FCM-Sketch: Generic Network Measurements with Data Plane Support

Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, Mun Choon Chan
{songch,pravein,lowkh,chanmc}@comp.nus.edu.sg
School of Computing, National University of Singapore

ABSTRACT

Sketches have successfully provided accurate and fine-grained measurements (e.g., flow size and heavy hitters) which are imperative for network management. In particular, Count-Min (CM) sketch is widely utilized in many applications due to its simple design and ease of implementation. There have been many efforts to build monitoring frameworks based on Count-Min sketch. However, these frameworks either support very specific measurement tasks or they cannot be implemented on high-speed programmable hardware (PISA).

In this work, we propose FCM, a framework that is designed to support generic network measurement with high accuracy. Our key contribution is FCM-Sketch, a data structure that has a lightweight implementation on the emerging PISA programmable switches. FCM-Sketch can also be used as a substitute for CM-Sketch in applications that use CM-Sketch. We have implemented FCM-Sketch on a commodity programmable switch (Barefoot Tofino) using the P4 language. Our evaluation shows that FCM-Sketch can reduce the errors in many measurement tasks by 50% to 80% compared to CM-Sketch and other state-of-the-art approaches.

CCS CONCEPTS

• **Networks** → **Network measurement**; **Programmable networks**.

KEYWORDS

Streaming Algorithm, Sketches, Network Measurement, Generic, Data Plane, Programmable Switch, PISA, P4

ACM Reference Format:
Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, Mun Choon Chan. 2020. FCM-Sketch: Generic Network Measurements with Data Plane Support. In *The 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '20), December 1–4, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3386367.3432729

1 INTRODUCTION

Network measurement is indispensable for efficient network management such as load balancing, congestion control, quality of service, scheduling, and anomaly detection [13, 46, 47, 50, 57, 63]. To assist in-network management, fine-grained measurements such as per-flow size [22], heavy hitter detection [54], cardinality [58], etc. are needed.

With the abundance of memory and availability of parallel processing (e.g., SIMD and multi-core), software switches have been used as platforms for network measurement [31, 51, 62]. However, with network link speeds reaching 400 Gbps and switching capacity exceeding tens of Tbps [1], it is challenging for software processing speed to scale accordingly [43]. Techniques like sampling (e.g., NetFlow [3]) are often utilized to reduce measurement overhead. However, it cannot provide accurate and fine-grained statistics. Often, there is an inevitable loss in measurement resolution and accuracy if network monitoring is performed solely in software without hardware (data-plane) support.

To overcome this challenge, sketch, in particular, Count-Min (CM) sketch [22] is often used to support network measurement and queries [20, 21, 34, 37, 42, 53] in the data-plane. However, while Count-Min sketch incurs low computational overhead and has compact memory footprint, it has poor accuracy.

Recently, many network monitoring frameworks [12, 32, 33, 44, 54, 55, 59] have been proposed to operate entirely in the network data-plane by leveraging data-plane programmability [8]. While some of the frameworks [12, 33, 55] are designed to be implemented on programmable switches, they do not support a general set of network measurement tasks with a single data structure. On the other hand, solutions that support generic queries either require substantial hardware resources [32] or cannot be implemented on the switching hardware without significant loss of accuracy [44, 59].

Clearly, what we need is a system that can support high resolution, generality and scaling with the switching fabric throughput at the same time. To this end, we propose the **FCM** (**F**eed-forward **C**ount-**M**in sketch) framework. The FCM framework operates at both switch data-plane and control-plane. In the data-plane, we propose a novel data structure, **FCM-Sketch**, that supports fine-grained measurement, such as per-flow size, cardinality and heavy hitter detection, entirely in the data plane at line-rate. In the control-plane, FCM leverages the CPU and DRAMs to enable complex measurement tasks (e.g. flow size distribution and entropy [13, 15, 23]) using the data collected by FCM-Sketch in the data-plane.

One can think of FCM-Sketch as a better Count-Min sketch that is more memory efficient and accurate. FCM-Sketch can directly replace Count-Min sketch in existing systems that require queries and statistics in the data-plane. The design of FCM-Sketch is based on the following ideas. First, FCM-Sketch has a tree-based feed-forward design that leverages the multi-stage processing pipeline of PISA [5] and distributes the computation along the pipeline. Second, FCM-Sketch uses counters of different sizes at different

CoNEXT '20, December 1–4, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-7948-9/20/12…$15.00
https://doi.org/10.1145/3386367.3432729

levels of the tree. Compared to Count-Min sketch, FCM-Sketch uses a larger number of small-size counters at leaves and a smaller number of large-size counters at the core. In FCM-Sketch, counts of incoming flows are first accumulated in the leaf counters that overflow to the next level (parent) counters upon saturation at the lower levels. Since, in practice, traffic flows are highly skewed with large number of small flows [14], memory of the small size counters at the leaf nodes is utilized more efficiently. This design addresses the inefficiency in the Count-Min sketch design where a uniform counter size is used to record counts for all flow sizes.

To summarize, our contributions are as follow:

- We propose FCM-Sketch, a novel data structure that can be used as a substitute for Count-Min sketch to achieve higher accuracy. FCM-Sketch can be used in a broad class of applications, including network measurements [32, 51, 59, 61], online network control [24, 41, 53, 56], data-plane queries [20, 21, 34, 37, 42] and data streaming [25, 48].
- FCM-Sketch can take advantage of the multi-stage switching pipeline of PISA and can be accurately implemented in data-plane running at line-rate. Also, its compact design reduces data-plane resources (SRAM, N(Stages), etc.) consumption. Our evaluation shows that FCM consumes fewer hardware resources compared to other state-of-the-art measurement frameworks.
- We have designed and implemented the FCM framework on both a software simulator and Barefoot Tofino [8] switches. FCM is resource efficient in the data-plane and supports general measurement tasks with high accuracy.

Our evaluation shows that FCM-Sketch outperforms Count-Min sketch significantly, achieving a 88% reduction in the error for flow size estimation. We have also compared the performance of FCM, ElasticSketch, UnivMon, and FCM+TopK (an implementation of FCM-Sketch coupled with the Top-K algorithm of ElasticSketch). Our evaluation shows that the errors of FCM+TopK is at least 90% less than UnivMon [44] and at least 50% less than ElasticSketch [59] across almost all measurement tasks.

The paper is organized as follows. We introduce the background and motivation in §2. Then, we present the measurement framework of FCM in §3 and §4. Accuracy analysis is presented in §5 and a design to improve CM-based application (FCM+TopK) in §6. The evaluation on software simulator is presented in §7 and the evaluation on Tofino hardware in §8. Finally, we present a related work in §9 and conclusion in §10.

2 BACKGROUND AND MOTIVATION

Traditionally, network monitoring has relied on techniques like NetFlow [3], sFlow [6], etc. Due to memory constraints, these techniques perform sampling on the incoming packets to collect statistics. As a result of their extremely low sampling rates, they do not capture an accurate picture of the network. To overcome this memory-accuracy tradeoff, probabilistic hash-based data-structures called sketch are used in the network devices to get approximate statistics. Predominantly, Count-Min sketch is used to estimate the flow counts due to its simplicity.

Count-Min sketch: The Count-Min (CM) sketch data structure consists of d counter arrays $\{C_1, ..., C_d\}$. Upon an flow's arrival, a set of (d) independent hash functions are applied on the flow id to select

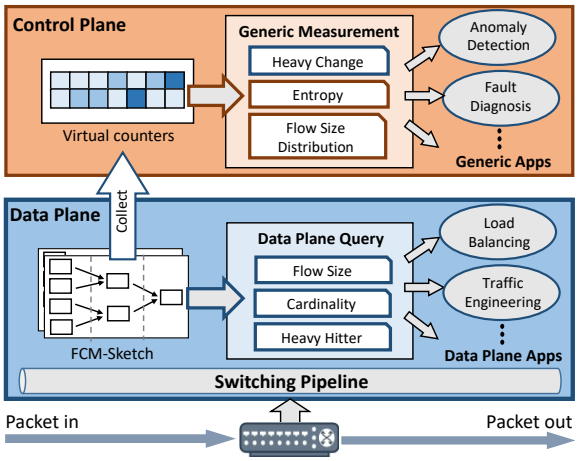


Figure 1: FCM Overview: Simple measurement tasks are available in the data plane, and the control plane calculates more generic statistics from the reported sketches.

d counters (one from each counter array) and the value of each counter is incremented. To estimate a flow's frequency, the same set of independent hash functions are applied and the minimum value among the corresponding counters is obtained. This value provides an estimate of the flow's frequency. Due to its simplicity, CM-Sketch enables a flow's statistics to be accessed at line-rate in the data-plane. This feature has enabled several applications like scheduling, caching, load-balancing, etc to maintain statistics without incurring too much resources. Even though CM-Sketch provides a reasonable performance for a fixed memory footprint, it has the following problems : 1) Memory inefficiency : Since all the counters are of the same size, skewness in traffic with higher number of smaller flows leads to low utilization of allocated memory in a counter. 2) Performance bound: For a given memory size, there is an upper-bound in the accuracy and increasing the computational capabilities (e.g., hashes) does not directly increase the performance [28].

Programmable Switches: The emergence of programmable switches (PISA [5]) have triggered renewed interests in addressing the shortfalls in network monitoring. Recently, approaches such as OpenSketch [61], ElasticSketch, etc. have proposed data-structures using CM-Sketch to perform generic measurements. While they perform much better than the basic CM-Sketch, they cannot be directly implemented on PISA due to complex arithmetic and memory access patterns. To run these algorithms on PISA, approximate implementations are needed and these approximations result in significant loss of accuracy [44, 54, 59].

In this paper, we ask the following question : *"Can we leverage the PISA architecture to design a simple data-structure that can outperform Count-Min and at the same time support generic measurements."*.

FCM-Sketch: We propose *Feed-forward Count-Min sketch*, a tree-based feed-forward scheme to record traffic statistics in multiple stages. The initial stage composes of many small size counters (e.g. 4/8-bit), while the later stages use fewer but larger counters (e.g. 16/32-bit). The main intuition behind the design of the FCM-Sketch is as follows: (1) As the number of small flows tend to dominate in most traffic patterns [14], FCM-Sketch's approach keeps counting of (most) small flows in the leaf nodes and reduces hash collisions for

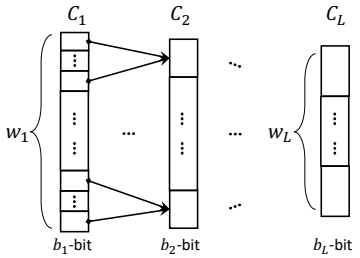


Figure 2: Data structure of single-tree FCM-Sketch.

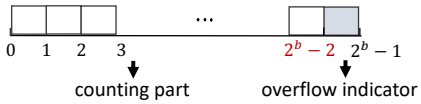


Figure 3: Data structure of a single node with b bits. Count value ranges from 0 to 2^b-2. The value of 2^b-1 is used to indicate a count of at least 2^b-2 plus overflow.

larger flows later along the stages. This is particularly effective for highly-skewed traffic distributions. (2) FCM-Sketch's multi-stage feed-forward data structure naturally fits the multi-stage pipeline in PISA [5], thus enabling implementation that operates at line-rate.

The overall system, the FCM framework, is shown in Figure 1. The framework comprises of : 1) FCM-Sketch in the data-plane (§3) to aggregate statistics and support simple queries like flow size, cardinality and heavy hitter detection at line-rate for applications like load balancing, traffic engineering, etc. 2) Algorithms in the control-plane (§4) to aggregate data-plane statistics from FCM-Sketch to support complex measurements like heavy change detection, entropy and flow size distribution for applications like anomaly detection, network fault diagnosis, etc.

3 DATA-PLANE: FCM-SKETCH

3.1 Data Structure

FCM-Sketch uses a *k-ary* tree-based data structure. We will first present the case for a single tree and then briefly discuss the extension to multiple trees later.

Single-tree FCM-Sketch: As shown in Figure 2, a single-tree FCM-Sketch consists of L node arrays $C_l, 1 \le l \le L$, where L is the total number of stages. Each array C_l is composed of w_l nodes where each node is of size b_l-bit at stage l. The size of C_l is $w_l * b_l$. For a *k-ary* tree structure, w_l decreases by a factor of k from an earlier to the later stage, i.e., $w_{l+1} = \frac{w_l}{k}$. On the other hard, b_l increases with each stage i.e $b_{l+1} > b_l$.

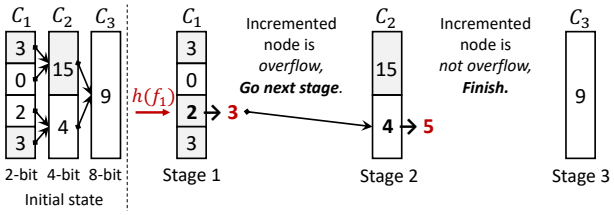
Node in FCM-Sketch: Consider a node in the FCM-Sketch with b bits[1]. The node value is used to convey (1) the counter value and (2) the overflow status. Initially, the counter value is 0 and the overflow status is false. The counter value of a node ranges from 0 to 2^b-2. If the count value is larger than 2^b-2, the node value is set to 2^b-1 and the overflow status is true. Once a node's counter is in the overflow state, all increments have to be carried forward to the next stage. The structure is shown in Figure 3.

In summary, if the node value is between 0 and 2^b-2, the count value is the same as the node value. If the node value is 2^b-1, the node's overflow status is true. Hence, the count value at this node is 2^b-2 and counts at later stages have to be taken into account.

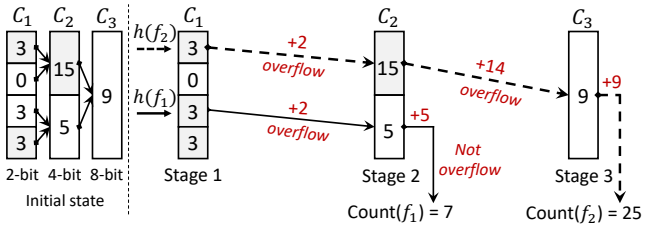
[1]We ignore the subscript of b for brevity

Algorithm 1 Increment(l, i_l)

```
1: if 0 ≤ C_{l,i_l} ≤ 2^{b_l} − 2 then
2:     C_{l,i_l} ← C_{l,i_l} + 1.
3: if C_{l,i_l} = 2^{b_l} − 1 and l < L then
4:     Increment(l + 1, ⌊i_l/k⌋ ).
```



(a) Update operation of a flow key f_1.



(b) Count-query of a flow key f_1 and f_2.

Figure 4: Example operations of FCM-Sketch.

Design Intuitions/Advantages: We highlight the various design intuitions and advantages of FCM-Sketch as follows:

(1) *Collision reduction*: The multi-stage feed-forward design restricts the counters of small flows in the earlier stages, thus reducing collision errors in the later stages with heavy flows.
(2) *Overflow indicator*: An overflow indication using the maximum value of a counter as opposed to a bit-indicator used by previous approaches [19, 60] helps in efficient usage of bit-space as well as minimizes memory accesses.
(3) *Memory accesses*: The multi-stage feed-forward design requires a single memory access per-stage and fits well to the pipeline architecture of PISA.

3.2 Update and Count-Query

Update: Given an incoming packet with flow key f, we choose the node at stage 1 with the hash index $i = h(f) \bmod w_1$. Next, we apply Algorithm 1 by invoking *Increment(1, i)*. In the algorithm, C_{l,i_l} is the node value at stage l with index i_l. If the node in the current stage l is in *overflow* state, then the increment goes to the next stage by invoking *Increment*$(l+1, \lfloor i_l/k \rfloor)$. The algorithm stops when encountering a node that is not in *overflow* state or the final level is reached.

Count-query: To retrieve the count estimate for a flow key, the sum of all corresponding counter values along one or more stages are accumulated until the stage where the corresponding node is not in the *overflow* state or until the last stage has been reached.

While we present the update and count query separately for clarity, these two operations are performed at the same time in practice for efficient memory accesses.

Example - Update (Figure 4a): Consider an FCM-Sketch composed of a binary tree with three stages and counter size $b = [2, 4, 8]$ bits respectively as in Figure 4. Suppose a flow key f_1's packet is hashed into the third counter at stage 1. This counter's value being 2 is already the maximum counting range ($2^2 - 2$). The value is incremented to 3 to indicate that overflow has occurred and the update then moves to the next stage. In the second stage, the flow key is mapped to the second counter with a value of 4. Since the counter is made up of 4 bits at stage 2, the maximum value is 14 ($2^4 - 2$). Hence, the counter is incremented to 5 and the update ends.

Example - Count-query (Figure 4b): Consider a query operation on the same flow f_1. With overflow in stage 1 and no overflow in stage 2, the count for this flow will be 7 (sum of stage 1 (value of 2) and stage 2 (value of 5)). Consider another flow f_2 that is hashed to the first counter in stage 1 and first counter in stage 2. Since both the counters in stages 1 and 2 are in overflow, count for this flow includes all three stages i.e. 2 + 14 + 9 = 25.

Extension to Multi-trees: A single-tree FCM-Sketch is synonymous to CM-Sketch with a single hash table. Hence, multiple trees are a natural extension to a single tree FCM-Sketch to improve accuracy. In the case of FCM-Sketch with multiple trees, multiple independent hash function maps are used. The final count is the minimum value over all the count in multiple trees similar to CM-Sketch. Note that multiple trees can be operated upon in parallel in the data-plane as they use independent memory units.

3.3 Data-Plane Queries

A data-plane query is a query which can be processed using information maintained in the FCM-Sketch without additional processing in the control-plane. FCM-Sketch can be used to answer the following data-plane queries.

- **Flow size estimation:** FCM-Sketch estimates the flow size corresponding to the flow key using a count-query. Note the count can be interpreted in different ways, e.g., bytes, packets, etc.
- **Heavy hitter detection:** FCM-Sketch's flow size estimation can in turn be used to classify flows as heavy hitters using configured thresholds.
- **Cardinality estimation:** FCM-Sketch can be used to estimate the cardinality (number of distinct flows) with respect to a flow key using Linear Counting (LC) [58]. LC estimates the maximum likelihood of the number of empty counters for a given cardinality. Formally, the cardinality estimator is $\hat{n} = -w_1 \log(\overline{w}_1^0 / w_1)$, where $\overline{w}_1^0$ is the average number of empty leaf nodes among those at stage 1. This can be computed in the data plane using lookup tables.

The above measurements have been extensively used for many data-plane applications which are beyond traditional data-plane functionality, such as load balancing of hot objects [34, 37, 42], packet scheduling [53], queue measurement [21], and microburst detection [20], etc.

4 CONTROL PLANE: ALGORITHMS

In this section, we explain the algorithms used in the control-plane to aggregate the statistics from the data-plane and estimate various statistics. The control-plane, equipped with substantially more processing capabilities, converts the FCM-Sketch to virtual counters (§4.1) and applies EM (Expectation Maximization) algorithm (§4.2) to recover information loss in the data-plane caused by hash collisions to obtain flow size distribution. Lastly, we explain the methodology (§4.4) to estimate entropy and detect heavy change that are useful for applications like anomaly detection, fault diagnosis, etc.

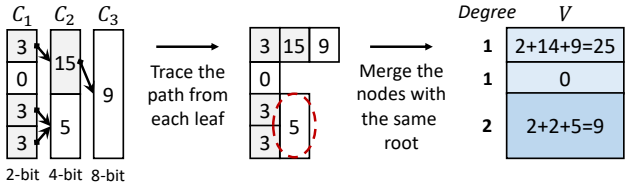


Figure 5: Example of conversion algorithm in the control plane.

4.1 FCM-Sketch to Virtual Counters

The first step is to aggregate the counters of the FCM-Sketch from the data-plane. Since FCM-Sketch is a probabilistic data structure, inaccuracies due to hash collisions are common. Additionally, hash collisions could happen at any stage. Hence, the challenge is to untangle the hash collision. We address this by building a linear counter array, which we call virtual counters. The desired property of the virtual counter is that it needs to capture the relationship among flows that encountered collision and shared the same counters at different stages. Details of the *Conversion Algorithm* is presented below.

Conversion Algorithm: The algorithm consists of two steps to convert a single tree of FCM-Sketch into a virtual counter array.

(1) For each leaf node at level 1, trace the path starting from the leaf node towards the top of the tree until a node that has not overflown or the node at the final level is reached.
(2) All paths (sub-tree) that end at the same (highest level) node are merged into a single **virtual counter**. The value of each virtual counter is the sum of all node counters in the sub-tree. Each virtual counter is also associated with a parameter called the **degree** which is the number of paths merged to form the virtual counter.

It is important to note that in the construction of a virtual counter, the total count is preserved. Therefore, each virtual counter corresponds to the exact count for the given sub-tree it represents.

Example: In Figure 5, we illustrate the conversion algorithm using the previous example. The path of first leaf node $C_{1,0}$ (the count value is 2 and the node is in the overflow state) can be traced to $C_{2,0}$ and ends at $C_{3,0}$. The accumulated count is 2 + 14 + 9 = 25. Since the path has no common leaf node with other paths leading to $C_{3,0}$, a *degree* of 1 is assigned. We denote this virtual counter as V_1^1, the virtual counter with degree 1 and index 1. The process for the second leaf (starting from $C_{1,1}$) is similar, resulting in a virtual counter value of 0 and a *degree* of 1. We denote this count as V_2^1.

The paths from the third and fourth leaf nodes ($C_{1,2}$ and $C_{1,3}$) share the counter at $C_{2,1}$ in stage 2. Both paths end at $C_{2,1}$ as the node is not in the overflow state. Hence, their virtual counters will be combined with the merged count 2 + 2 + 5 = 9, and the virtual counter has a *degree* 2. This virtual counter is denoted as V_1^2.

Each tree in the FCM-Sketch is converted to a virtual array using the above method. A multi-tree FCM-sketch would be converted into multiple virtual arrays.

4.2 Expectation-Maximization

Given the virtual counter arrays V, the next step would be to accurately estimate the flow size distribution ϕ and the total number of flows n. We develop a Maximum Likelihood Estimator (MLE) of ϕ and n under the unobserved latent variables (i.e. hash collisions between flows). We use Expectation-Maximization (EM) algorithm [38], which is an iterative method to find the maximum likelihood estimate of the parameters (ϕ,n). Given two sets of unknown ($\{\phi,n\}$, and latent variables), the EM algorithm consists of two iterative steps: (1) guess the values of ϕ, n to estimate the (expected) latent variables, and (2) compute the better guess of ϕ, n by using the newly estimated latent variables. The output of step 2 can then be used in the next iteration of step 1. This process is repeated either over a fixed time/iterations or till some threshold (e.g. accuracy estimate) is reached.

EM for Single-tree FCM-Sketch: Suppose a single-tree FCM-Sketch and its corresponding virtual counter array V whose maximum degree and value are D and z, respectively. We group the virtual counters of the same degree ξ, and denote the group and its number of counters as V^{ξ} and m^{ξ}, respectively.

At each iteration s, the EM algorithm updates the estimates $\phi^{(s)}$ and $n^{(s)}$. At iteration $s+1$, $\phi^{(s+1)}$ and $n^{(s+1)}$ are updated as: For any possible flow size j, $\phi_j^{(s+1)} = n_j^{(s+1)}/n^{(s+1)}$, $n^{(s+1)} = \sum_{j=1}^{z} n_j^{(s+1)}$, and

$$n_j^{(s+1)} = \sum_{\xi=1}^{D} \sum_{i=1}^{m^{\xi}} \sum_{\beta^{\xi} \in \Omega(V_i^{\xi},\xi)} p(\beta^{\xi}|V_i^{\xi}, \phi^{(s)}, n^{(s)}) * \beta_j^{\xi}, \quad (1)$$

where $\Omega(V_i^{\xi}, \xi)$ is the set of all possible combinations of collision between flows that can build-up the i-th virtual counter of *degree* ξ. In Eqn. 1, the probability is calculated by Bayes' rule:

$$p(\beta^{\xi}|V_i^{\xi}, \phi^{(s)}, n^{(s)}) = \frac{p(\beta^{\xi}|\phi^{(s)}, n^{(s)})\mathbb{I}\{\beta^{\xi} \in \Omega(V_i^{\xi}, \xi)\}}{\sum_{\alpha^{\xi} \in \Omega(V_i^{\xi},\xi)} p(\alpha^{\xi}|\phi^{(s)}, n^{(s)})}, \quad (2)$$

where $p(\beta^{\xi}|\phi, n) = \prod_{j=1}^{z} p(\beta_j^{\xi}|\phi, n)$ and each $(\beta_j^{\xi}|\phi, n)$ follows Poisson$(n\phi_j\xi/w_1)$.

To understand the details of EM algorithm, we discuss the following in §4.3.

- $p(\beta_j^{\xi}|\phi, n)$: probability modeling of latent variables,
- $\Omega(V_i^{\xi}, \xi)$: possible combinations of collisions for V_i^{ξ},
- Initialization and complexity of EM algorithm.

We skip the derivation of estimates for ϕ, n due to space constraints. More details of EM algorithm can be found in [38].

4.3 Details of EM Algorithm

Probability Modeling: Given the flow size distribution ϕ and the total number of flows n, consider $n\phi_j$ number of size j flows. Each of them would be uniformly hashed into the total w_1 number of leaf nodes. As a virtual counter of degree ξ includes ξ leaf nodes, the (prior) probability of each flow hashed into the virtual counter follows Bernoulli($\frac{\xi}{w_1}$). Hence, the (prior) probability of the number of size j flows to be hashed into a virtual counter of degree ξ (i.e., β_j^{ξ}) follows Binomial, and is approximated by Poisson distribution. Formally,

$$\beta_j^{\xi}|\phi, n \sim \text{Poisson}(\frac{n\phi_j\xi}{w_1}).$$

Since we can assume the collision events for different sizes of flows are independent by hashing, $p(\beta^{\xi}|\phi, n) = \prod_{j=1}^{z} p(\beta_j^{\xi}|\phi, n)$.

Note that the previous statement does not hold if conditioned on a specific virtual counter value (data) since the value limits the possible set of flows due to data-dependency. By Bayes' rule, the posterior probability of β^{ξ} given the virtual counter value V_i^{ξ} is computed as :

$$p(\beta^{\xi}|V_i^{\xi}, \phi, n) = \frac{p(\beta^{\xi}|\phi, n)p(V_i^{\xi}|\beta^{\xi}, \phi, n)}{\sum_{\alpha^{\xi} \in \Omega} p(V_i^{\xi}|\alpha^{\xi}, \phi, n)p(\alpha^{\xi}|\phi, n)} = \frac{p(\beta^{\xi}|\phi, n)\mathbb{I}\{\beta^{\xi} \in \Omega(V_i^{\xi}, \xi)\}}{\sum_{\alpha^{\xi} \in \Omega(V_i^{\xi},\xi)} p(\alpha^{\xi}|\phi, n)}$$

where $\mathbb{I}\{\cdot\}$ is an indicator function, Ω is a set of all combinations of flows, and $\Omega(V_i^{\xi}, \xi)$ is a set of all possible combinations of flows for the virtual counter V_i^{ξ} of *degree* ξ.

Notation	Definition
n	Total number of flows
ϕ_j/n_j	Fraction / Number of flows that are of size j.
V	A virtual counter array
D	Maximum degree of counters in V
z	Maximum value of counters in V
V^{ξ}	A group of virtual counters of degree ξ in V
m^{ξ}	Number of counters in V^{ξ}
$\Omega(V_i^{\xi}, \xi)$	A set of possible combinations of collisions between flows that can build-up V_i^{ξ}

Table 1: Notations in EM for single-tree FCM-Sketch.

Likelihood Estimation of $\Omega(V_i^{\xi}, \xi)$: Note that the possibility of building-up the virtual counter of a specific value and *degree* is data-dependent since it is determined by the occurrence of *overflow* events. Hence, the possible set of combinations for a specific virtual counter depends on *how the virtual counter has been produced from the collected FCM-Sketch.*

Virtual counters with the same value and *degree* could have different possible sets based on where their paths have met and how they have been merged. In particular, there are two constraints to consider. First, non-empty virtual counters of degree ξ should include at least ξ flows, because it has ξ paths and each path has at least one flow hashed into its leaf node. Second, based on how V_i^{ξ} is produced, the sum of the counts on each of ξ paths should be large enough to result in overflow.

For example, consider the virtual counter $V_1^2 = 9$ and the set $\Omega(V_1^2, 2)$ in Figure 5. It has two paths which meet in stage 2. Hence, there must be at least two flows and a solution of one flow of size 9 is not possible. Likewise, many such combinations could be ruled out. For example, the combination of two flows with size 1 and 8 is

not possible since a flow of size 1 is not large enough to cause an overflow in the leaf node. With the above constraints, the possible combinations of sizes for 2 flows are {3, 6} and {4 ,5}. Clearly, there are many other combinations of 3 to 9 different flows that can produce the observed value. However, these combinations with more number of flows hashing into 2 leaf nodes are less likely.

Heuristic to Reduce Estimation Complexity: Note that enumerating all possible combinations of collision is impractical since there are enormous number of combinations to consider. In order to reduce the computation time, we use the following observations: (1) There are very small number of (virtual) counters with large value and/or *degree* as the real traffic flows have highly skewed distribution, and (2) collisions of large number of flows are rare. Similar to MRAC [38], we truncate the set of possible combinations based on the counter value and *degree*, and reduce the computational complexity with very small impact on accuracy.

Initialization: In EM algorithm, each iteration uses the estimates produced by the preceding iteration to refine the estimation. In the beginning, it should set the initial guess $\phi^{(0)}$ and $n^{(0)}$. We generate the initial guess as the observed flow size distribution by count queries of all hash index and the number of non-empty leaf nodes, respectively.

EM for Multi-tree FCM-Sketch: The EM algorithm of single-tree FCM-Sketch can be easily extended to multi-tree since the trees operate independently. We present the result for multi-tree based FCM-Sketch in Appendix §A and skip the details of derivation.

4.4 Measurement in the Control Plane

By periodically[2] collecting FCM-Sketch from the data plane and converting the data into virtual counters, the control plane can support complex measurement tasks such as:

- **Flow size distribution:** The distribution of flow sizes through EM algorithm as described in §4.2.
- **Entropy estimation:** The flow size distribution can in turn be used to estimate the entropy [40] by expressing it based on flow size distribution. Formally, entropy $H = -\sum_k (k * \frac{n_k}{m} \log \frac{n_k}{m})$ where n_k is the number of size-k flows.
- **Heavy change detection:** Flows whose sizes in two adjacent time windows have changed over a predefined threshold can be classified as heavy change. Note that if the change of flow size is over the threshold, at least one of sizes for the time windows should be over the threshold. This can be done by collecting the candidate heavy flows over the threshold for both windows. Then the next step would be to compare their count-queries from the collected sketches, and report if the change is over the threshold.

Such measurements can be extended to support many other applications such as detecting security attacks [15, 23], flash crowds [36], or understanding the properties of underlying network traffic [14].

5 ACCURACY ANALYSIS

Similar to CM-Sketch, FCM-Sketch always overestimates the flow size upon hash collisions. In this section, we show the accuracy guarantee of FCM-Sketch's count-query. Let $\mathbf{x} = [x_1, \ldots, x_n]$ be a vector of flow size in data streaming where x_i is the size of the i-th flow. Typically, the accuracy analysis of sketch is configured in terms of two parameters: error fraction (ϵ) and error probability (δ). In the following theorem, we present the accuracy guarantee of FCM-Sketch in terms of the total number of incoming packets ($\|\mathbf{x}\|_1$).

Theorem 5.1. *Suppose the virtual counters converted from FCM-Sketch (§4.1) has a finite maximum degree D. Denote the number of leaf nodes of each tree in the FCM-Sketch as* $w_1 = \lceil \frac{e}{\epsilon} \rceil$ *(e is Euler's number), the number of trees as* $d = \lceil \ln \frac{1}{\delta} \rceil$*, and the accuracy parameters* $\epsilon, \delta > 0$*. Given d pairwise independent hash functions, the count-query* $\hat{x}_i$ *for flow i is bounded by*

$$\hat{x}_i \leq x_i + \epsilon \|\mathbf{x}\|_1 + \epsilon(D-1)(\|\mathbf{x}\|_1 - w_1\theta_1)\mathbb{I}\{\|\mathbf{x}\|_1 > w_1\theta_1\} \quad (3)$$

with probability at least $1-\delta$*, where* $\|\cdot\|_1$ *is 1-norm,* θ_1 *is the maximum counter value at stage 1 (*$2^{b_1}-2$ *with* b_1*-bit), and* $\mathbb{I}\{\cdot\}$ *is an indicator function.*

Note that, if only a single level of tree is used, with the same number of counters, the error bounds for FCM-Sketch and CM-Sketch [22] are the same. Additionally, the accuracy parameter ϵ is inversely proportional to the number of leaf counters by definition.

In Theorem 5.1, the only assumption we make for the analysis is the finite maximum degree D which obviously holds; D is bounded by the number of leaf nodes of a tree. Note that the dependency on D disappears if the total number of packets is less than $w_1\theta_1$. In particular, the condition translates to; For 1.3 MB memory, $w_1\theta_1$ is about $133M$ using two 8-ary trees with 8, 16, 32-bit counters in each stage. This corresponds to 992 Gb traffic with 1000-byte packets.

If the above condition holds true, barring parameter ϵ, the error bound of FCM-Sketch takes the exact same form as that of CM-Sketch. The intuition behind FCM-Sketch's advantage over CM-Sketch is as follows. CM-Sketch uses counters of uniform size and thus requires large counters (e.g., 32-bit) to record large flows. On the other hand, FCM-Sketch utilizes smaller counters (e.g., 8-bit) at the earlier stage, while larger counters are used only for later stages and the number of large counters decreases with the number of stages. For the same amount of memory, we can thus provision for many more (small) counters at the earlier stages. Therefore, the accuracy parameter ϵ for FCM-Sketch is much smaller than that of CM-Sketch with the same memory.

On the other hand, if the condition does not hold, the error bound in Eqn. 3 increases with the maximum *degree D*. This matches with our intuition in that D tends to increase with the number of *overflow* nodes increases which may additionally result in more collision error for count-query. Specifically, the condition would be violated if flow sizes are uniformly large. In this case, the accuracy may drop due to their collisions in the later stages.

Lastly, the error bound in Theorem 5.1 focuses on describing the overflow events at stage 1 as in the last term of Eqn. 3. In Appendix §B, we present its general form with considerations for all stages.

6 APPLICATION STUDY: FCM+TOPK

Many existing network monitoring frameworks for generic measurement tasks rely on the design of CM-Sketch (e.g., ElasticSketch [59], SketchLearn [32], OpenSketch [61]). In this section, we

[2]The frequency of the control-plane is controlled by the user. It can be periodic, or event-driven.

show how FCM-Sketch can replace CM-Sketch and be integrated into ElasticSketch as a representative application and analyze the accuracy improvement.

ElasticSketch [59] consists of two main components: 1) a Top-K algorithm that filters candidate heavy flows and maintains their key-values in multiple hash tables, and 2) a CM-Sketch that maintains statistics for the residual packets after the Top-K algorithm. Since heavy flows have been filtered, the residual flows ideally can be tracked with small size counters. Hence, more number of small size (e.g. 8-bit) counters could be used in CM-Sketch compared to the use of larger (e.g. 32-bit) counters.

While the intuition behind ElasticSketch is effective on highly skewed traffic flows, it requires the Top-K algorithm to minimize the likelihood of the 8-bit counters in the CM-Sketch from overflowing. Thus, ElasticSketch needs to allocate sufficient memory to the Top-K algorithm's tables, resulting in less memory for the CM-Sketch and reducing the overall accuracy.

To overcome this drawback, we could use FCM-Sketch instead of CM-Sketch in ElasticSketch. Lets call this combination **FCM+TopK**. Since FCM-Sketch can handle different flow sizes more efficiently (by allowing overflow to higher level stages), a much smaller amount of memory can be allocated to the Top-K algorithm. As a result, FCM-Sketch can be allocated even larger amount of the available memory to further improve accuracy of the non-heavy flows.

We present the error bound of FCM+TopK as follows:

THEOREM 6.1. *The count-query $\hat{x}_i$ of FCM+TopK for flow i is bounded by*

$$\hat{x}_i \leq x_i + \epsilon\|\mathbf{x_L}\|_1 + \epsilon(D-1)(\|\mathbf{x_L}\|_1 - w_1\theta_1)\mathbb{I}\{\|\mathbf{x_L}\|_1 > w_1\theta_1\} \quad (4)$$

with probability at least $1-\delta$, where $\|\mathbf{x_L}\|_1$ denotes the sum of counts passed after the Top-K algorithm.

Note that if only a single level tree is used, the error bound is same as that of ElasticSketch. Thus, FCM+TopK can have a tighter error bound than ElasticSketch using the same Top-K algorithm. We present the proof in Appendix §B.

7 EVALUATION ON SOFTWARE

In this section, we evaluate the accuracy of the software implementation of FCM compared to existing frameworks. First, we describe the software implementation of FCM-Sketch and other frameworks in §7.1 and the evaluation setup in §7.2. Next, we compare the accuracy of FCM-Sketch and FCM+TopK with CM-Sketch, CU-Sketch [26][3], Hashpipe [54], MRAC, Hyperloglog [27], and PyramidSketch [60] for specific measurement task in §7.3. In §7.4, we look at how parameters of FCM should be selected for different traffic distribution skewness. Finally, we compare the performance of FCM and FCM+TopK with frameworks such as UnivMon [44] and ElasticSketch [59] across generic measurement tasks in §7.5. We defer the evaluation on PISA hardware (Barefoot Tofino) to §8.

7.1 Implementation

We have implemented FCM-Sketch and all the other measurement frameworks in C++. We conducted evaluations on a server with 64 cores (Intel Xeon E5-2683V4@2.1GHZ) and 256GB DRAM. We use BobHash by default as recommended [30]. We implement CM-Sketch, CU-Sketch, Hashpipe, and MRAC using 32-bit counter arrays where counters are uniformly chosen by hash functions. Hyperloglog (HLL) is implemented using a 8-bit counter array and UnivMon is implemented based on multi-level sampling-and-sketching to extract statistics for generic monitoring capability. We use the open source implementation of PyramidSketch with CM-Sketch (PCM) [9]. Since Conservative-Update (CU) can improve the count-query of both FCM and PyramidSketch in a similar degree, we skip the implementation of CU for both. Finally, for ElasticSketch, we use its P4-version platform based on the published source code [10]. We explain the parameters of each algorithm in §7.2.

[3]CM-Sketch with Conservative Update

Measurement tasks	Metrics	Benchmark solutions
Flow size estimation	ARE, AAE	Count-Min (CM), CU, PCM
Heavy hitter detection	F1-score	UnivMon, Hashpipe (HP)
Cardinality estimation	RE	UnivMon, Hyperloglog (HLL)
Flow size distribution	WMRE	MRAC
Entropy estimation	RE	UnivMon, MRAC
Support-All	-	ElasticSketch

Table 2: Measurements, evaluation metrics and benchmarks.

7.2 Evaluation Setup

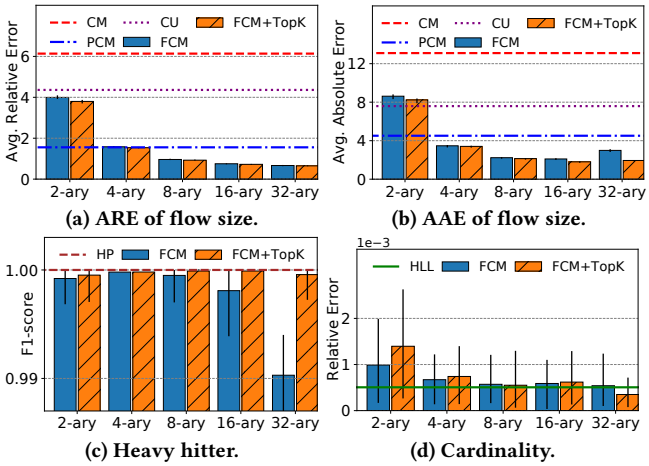
Traffic Traces: We use 32 non-overlapping traces from the CAIDA Equinix-NYC data monitor [2] collected on 19th January, 2019. We use source-IP as the flow key. We did not use finer classification such as the 5 IP tuples because that would result in many more short flows and even higher skewness in the data trace. Each trace contains about 20M packets and 0.5M distinct flows in a 15s window.

Parameters Configurations: FCM-Sketch is composed using two trees which have 8, 16, and 32-bit counters in each stage by default. We use byte-aligned counters for ease of implementation, which is crucial for execution on the programmable hardware. For FCM+TopK, we use a single level of Top-K algorithm with 4K entries and use the rest of the memory for the FCM-Sketch. CM-Sketch and CU-Sketch are composed using three counter arrays which have been reported to have the best accuracy [28]. Similarly, MRAC uses a single counter array for the best accuracy. Hashpipe uses 6 hash tables, and PyramidSketch is combined with CM-sketch (called PCM) using 4 hashes, 4-bit counter size, and 64-bit machine word. The parameters for Hashpipe and PCM are the same as those used in the published version. Univmon uses 16 levels, where each level records 2K heavy hitters using a heap and use the rest of the memory for the sketch. Finally, for ElasticSketch, we use 4 levels for Top-K algorithm where each level has 8K key-value entries and the rest of the memory for the sketch. Unless stated otherwise, we follow the above-stated configurations in the rest of the evaluation.

Task, Metric and Benchmark: We perform different measurement tasks (refer to Table 2) and report the performance metrics corresponding to each tasks[4]. We additionally highlight the current benchmark solution for each of the task in Table 2. We explain how the metrics are derived as follows:

- *ARE (Average Relative Error)*: $\frac{1}{N}\sum_{i=1}^{N}\frac{|x_i-\hat{x}_i|}{x_i}$, where N is the number of flows, x_i and $\hat{x}_i$ are true and estimated flow sizes.

[4]We will not show the result for heavy change detection as it is very close to that of heavy hitter detection.



(a) ARE of flow size. (b) AAE of flow size.

(c) Heavy hitter. (d) Cardinality.

Figure 6: Accuracy comparison of data-plane queries for different k-ary trees compared to CM, CU, PCM, HP and HLL.

- *AAE (Average Absolute Error)*: $\frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{x}_i|$, where there is no normalization.
- *F1-score*: $\frac{2\times PR\times RR}{PR+RR}$ where PR (Precision Rate) is the ratio of true instances reported and RR (Recall Rate) is the ratio of reported true instances.
- *WMRE (Weighted Mean Relative Error)* [38]: $\frac{\sum_{i=1}^{z}|n_i-\hat{n}_i|}{\sum_{i=1}^{z}\frac{n_i+\hat{n}_i}{2}}$, where z is the maximum flow size, n_i and $\hat{n}_i$ are true and estimated numbers of flows with size i, respectively.
- *RE (Relative Error)*: $\frac{|\hat{x}-x|}{x}$, where x and $\hat{x}$ are true and estimated statistics, respectively.

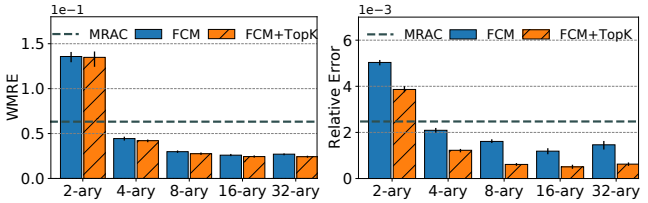
In our experiments, we set the prefixed threshold of heavy hitter detection as $10K$ packets. This threshold is about 0.05% of the total number of packets in one trace.

7.3 Accuracy of FCM and FCM+TopK

The accuracy of FCM-Sketch depends on its configuration, i.e., *k-ary* tree structure. In this section, we investigate the impact of this configuration on the measurement accuracy of FCM-Sketch and FCM+TopK by varying k from 2 to 32 for a fixed 1.5 MB of memory. In our settings, configurations with higher k values will result in more leaf nodes and fewer root nodes. We do not evaluate configurations of $k \geq 64$ as this results in too few counters in the later stages.

7.3.1 Query Accuracy. We evaluate the accuracy of both FCM-Sketch and FCM+TopK compared to the baseline such as CM, CU, PyramidSketch with CM (PCM), Hashpipe (HP), and Hyperloglog (HLL) for data-plane queries and MRAC for control-plane queries.

Data-Plane Queries: We show the results in Figure 6 with error bars of 10% to 90%. We observe that FCM-Sketch and FCM+TopK perform significantly better than CM-Sketch and CU-Sketch in terms of flow-size estimation. Specifically, in Figure 6a, using 16-ary trees, the relative errors (AREs) of FCM-Sketch and FCM+TopK are both 88% lower than CM-Sketch. Similarly, in Figure 6b, the absolute errors (AAEs) of FCM-Sketch and FCM+TopK are 84% and 86% lower than CM-Sketch, respectively. Compared to PCM, both achieve 53% lower AREs, and 53% and 60% lower AAEs, respectively.



(a) Flow size distribution. (b) Entropy.

Figure 7: Accuracy comparison of control-plane queries for different k-ary trees compared to MRAC.

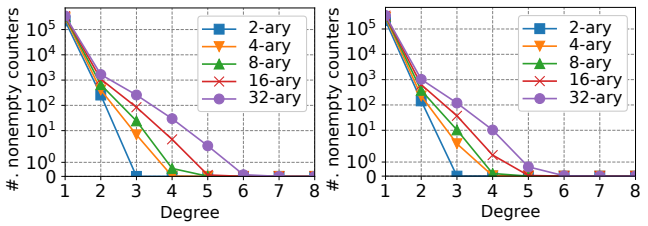


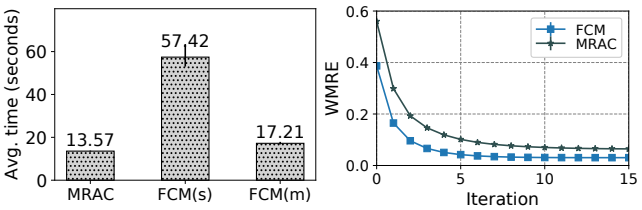
Figure 8: Histogram of non-empty virtual counters from FCM (left) and FCM+TopK (right) for different k-ary trees.

Next, for heavy-hitter detection and cardinality, FCM-Sketch and FCM+TopK show comparable performance with task-specific solutions, e.g., Hashpipe and Hyperloglog for the data set used.[5] For heavy-hitter detection (Figure 6c), we observe that while FCM+TopK performs better than FCM-Sketch at all configurations, there is a significant improvement of FCM+TopK over FCM-Sketch at $k = 32$ due to increase in hash collisions. In the case of cardinality (Figure 6d), we observe that the relative error decreases for both FCM-Sketch and FCM+TopK with increase in k.

Control-Plane Queries: We show the accuracy for control-plane queries (flow-size distribution and entropy) using the EM algorithm of FCM-Sketch and FCM+TopK against MRAC in Figure 7. For $k \geq 4$ (*k-ary* trees), compared to MRAC, FCM-Sketch and FCM+TopK have smaller errors. Specifically when using 16-ary trees, both have 59% and 62% smaller errors (WMRE) for flow size distribution (Figure 7a) and 52% and 80% lower relative errors (RE) for entropy estimation (Figure 7b). MRAC performs better than FCM and FCM+TopK for the 2-ary trees due to higher hash collisions at $k = 2$. This shows that the ability to have more counters at the lower level of trees by using higher k improves the memory utilization and thus achieves better performance.

Note that while the errors of FCM-Sketch, except heavy hitter detection, tend to decrease with increasing k, the AAE of flow size (Figure 6b) and RE of entropy estimation (Figure 7b) increase when k is 32. This is because collisions of flows are more likely to occur at later stages with only a small number of counters, and the flows at later stages are heavy. The same reason explains why the accuracy (F1-score) for heavy hitter detection (Figure 6c) decreases at $k = 32$ due to the increase of collisions between heavy flows. On the contrary, FCM+TopK always achieves high accuracy for all configurations. This is because its Top-K algorithm isolates candidate heavy flows from FCM-Sketch thus reduces possible collisions in counters at later stages.

[5] Hyperloglog can perform better than FCM-Sketch, which uses linear counting, for data set with much larger flow cardinality.



(a) Runtime of 1 iteration. (b) Convergence.

Figure 9: Actual runtime (per-iteration) and convergence of EM algorithm.

7.3.2 Effectiveness of Complexity Reduction Heuristic. We portray the effect of applying the heuristic based on degree to reduce the complexity of the EM algorithm (§4.3). We express the complexity in terms of number of non-empty virtual counters, since small number of counters reduce the estimation complexity. In Figure 8, we depict the average number of non-empty virtual counters for different degrees of FCM by generating the counters with 100 random seeds of hashing. We observe that the numbers of virtual counters tend to exponentially decrease with the increase in degree (§4.1). In particular, the number of counters for degree higher than 2 is less than 100 and 50 for 16-ary of FCM-Sketch and FCM+TopK, respectively. Recall that the computation overhead of EM increases rapidly with the number of counters with high degree. The small number of higher degree counters allows us to effectively reduce the estimation complexity with very small accuracy loss.

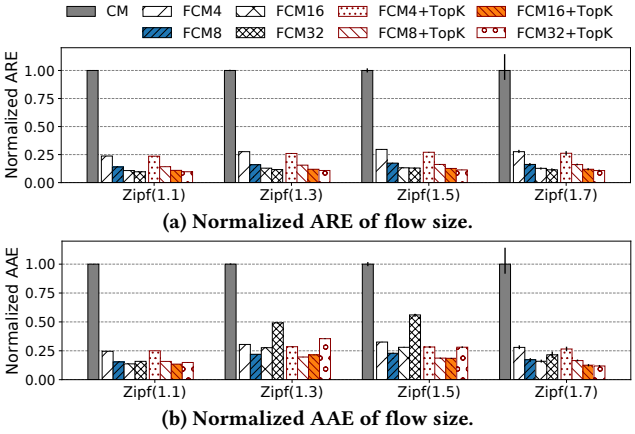
Figure 9a illustrates the per-iteration runtime of EM. In the runtime evaluation, we use 8-ary trees for FCM-Sketch with a single-thread (called FCM(s)) and a multi-threaded (called FCM(m)) version. As the EM processing for different degrees and trees are independent and can be computed in parallel, multi-thread processing can provide significant speedup. We observe that FCM(m) performs 3-4 times faster than the corresponding single-threaded version, but still takes slightly longer time than MRAC. Note that, by using the complexity reduction technique, EM loops for higher degrees are much efficient. Therefore, the major part of the runtime of the multi-threaded FCM is governed by that of the degree 1 counters. Note that each MRAC's counter is equivalent to a virtual counter with a single path. Hence, the complexity of EM for MRAC and for FCM with degree 1 is similar.

Figure 9b shows WMRE varies with the number of EM iterations. We observe that the error for FCM stabilizes within 5 iterations. Also, FCM achieves lower error for the same number of iterations than MRAC. Therefore, in spite of consuming a slightly longer per-iteration runtime, FCM can achieve better performance than MRAC for an fixed running time budget.

7.4 Parameterization of FCM

The key parameters of FCM-Sketch are (1) k, the number of branches in the k-ary tree, (2) the number of trees, and (3) counter size at each level of the tree. For ease of implementation, we use byte-aligned counter sizes (e.g., 8, 16, 32-bit) at each level of the tree. In this section, we look at how the number of branches (k) and number of trees should be selected based on how their performance varies with different traffic mixture (i.e., flow size distribution).

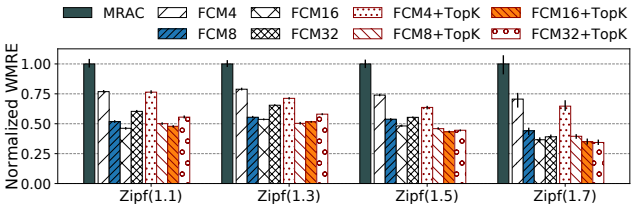
7.4.1 Parameter k. The key parameter of FCM is k (number of branches of the k-ary trees) as a sub-optimal choice for k may lead



(a) Normalized ARE of flow size.

(b) Normalized AAE of flow size.

Figure 10: Comparison of FCM and FCM+TopK for various traffic distribution using different parameter (k-ary) normalized to CM-Sketch.



(a) Normalized WMRE of Flow size distribution.

Figure 11: Comparison of FCM and FCM+TopK for various traffic distribution using different parameter (k-ary) normalized to MRAC.

to memory under-utilization and deterioration of accuracy (§7.3). Since the traffic distribution can vary dynamically, it is hard to determine the optimal parameter. While it is possible to vary k dynamically depending on the counter values at different levels, such dynamic reconfiguration is difficult to implement in practice, especially on the actual hardware. Instead, we investigate how a static k parameter can be selected to provide the best trade-off on diverse traffic mixture. We generate synthetic traces following Zipf (α) distribution [49] with different skewness α (between 1.1 to 1.7). Each trace has a fixed total volume of 20M packets and an average flow size of about 50 packets (similar to the CAIDA trace). The maximum flow size in each trace varies between 400 to 100K packets. The memory size is set to 1.5 MB. We vary k between 4 to 32, and normalize the accuracy of FCM and FCM+TopK to CM-Sketch and MRAC for flow size and its distribution estimation, respectively. The results are shown in Figure 10 and 11.

We observe that all configurations of FCM and FCM+TopK outperform CM-Sketch and MRAC for all traffic distributions. Specifically, it is notable that higher k does not always provide better performance. For instance, when α is 1.3 or 1.5, FCM-Sketch using 32-ary shows nearly 2x higher AAE than the corresponding 4-ary, and slightly higher WMRE than the corresponding 8-ary. This is because of the likelihood of flows to overflow the leaf nodes and end up with collision at later stages of FCM. In contrast, for FCM+TopK, the performance using higher k becomes less sensitive to the traffic skewness because a Top-K algorithm filters out the heavy flows.

Task (Metric)	FCM-Sketch			FCM+TopK		
	2	3	4	2	3	4
Flow size (ARE)	0.961	0.625	**0.622**	0.723	0.487	**0.471**
Flow size (AAE)	2.233	1.422	**1.394**	1.803	1.216	**1.170**
Flow size dist. (WMRE)	**0.030**	0.053	0.097	**0.024**	0.040	0.069
Entropy (RE)	**0.0016**	0.0031	0.0050	**0.0005**	0.0015	0.0027
Cardinality (RE)	0.0006	0.0009	0.0005	0.0006	0.0006	0.0005

Table 3: Comparison of FCM and FCM+TopK for different number of trees. The lower value is better.

Take-away: For a static parameter setting, k should be able to achieve good performance for a majority of the traffic distributions. Our evaluation shows that 8-ary for FCM-Sketch and 16-ary for FCM+TopK achieve good accuracy trade-off for different tasks and traffic skewness.

7.4.2 Number of Trees. Table 3 shows the performance comparison of FCM and FCM+TopK using different number of trees (between 2 to 4). We use 8-ary for FCM and 16-ary for FCM+TopK in the evaluation. We observe that more trees provide higher accuracy for flow size estimation but lower accuracy for flow size distribution and entropy. However, since the computation and resource overhead of FCM significantly increases with the number of trees used (e.g., hash, arithmetic, etc), we choose to use 2 tree configuration of FCM in the rest of the evaluation.

7.5 Comparison with State-of-the-Art Approaches

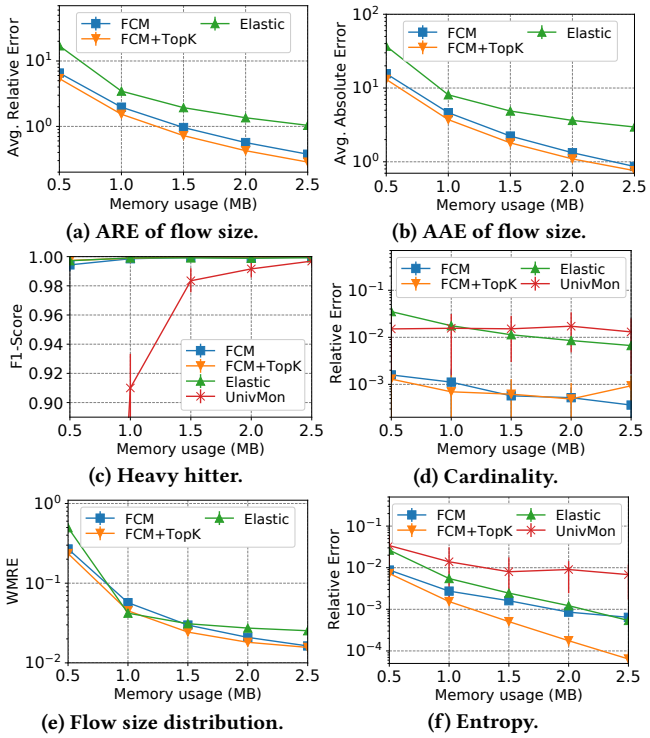
In the previous evaluation, the comparison focuses on baseline such as CM-Sketch and MRAC. In this section, we evaluate the accuracy of FCM and FCM+TopK as general measurement platforms and compare them with existing state-of-the-art approaches, namely UnivMon and ElasticSketch. We compare all 4 frameworks given the same amount of memory. We do not evaluate the performance of UnivMon for flow size and flow size distribution as these measurement tasks were not evaluated in the original paper. Based on the previous results, we use 8-ary trees for FCM-Sketch and 16-ary trees for FCM+TopK.

Flow Size Estimation (Figure 12a-12b): FCM and FCM+TopK achieve better accuracy/memory trade-off than ElasticSketch. When using 1.5 MB of memory, the ARE and AAE of FCM are 50% and 54% lower than ElasticSketch respectively. FCM+TopK achieves even better performance. The ARE and AAE are both 63% lower than ElasticSketch.

Heavy Hitter Detection (Figure 12c): FCM and FCM+TopK achieve F1-scores of above 99.4% and 99.7% respectively. All three frameworks are significantly better than UnivMon and can achieve F1-score of at least 99.9% when using 1.0 MB or more of memory.

Cardinality Estimation (Figure 12d): FCM and FCM+TopK outperform the other solutions more than 10× in all memory sizes. The key reason comes from having more (leaf) counters to cater to the large number of small flows.

Flow Size Distribution (Figure 12e): All three algorithms perform well for flow size distribution. ElasticSketch's use of the Top-K algorithm helps the EM algorithm to minimize the estimation error for heavy flows. Nevertheless, we observe that FCM+TopK always achieves the lowest error.



(a) ARE of flow size. (b) AAE of flow size. (c) Heavy hitter. (d) Cardinality. (e) Flow size distribution. (f) Entropy.

Figure 12: Accuracy comparison for five measurement tasks.

Entropy Estimation (Figure 12f): We observe that FCM is more accurate than the other solutions for all memory sizes. When using 1.5 MB of memory, the RE of FCM is 34% and 80% lower than ElasticSketch and UnivMon on the average respectively. FCM+TopK achieves even better performance, having 69% lower RE than FCM on the average.

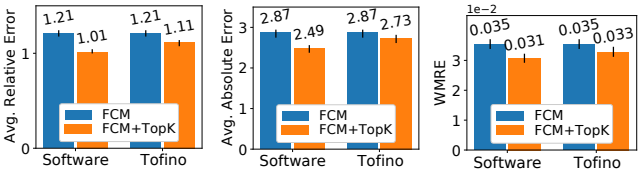
Take-away: In summary, we observe that FCM-Sketch is able to achieve equal or better accuracy than the state-of-the-art for all measurement tasks. When combined with a Top-K algorithm, FCM+TopK improves the performance futher and achieves the best performance for all measurement tasks.

8 EVALUATION ON PISA HARDWARE

In this section, we describe how FCM-Sketch and FCM+TopK are implemented on PISA programmable switches in §8.1. We evaluate their accuracy on commodity Barefoot Tofino [7] switches in §8.2 and hardware resource utilization in §8.3.

8.1 Hardware Implementation

Data-Plane - FCM-Sketch: The design of FCM-Sketch takes advantage of the feed-forwarding multi-stage pipeline of PISA and can be easily implemented in the hardware. We implement FCM-Sketch in P4 [16] using ≈350 lines of code. Each layer of the tree (counter array) is implemented by a register array in SRAM. For multi-tree based FCM-Sketch, each tree operates on independent memory units in parallel. We leverage the Stateful Arithmetic-logic Units (stateful ALUs) which allow write-and-return of a register value with user-defined conditions within a single stage. Each stateful ALU updates a register at each stage, and a decision to move on to the next stage of FCM-Sketch is taken based on the output



(a) ARE and AAE of Flow size. (b) Flow size dist.

Figure 13: Accuracy comparison for implementations on our testbed and Tofino using same memory size.

register value as shown in Algorithm 1. The data-plane supports count-query and heavy hitter detection using the output values. For cardinality estimation, we pre-install a variable-estimate lookup table using TCAM matching rules using P4 [16]. The source-code of FCM-Sketch is available at [11].

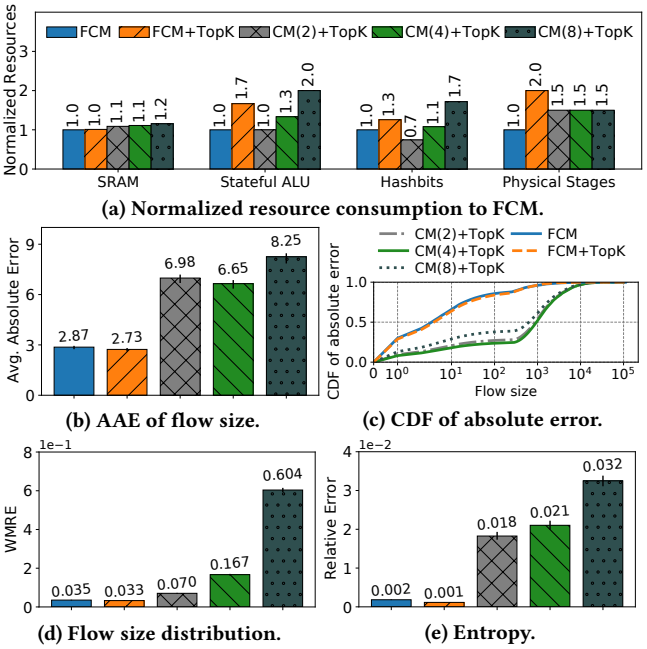
Data-Plane - FCM+TopK: While the design of FCM-Sketch can be faithfully implemented on PISA, this does not hold true for FCM+TopK. The challenges of implementing FCM+TopK on PISA, which also apply to other approaches such as ElasticSketch and Hashpipe [54], are as follows. First, similar to Hashpipe [54], the high-level idea of ElasticSketch's Top-K algorithm is to compare-and-evict small flows in multi-stage key-value hash tables with incoming (possibly heavy) flows. Unfortunately, such a "multi-stage rolling" process requires memory access patterns that is prohibitive in today's commodity PISA switches (see [12] for more details). Second, it needs to swap a key-value pair stored in registers with metadata (or PHV) fields during eviction. However, existing stateful ALUs can modify only a limited number of fields in a single stage. For these reasons, the algorithms can only be implemented on the actual hardware with approximations that can result in significant loss of accuracy.

We implement FCM+TopK using a modified version of ElasticSketch's Top-K algorithm based on duplicate hash tables and stateful ALUs. Instead of multi-level, we use one level of hash table to filter heavy flows.

Control-Plane: We implement the control-plane in C. The control-plane runs the EM algorithm and answers generic measurement queries. To access sketch data in the data-plane, we read FCM-Sketch registers from the data plane in batch using runtime APIs.

8.2 Accuracy Comparison

8.2.1 Software vs Hardware Implementation. We evaluate the accuracy of FCM-Sketch and FCM+TopK implemented on the Barefoot Tofino switch compared to the BMv2 versions for two representative measurements (flow size and its distribution) in Figure 13. Note that the "software versions" for FCM-Sketch and FCM+TopK shown here are the same implementations as the ones in §7.5. The same configurations are used, except that we allocate about 1.3 MB memory size to comply with the hardware configuration limits (e.g., the size of register array in a stage). With a different Top-K implementation, we observe that the FCM+TopK implementation on Tofino shows a small increase in error compared to the BMv2 implementation. As expected, there is no difference in performance between the software and hardware implementations of FCM-Sketch. Note that while the use of duplicate hash tables and stateful ALUs to approximate the Top-K algorithm (§8.1) does not have significant impact on the accuracy of FCM+TopK, this approximation will not work well for ElasticSketch. This is because the simplified Top-K algorithm would allow too many heavy flows to go to the CM-Sketch, resulting in a large increase in error.



(a) Normalized resource consumption to FCM.
(b) AAE of flow size.
(c) CDF of absolute error.
(d) Flow size distribution.
(e) Entropy.

Figure 14: Normalized resource consumption and accuracy comparison for FCM, FCM+TopK, and CM+TopK on Tofino.

8.2.2 Performance Comparison. We compare the performance of FCM, FCM+TopK and CM+TopK on the Barefoot switch. CM+TopK emulates the performance of ElasticSketch. Due to hardware limitations, the TopK algorithms of FCM+TopK and CM+TopK use only one level of hash table. The same configurations for FCM and FCM+TopK as stated in §7.5 are used. For CM+TopK, we use d arrays of 8-bit registers (called CM(d)+TopK) and allocate 16K entries for one level of Top-K algorithm. As it is not possible to configure these algorithms to use the same amount of hardware resources because the algorithms and data structures are different, we compare the normalized hardware resource consumption of the different algorithms using FCM as the baseline. Figure 14a depicts the relative amount of resources (SRAM, Stateful ALUs, Hashbits and physical stages) used by the different algorithms on Tofino.

Our evaluation shows that FCM and FCM+TopK achieve at least 50% lower error than any CM+TopK for overall measurement tasks (Figure 14b-e), while using similar amount of hardware resources. Specifically, we observe that the error of CM+TopK mainly comes from large flows. This is because large flows are not sufficiently filtered by the TopK algorithm and these flows cause the 8-bit registers to overflow. Unfortunately, neither allocating more memory on Top-K nor using 16-bit registers improve the accuracy. On the contrary, FCM-Sketch achieves higher accuracy and its performance is further improved with the Top-K algorithm.

8.3 Resource Overhead

Hardware Resource Utilization: Table 4 shows the hardware resource overhead of FCM-Sketch, FCM+TopK and switch.p4 [4] on Tofino. The sketch configuration for FCM-Sketch and FCM+TopK

Resource	switch.p4	FCM-Sketch	FCM+TopK
SRAM	30.52%	9.38%	9.48%
Match Crossbar	37.50%	2.28%	5.40%
TCAM	28.12%	0.0%	0.0%
Stateful ALUs	22.92%	12.50%	20.83%
Hash Bits	33.43%	2.02 %	2.54%
VLIW Actions	36.98%	1.30%	2.60%
Physical Stages	12	4	8

Table 4: Comparison of hardware resource consumption on Tofino with 1.3 MB memory

Solutions	Measurement	No. Stages	Stateful ALUs
FCM-Sketch	**Generic**	**4**	**12.50%**
FCM+TopK	**Generic**	**8**	**20.83%**
SketchLearn [32]	Generic	9	68.75%
QPipe [33]	Quantile	12	45.83%
SpreadSketch [55]	Superspreader	6	12.50%
HashPipe [54]	Heavy hitter	BMv2 Implementation	
ElasticSketch [59]	Generic	BMv2 Implementation	
UnivMon [44]	Generic	BMV2 Implementation	

Table 5: Comparison of resource consumption for existing solutions on Tofino.

are the same as those used in the evaluation (§7.5). The baseline switch.p4 [4] implements various common networking features applicable to a typical datacenter switch.

We observe that FCM-Sketch consumes only 4 stages and a small amount of resources, while FCM+TopK consumes four additional stages. Both FCM and FCM+TopK consume a relatively higher proportion of stateful ALUs compared to switch.p4. Nevertheless, FCM-Sketch can fit easily into existing switch.p4 for data-center switches and still have sufficient resources left to allow additional applications (e.g. scheduling, load-balancing) to be implemented.

Comparison with Existing Solutions: We highlight the resource consumption of existing measurement solutions implemented in Barefoot Tofino switches in Table 5. Note that the solutions for a specific measurement task (QPipe [33], SpreadSketch [55]) as well as for generic measurement tasks (SketchLearn [32]) consume much more resources (stages and stateful ALUs) than FCM-Sketch. For frameworks such as Hashpipe, ElasticSketch and UnivMon, only the BMv2 versions are available. Implementations of these framework on the Tofino switches would involve non-trivial changes.

Resources for Data-plane Queries: To support data-plane queries such as cardinality estimation, FCM-Sketch requires additional resources. These resources include a small number (< 10) of TCAM entries, 10.42% of stateful ALUs, and one extra stage to calculate a final result of count-query. For cardinality estimation, the stateful ALUs track the number of empty leaf counters $\overline{w}_1^0$ and TCAM is used to implement lookup tables for its corresponding estimator $\hat{n}$ (§3.3). Detailed discussion about TCAM-based implementation is presented in Appendix §C.

Accuracy-Complexity Tradeoff: FCM-Sketch needs more processing time than CM-Sketch using sequential, single-threaded processing. However, as the data structure of FCM is easily fitted to the PISA switching pipeline and hardware constraints, processing remains at line-rate and the impact of more physical stages over CM on latency remains very small in the range of tens of nanoseconds.

9 RELATED WORK

Measurement on Programmable Switches: With the flexibility of programmable data plane [5, 8, 16, 18], network measurement has been extensively studied in various applications [20, 35, 42, 52, 53]. There have been many efforts to accurately estimate measurements such as flow size [22], quantile [33], heavy hitter [12, 29, 54], cardinality [52, 55], or entropy [39]. Unfortunately, to concurrently meet the requirements of diverse applications, significant processing resources, which are at a premium in switching ASICs, have to be consumed. Therefore, it is imperative that an unified data structure can provide accurate results for these general measurement tasks.

Sketches with Generality: UnivMon [44] leverages recursive sampling-and-sketching based on universal streaming theory [17]. However, its heavy hitter collection at each step is non-trivial to implement on hardware. SketchLearn [32] reduces user-burden of configuration-tuning through statistical inferences, but incurs loss of information for the versatility. It is less accurate than Count-Min sketch in practice. ElasticSketch [59] encodes non-heavy flows into Count-Min sketch with small-size (e.g., 8-bit) counters after filtering heavy flows with key-value hash tables. Specifically, it proposes a novel Top-K algorithm which achieves a higher accuracy than a recent proposal (Hashpipe [54]). Unfortunately, both Hashpipe and ElasticSketch cannot be implemented on current commodity programmable switches without significant accuracy loss [12].

Counter-sharing: One effective way to improve memory efficiency is to share underutilized bits between counters, called counter-sharing. Counter Braids [45] is one of the earliest work that apply counter-sharing. Cold Filter [62] proposes a novel meta-framework by filtering small flows at the first stage and running additional algorithms separately on the residual large flows. Although Cold Filter shares some similarity with FCM-Sketch's design, it uses entirely different algorithms and data structures and cannot be easily implemented in the data-plane. To the best of our knowledge, FCM-Sketch is the first counter-sharing work that runs efficiently on PISA and supports generic measurements with high accuracy.

10 CONCLUSION

We present FCM, a framework supporting accurate and various network measurement tasks by leveraging a sketch design of feed-forwarding counters called FCM-Sketch. FCM-Sketch's design fits well with programmable switch architecture (PISA) and can be implemented on PISA efficiently, consuming only a small percentage of hardware resources. Additionally, FCM-Sketch can be used to improve the performance of Count-Min Sketch based applications. Our evaluation shows that FCM achieves at least 50% lower error rates than other state-of-the-art sketches across almost all measurement tasks.

ACKNOWLEDGMENTS

We sincerely thank the anonymous reviewers and our shepherd for their thoughtful comments and invaluable feedback. We would also like to thank Raj Joshi and Nishant Budhdev for their comments and suggestions on the paper. This research is supported by the Singapore Ministry of Education Academic Research Fund Tier 2 (Grant Number: MOE2019-T2-2-134).

REFERENCES

[1] Barefoot tofino 2. https://www.barefootnetworks.com/press-releases/barefoot-networks-unveils-tofino-2-the-next-generation-of-the-worlds-first-fully-p4-programmable-network-switch-asics/.

[2] The caida ucsd anonymized internet traces - 20190117. http://www.caida.org/data/passive/passive_dataset.xml.

[3] Netflow. https://en.wikipedia.org/wiki/NetFlow.

[4] P4 Language Consortium. 2018. Baseline switch.p4. https://github.com/p4lang/switch/blob/\master/p4src/switch.p4.

[5] Portable Switch Architecture. https://p4.org/p4-spec/docs/PSA-v1.0.0.pdf.

[6] Sflow. https://en.wikipedia.org/wiki/SFlow.

[7] Wedge 100bf-32x. https://www.edge-core.com/productsInfo.php?\cls=1&cls2=180&cls3=181&id=335.

[8] The world's fastest and most programmable networks. https://www.barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/.

[9] Pyramidsketch source code. https://github.com/zhouyangpkuer/Pyramid_Sketch_Framework, 2017.

[10] Elasticsketch source code. https://github.com/BlockLiu/ElasticSketchCode, 2018.

[11] Fcm-sketch source code. https://github.com/fcm-project, 2020.

[12] R. B. Basat, X. Chen, G. Einziger, and O. Rottenstreich. Designing heavy-hitter detection algorithms for programmable switches. *IEEE/ACM Transactions on Networking*, 2020.

[13] I. Basicevic, S. Ocovaj, and M. Popovic. The value of flow size distribution in entropy-based detection of dos attacks. *Security and Communication Networks*, 2016.

[14] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, 2010.

[15] P. Bereziński, B. Jasiul, and M. Szpyrka. An entropy-based network anomaly detection method. *Entropy*, 2015.

[16] 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 Computer Communication Review*, 2014.

[17] V. Braverman and R. Ostrovsky. Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2013.

[18] Cavium. Xpliant ethernet switch product family, 2018.

[19] M. Chen, S. Chen, and Z. Cai. Counter tree: A scalable counter architecture for per-flow traffic measurement. *IEEE/ACM Transactions on Networking (TON)*, 2017.

[20] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, and O. Rottenstreich. Catching the microburst culprits with snappy. In *Afternoon Workshop on Self-Driving Networks*, 2018.

[21] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, and T.-Y. Wang. Fine-grained queue measurement in the data plane. In *ACM CoNEXT*, 2019.

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

[23] J. David and C. Thomas. Ddos attack detection using fast entropy approach on flow-based network traffic. *Procedia Computer Science*, 2015.

[24] D. Ding, M. Savi, G. Antichi, and D. Siracusa. Incremental deployment of programmable switches for network-wide heavy-hitter detection. In *IEEE Conference on Network Softwarization (NetSoft)*, 2019.

[25] G. Einziger, O. Eytan, R. Friedman, and B. Manes. Adaptive software cache management. In *19th International Middleware Conference*, 2018.

[26] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 2003.

[27] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms (AofA)*, 2007.

[28] A. Goyal and H. Daumé III. Approximate scalable bounded space sketch for large data nlp. In *EMNLP*, 2011.

[29] R. Harrison, Q. Cai, A. Gupta, and J. Rexford. Network-wide heavy hitter detection with commodity switches. In *ACM SOSR*, 2018.

[30] C. Henke, C. Schmoll, and T. Zseby. Empirical evaluation of hash functions for multipoint measurements. *ACM SIGCOMM Computer Communication Review*, 2008.

[31] 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 *ACM SIGCOMM*, 2017.

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

[33] N. Ivkin, Z. Yu, V. Braverman, and X. Jin. Qpipe: Quantiles sketch fully in the data plane. In *ACM CoNEXT*, 2019.

[34] 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 *ACM SOSP*, 2017.

[35] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In *ACM SOSR*, 2016.

[36] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking*, 1995.

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

[38] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *ACM SIGMETRICS Performance Evaluation Review*, 2004.

[39] Y.-K. Lai, K.-Y. Shih, P.-Y. Huang, H.-P. Lee, Y.-J. Lin, T.-L. Liu, and J. H. Chen. Sketch-based entropy estimation for network traffic analysis using programmable data plane asics. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019.

[40] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. *ACM SIGMETRICS Performance Evaluation Review*, 2006.

[41] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary. Offloading real-time ddos attack detection to programmable data planes. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

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

[43] 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 *ACM SIGCOMM*, 2019.

[44] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *ACM SIGCOMM*, 2016.

[45] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: a novel counter architecture for per-flow measurement. *ACM SIGMETRICS Performance Evaluation Review*, 2008.

[46] J. McCauley, A. Panda, A. Krishnamurthy, and S. Shenker. Thoughts on load distribution and the role of programmable switches. *ACM SIGCOMM Computer Communication Review*, 2019.

[47] 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 *ACM SIGCOMM*, 2017.

[48] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *VLDB*, 2012.

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

[50] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM*, 2015.

[51] P. Roy, A. Khan, and G. Alonso. Augmented sketch: Faster and more accurate stream processing. In *ACM SIGMOD*, 2016.

[52] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the power of flexible packet processing for network resource allocation. In *USENIX NSDI*, 2017.

[53] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy. Approximating fair queueing on reconfigurable switches. In *USENIX NSDI*, 2018.

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

[55] L. Tang, Q. Huang, and P. P. Lee. Spreadsketch: Toward invertible and network-wide detection of superspreaders. In *IEEE INFOCOM*, 2020.

[56] B. Turkovic, F. Kuipers, N. van Adrichem, and K. Langendoen. Fast network congestion detection and avoidance using p4. In *2018 Workshop on Networking for Emerging Applications and Technologies*, 2018.

[57] V. Đukić, S. A. Jyothi, B. Karlaš, M. Owaida, C. Zhang, and A. Singla. Is advance knowledge of flow sizes a plausible assumption? In *USENIX NSDI*, 2019.

[58] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems (TODS)*, 1990.

[59] 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 *ACM SIGCOMM*, 2018.

[60] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li. Pyramid sketch: A sketch framework for frequency estimation of data streams. *VLDB*, 2017.

[61] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *USENIX NSDI*, 2013.

[62] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig. Cold filter: A meta-framework for faster and more accurate stream processing. In *ACM SIGMOD*, 2018.

[63] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 2015.

A EXPECTATION-MAXIMIZATION FOR MULTI-TREE BASED FCM-SKETCH

Along with the conversion algorithm in §4, multi-tree based FCM-Sketch is converted into multi-array virtual counters. Denote $\{V_k\}_{k=1}^{d}$ as d virtual counter arrays from d distinct trees, and $\{V_k^{\xi}\}_{\xi=1}^{D_k}$ each as virtual counters of *degree* ξ in V_k, and D_k is the maximum *degree* in V_k. Let the number of counters in V_k^{ξ} as m_k^{ξ}. At iteration $s+1$ of EM algorithm, the parameters are similarly updated with the case of single-tree based FCM-Sketch (§4.2) where number of size j flows is updated by

$$n_j^{(s+1)} = \frac{1}{d}\sum_{k=1}^{d}\sum_{\xi=1}^{D_k}\sum_{i=1}^{m_k^{\xi}}\sum_{\beta_k^{\xi}\in\Omega(V_{k,i}^{\xi},\xi)} p_{k,i,\xi}^{(s)} * \beta_{k,j}^{\xi} \tag{5}$$

where $\Omega(V_{k,i}^{\xi})$ is the possible set of collisions that can produce $V_{k,i}^{\xi}$, and $p_{k,i,\xi}^{(s)} = p(\beta_k^{\xi}|V_{k,i}^{\xi}, \phi^{(s)}, n^{(s)})$ is computed similarly to Eqn. 2. Note that the processes for different *degree* and virtual arrays can be performed in parallel since they use independent memory units and hash functions.

B PROOF OF ACCURACY ANALYSIS

In this section, we derive a general form of FCM-Sketch's error bound and then prove Theorem 5.1 and 6.1. As defined in §5, we use the size vector $\mathbf{x}$.

Lemma B.1 (Error Bound of FCM-Sketch). *Suppose the virtual counters converted from FCM-Sketch (§4.1) has a finite maximum degree D. Denote the number of leaf nodes of each tree in the FCM-Sketch as $w_1 = \lceil \frac{e}{\epsilon} \rceil$ (e is Euler's number), the number of trees as $d = \lceil \ln \frac{1}{\delta} \rceil$, and the accuracy parameters $\epsilon, \delta > 0$. Given d pairwise independent hash functions, the count-query $\hat{x}_i$ for flow i is bounded by*

$$x_i \le \hat{x}_i \le x_i + \epsilon \max_{1\le\xi\le D}(\xi\|\mathbf{x}\|_1 - w_1\eta_\xi) \tag{6}$$

with probability at least $1-\delta$, where $\|\cdot\|_1$ is 1-norm, θ_j is the maximum counter value at stage j ($2^{b_j}-2$ with b_j-bit), and

$$\eta_\xi = \sum_{j=1}^{\lceil \log_k \xi \rceil} \left(\left\lceil \frac{\xi}{k^{j-1}} \right\rceil - 1\right)\theta_j. \tag{7}$$

B.1 Proof of Lemma B.1

For simplicity, we first prove the singe-tree FCM-Sketch and then extend the result to the multi-tree case.

Clearly, the count-query of i-th flow is sum of the true flow size and error from hash collisions with other flows, i.e., $\hat{x}_i = x_i + E_i$. Our key idea is to decompose the error E_i into (1) the error when using the virtual counter value as the query estimate instead of the original count-query, and (2) the overestimation from the replacement compared to the original query. Formally, if we denote the corresponding virtual counter's value of i-th flow as $\hat{x}_i^V$,

$$E_i = \hat{x}_i - x_i = (\hat{x}_i^V - x_i) - (\hat{x}_i^V - \hat{x}_i). \tag{8}$$

Let us denote m^{ξ} as the number of non-empty virtual counters of *degree* ξ, and w_1^0, w_1 as the number of empty and total leaf nodes at stage 1 of FCM-Sketch, respectively. Also, denote the *degree* of the virtual counter corresponding to i-th flow as $degree(i)$. It is crucial that m^{ξ} and w_1^0 are random variables whereas w_1 is a constant.

Now we introduce two indicators. First, define $I_{i,\xi}$ as an indicator when the arbitrary chosen i-th flow has a virtual counter of *degree* ξ, formally, $I_{i,\xi} = \mathbb{I}\{degree(i) = \xi\}$. By the definition of *degree*, flows are likely to be hashed to a virtual counter proportional to its *degree*. Therefore, given the random variables m^{ξ} and w_1^0, the conditional expectation of indicator $I_{i,\xi}$ is

$$\mathbb{E}[I_{i,\xi}|m^{\xi}, w_1^0] = p(degree(i) = \xi|m^{\xi}, w_1^0) = \frac{\xi m^{\xi}}{\sum_{k=1}^{D} k m^k}$$

where the equality is by uniformity of hash[6]. Moreover, note that $\sum_{k=1}^{D} k m^k = w_1 - w_1^0$. This is because each non-empty virtual counter of *degree* k contains k non-empty leaf nodes, and their total number in virtual counters is always same with the number of non-empty leaf nodes. Lastly, by law of total expectation,

$$\mathbb{E}[I_{i,\xi}] = \mathbb{E}\big[\mathbb{E}[I_{i,\xi}|m^{\xi}, w_1^0]\big] = \mathbb{E}\left[\frac{\xi m^{\xi}}{w_1 - w_1^0}\right].$$

Next, we define an indicator of hash collision $I_{i,j,\xi}$ for different i-th and j-th flows where i-th flow has degree ξ and j-th flow is hashed to one of leaf nodes that will be merged to the virtual counter of i-th flow. Formally,

$$I_{i,j,\xi} = \mathbb{I}\big\{(h(j) \in G(h(i))) \wedge (i \ne j) \wedge (degree(i) = \xi)\big\}$$

where $G(h(i))$ is the set of indices of leaf nodes merged to the virtual counter of i-th flow. Note that the size of $G(h(i))$ is same with its *degree*. By the pairwise independent hashing and definition of *degree*,

$$p\big(h(j) \in G(h(i)), i \ne j \mid degree(i) = \xi, m^{\xi}, w_1^0\big) \le \frac{\xi}{w_1}$$

By law of total expectation,

$$\begin{aligned}\mathbb{E}[I_{i,j,\xi}] &= \mathbb{E}\big[\mathbb{E}[I_{i,j,\xi}|m^{\xi}, w_1^0]\big] \\ &= \mathbb{E}\Big[p\big(h(j)\in G(h(i)), i \ne j, degree(i)=\xi \mid m^{\xi}, w_1^0\big)\Big] \\ &\le \mathbb{E}\Big[p(degree(i)=\xi|m^{\xi}, w_1^0)\cdot\frac{\xi}{w_1}\Big] \le \frac{\xi}{w_1}\mathbb{E}\left[\frac{\xi m^{\xi}}{w_1 - w_1^0}\right]\end{aligned}$$

where the first inequality is by conditional probability. Lastly, the intermediate error is represented as the sum of flows except i-th flow that are hashed into the virtual counter of i-th flow, i.e., $\hat{x}_i^V - x_i = \sum_{\xi=1}^{D}\sum_{j=1}^{n} x_j I_{i,j,\xi}$,

Next, in Eqn. 8, we derive the *lower bound* η_ξ of $\hat{x}_i^V - \hat{x}_i$, the overestimation error of the virtual counter value atop the original count-query. Consider the case where the virtual counter of i-th flow has *degree* ξ. Our key observation is that regardless of the counter value, the smallest possible error of $\hat{x}_i^V - \hat{x}_i$ occurs when its ξ paths meet at the earliest possible stage. Otherwise, the paths except one for i-th flow pass through more number of nodes and they would be merged to the virtual counter. This would result in a higher overestimation error.

Tracking the smallest possible error η_ξ is simple; Accumulate all the counts of ξ paths and subtract the counts for a path of i-th flow.

[6] Pairwise independent hash does imply uniformity.

For example, consider the binary tree and the virtual counter of *degree* 3. Obviously, the earliest possible stage that 3 paths can meet is stage 3. In this case, all nodes that 3 paths pass through are three nodes at stage 1, two nodes at stage 2, and a last node at stage 3. On the other hand, the path of i-th flow goes through each node of stages until the stage 3. After subtracting the counts, $\eta_3 = 2\theta_1 + \theta_2$ where θ_i is the maximum counter value at stage i. Similarly, $\eta_1 = 0$, $\eta_2 = \theta_1$, $\eta_4 = 3\theta_1 + \theta_2$, $\eta_5 = 4\theta_1 + 2\theta_2 + \theta_3$, and so on. We can recursively derive the lower bound η_ξ for arbitrary *degree*, and the general form of η_ξ for *k-ary* trees is written as Eqn. 7. At the end, with combining with the probability of $degree(i) = \xi$, we get the lower bound of the error as $\hat{x}_i^V - \hat{x}_i \geq \sum_{\xi=1}^{D} \eta_\xi I_{i,\xi}$.

Consequently, by linearity of expectation and using the previous results,

$$\begin{aligned}
\mathbb{E}[E_i] &\leq \sum_{\xi=1}^{D}\Big(\sum_{j=1}^{n} x_j \mathbb{E}[I_{i,j,\xi}] - \eta_\xi \mathbb{E}[I_{i,\xi}]\Big) \\
&\leq \sum_{\xi=1}^{D}\Big(\sum_{j=1}^{n} x_j \mathbb{E}\Big[\frac{\xi m^\xi}{w_1 - w_1^0}\Big]\cdot\frac{\xi}{w_1} - \eta_\xi \mathbb{E}\Big[\frac{\xi m^\xi}{w_1 - w_1^0}\Big]\Big) \\
&= \frac{1}{w_1}\sum_{\xi=1}^{D}\Big(\mathbb{E}\Big[\frac{\xi m^\xi}{w_1 - w_1^0}\Big]\cdot\big(\xi\|\mathbf{x}\|_1 - w_1\eta_\xi\big)\Big) \\
&\leq \frac{1}{w_1}\mathbb{E}\Big[\frac{\sum_{\xi=1}^{D}\xi m^\xi}{w_1 - w_1^0}\Big]\cdot\Big(\max_{1\leq\xi\leq D}\big(\xi\|\mathbf{x}\|_1 - w_1\eta_\xi\big)\Big) \\
&= \frac{1}{w_1}\max_{1\leq\xi\leq D}\big(\xi\|\mathbf{x}\|_1 - w_1\eta_\xi\big) \\
&\leq \frac{\epsilon}{e}\cdot\max_{1\leq\xi\leq D}(\xi\|\mathbf{x}\|_1 - w_1\eta_\xi).
\end{aligned}$$

Finally, by applying Markov inequality,

$$p(\hat{x}_i - x_i > \epsilon \max_{1\leq\xi\leq D}(\xi\|\mathbf{x}\|_1 - w_1\eta_\xi)) \leq p(E_i > e\mathbb{E}[E_i]) < e^{-1}.$$

Extension the result to d virtual arrays for multi-tree based FCM-Sketch is trivial because the trees are all independent (similarly with Count-Min sketch [22]). As a result, the last inequality becomes $e^{-d} \leq \delta$, and this completes the proof.

B.2 Proof of Theorem 5.1 and 6.1

In the upper bound of Eqn. 6, the term $\xi\|\mathbf{x}\|_1 - w_1\eta_\xi$ inside the maximum can be written as $\|\mathbf{x}\|_1 + g(\xi)$ where $g(\xi) = (\xi-1)\|\mathbf{x}\|_1 - w_1\eta_\xi$. Note that for $\xi \geq 2$, η_ξ is lower bounded by taking only one term for $j = 1$ in the summation of η_ξ. Formally,

$$\eta_\xi = \sum_{j=1}^{\lceil \log_k \xi \rceil}\Big(\Big\lceil\frac{\xi}{k^{j-1}}\Big\rceil - 1\Big)\theta_j \geq (\xi-1)\theta_1.$$

This inequality obviously holds for $\xi = 1$. Therefore, $g(\xi)$ is always bounded as

$$g(\xi) \leq (\xi-1)\|\mathbf{x}\|_1 - w_1(\xi-1)\theta_1 = (\xi-1)(\|\mathbf{x}_1\| - w_1\theta_1).$$

Consequently, the upper bound in Eqn. 6 is bounded by

$$\max_{1\leq\xi\leq D}(\|\mathbf{x}\|_1 + g(\xi)) \leq \max_{1\leq\xi\leq D}(\|\mathbf{x}\|_1 + (\xi-1)(\|\mathbf{x}\|_1 - w_1\theta_1)).$$

When $w_1\theta_1 \geq \|\mathbf{x}\|_1$, the maximum is obviously $\|\mathbf{x}\|_1$. Otherwise, the maximum becomes $\|\mathbf{x}\|_1 + (D-1)(\|\mathbf{x}\|_1 - w_1\theta_1)$. This results in Eqn. 3.

The proof of Eqn. 4 is straightforward from Theorem 3.1 in ElasticSketch [59]. Denote $x_{h,i}, x_{L,i}$ as the partial counts of x_i kept in Top-K algorithm and sketch, respectively. The key intuition is that the estimator $\hat{x}_i$ is decomposed into values $\hat{x}_{h,i}$ in Top-K algorithm and residuals $\hat{x}_{L,i}$ in FCM-Sketch. As Top-K algorithm counts each flow exactly, $\hat{x}_{h,i} = x_{h,i}$. Also, the estimate in FCM-Sketch, $\hat{x}_{L,i}$, follows its error bound. Then combining the results completes the proof.

C CARDINALITY USING TCAM

In cardinality estimation, a scalability issue of TCAM entries may arise because $\overline{w}_1^0$ can vary from 0 to the number of leaf nodes. To overcome, we can give some space after each entry of $\overline{w}_1^0$ in the tables at a cost of additional error. For query, we lookup the nearest estimate on one side with longest prefix matching. For example, suppose we installed two TCAM entries of estimates for $\overline{w}_0^1 = 1000$ and 1200. When querying for $\overline{w}_0^1 = 1000$, we achieve its exact LC estimate $\hat{n}|_{\overline{w}_1^0=1000}$. On the contrary, if $\overline{w}_0^1$ is between 1001 to 1200, we use $\hat{n}|_{\overline{w}_1^0=1200}$ instead of its correct estimate. To control the additional error, we use the sensitivity of cardinality estimator $\frac{\partial\hat{n}}{\partial\overline{w}_1^0}$ and determine the spaces between entries. In our experiments, this allows us to save the table size by two orders while bounding the additional error by only 0.2%.