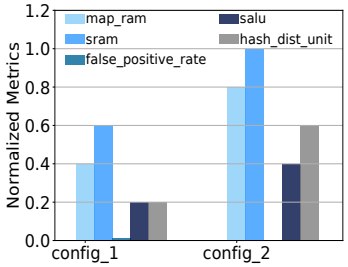

Fig. 14. (Case Study) The SS for users to choose preferred configurations.

P4 program reconfiguration. First, we write a P4 program with a Bloom Filter, which has two parameters, the number of hash functions and the size of a bit array. To reconfigure this program, we identify primitives referring to Table II and identify resource parameters.

```
1:Register<bit<1>, bit<16>>(2048, 0) bf_tbl;
2:Hash<bit<16>>(HashAlgorithm_t.CUSTOM, poly) h;
```

We use the following primitives listed in Table II.

```
1:reg_count        2:hash_init
```

We identify two resource parameters, *i.e.,* `reg_size` and `hash_cnt`, both of which can be configured to trade resources for a false positive rate. The identified parameters and the assigned ranges are shown in Table V. We set the resource budgets and performance expectations of 5 metrics, whose weights are -1.0. The performance expectation of $false\_positive\_rate$ is $[0, 0.2]$. The resource budgets are all $[0, 5]$. The scoring function is$f$=-$false\_positive\_rate$-$SALU$-Map_RAM-$Hash\_call$-$SRAM$).

Configuration recommendation. We filtered the configurations out of resource budgets and solve the final SS by algorithm 1. The metrics are normalized by dividing the upper bounds of the expectations. The configurations with the top-2 highest target scores are selected. Fig. 14 shows the SS including two recommended configurations. The user can choose config_1 due to the negligible false positive rate and minimal consumption of memory footprints.

D. Experiments

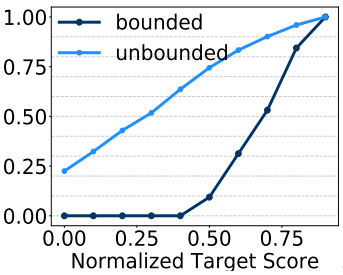

Fig. 15. (Exp#8) SketchGuide eliminates unbounded configurations.

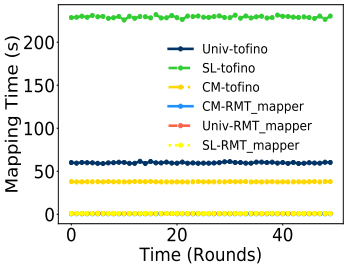

Fig. 16. (Exp#9) SketchGuide fast maps the resource usage using our extended RMT_mapper.

Exp#8: *SketchGuide eliminates unbounded configurations.* Maintaining the same setup in Exp#1, Fig. 15 shows that SketchGuide eliminates the configurations out of the resource budget. Also, SketchGuide efficiently searches SS candidates with only a small portion of unbounded candidates. SketchGuide guarantees the resource usage of a configuration within the budgets by eliminating the one out of the budgets.

Exp#9: *SketchGuide fast maps configurations to resource usage.* We measure the compilation durations of each program using a Tofino compiler and the extended RMT_mapper respectively. The compilation time of a P4 program is the bottleneck of the configuration searching of SketchGuide. Because SketchGuide iteratively searches the configurations and compiles the P4 programs to obtain the resource usage. Fig. 16 shows that the proxy of the Tofino compiler can significantly reduce the compilation durations. SketchLearn (SL) takes more than 200 seconds to finish a compilation in the compiler of a Tofino switch. The extended RMT_mapper can significantly reduce the compilation time and accelerates the deployment of sketches.