Panakos: Chasing the Tails for Multidimensional Data Streams

Fuheng Zhao
UC Santa Barbara
fuheng_zhao@ucsb.edu

Punnal Ismail Khan
UC Santa Barbara
punnalismail@ucsb.edu

Divyakant Agrawal
UC Santa Barbara
agrawal@cs.ucsb.edu

Amr El Abbadi
UC Santa Barbara
elabbadi@cs.ucsb.edu

Arpit Gupta
UC Santa Barbara
arpitgupta@ucsb.edu

Zaoxing Liu
Boston University
zaoxing@bu.edu

ABSTRACT

System operators are often interested in extracting different feature streams from multi-dimensional data streams; and reporting their distributions at regular intervals, including the heavy hitters that contribute to the tail portion of the feature distribution. Satisfying these requirements to increase data rates with limited resources is challenging. This paper presents the design and implementation of Panakos that makes the best use of available resources to report a given feature's distribution accurately, its tail contributors, and other stream statistics (e.g., cardinality, entropy, etc.). Our key idea is to leverage the skewness inherent to most feature streams in the real world. We leverage this skewness by disentangling the feature stream into hot, warm, and cold items based on their feature values. We then use different data structures for tracking objects in each category. Panakos provides solid theoretical guarantees and achieves high performance for various tasks. We have implemented Panakos on both software and hardware and compared Panakos to other state-of-the-art sketches using synthetic and real-world datasets. The experimental results demonstrate that Panakos often achieves one order of magnitude better accuracy than the state-of-the-art solutions for a given memory budget.

PVLDB Reference Format:
Fuheng Zhao, Punnal Ismail Khan, Divyakant Agrawal, Amr El Abbadi, Arpit Gupta, and Zaoxing Liu. Panakos: Chasing the Tails for Multidimensional Data Streams. PVLDB, 16(6): 1291 - 1304, 2023.
doi:10.14778/3583140.3583147

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ZhaoFuheng/Panakos-Chasing-the-Tail.

1 INTRODUCTION

Recent years have witnessed massive growth in Internet-connected applications (e.g., YouTube and Spotify) and devices (e.g., smartwatch, temperature sensors, programmable network switches). The ubiquity of these applications and devices has lowered the threshold for generating data. More concretely, updating an application to log an additional event (e.g., click, skip, etc.) is straightforward. Similarly, updating the packet-processing pipeline for programmable network switches, such as Intel Tofino-based switches [31], to log ingress (or egress) time for each network packet is trivial. The consumers of these streaming data, such as application developers, network operators, and others, are interested in characterizing various interesting events in (near) real-time from these multi-dimensional data streams to drive future decisions. For example, application developers are interested in understanding how the average temperature (aggregated every one minute) is distributed across different agricultural sensors based on various dimensions such as location, device type, etc. Similarly, network operators are interested in the distribution of the number of packets sent (or received) across different hosts or TCP connections every few seconds. In essence, they are interested in understanding how various key statistics, aggregated at regular intervals, are distributed in the multidimensional space.

These tasks often entail moving the data from a source (e.g., a browser, a sensor network switch) to a centralized collector and then using one-pass streaming algorithms to compute the aggregate statistics and then reporting the distribution of these statistics for different windows. Ideally, these tasks should be performed closer to the source in bandwidth-constrained operational settings to avoid the unnecessary movement of data, which can be prohibitively expensive. For example, moving packets to a remote collector is prohibitively costly even for high-speed networks, handling hundreds of Gbps data rates [29, 47]. Similar trends are also observed in sensor networks (e.g., Joltik [61]), where data transmission is power-hungry, and the frequent replacement of batteries is costly. However, compared to well-provisioned centralized collectors, data sources (e.g., switches, sensors) have limited computation and memory resources. Most existing streaming algorithms can *either* accurately calculate these aggregate statistics *or* report their distributions while making the best use of limited resources. However, we are not aware of any existing work that do *both*.

This paper explores the design of a *cost-effective* data-processing pipeline that processes the multidimensional data stream, and *accurately* reports the feature distribution based on the metric values and focuses on cumulative aggregation such as the count or frequency associated with each feature. We aim to not only accurately report the quantile summary of the distribution but also report the contributors to the distribution's tail such as items with frequencies in the higher quantiles (e.g., 99^{th} percentile). Two observations inspire our focus on the tail: (1) real-world data are intrinsically heavy-tailed, and (2) identifying tail contributors is critical for many data consumers. For example, network operators must identify victims of DDoS attacks to decide on mitigating actions [2].

Proceedings of the VLDB Endowment, Vol. 16, No. 6 ISSN 2150-8097.
doi:10.14778/3583140.3583147

Addressing these challenges and designing such a data-processing pipeline is challenging. None of the existing solutions are suited to the problem at hand. However, given that different existing works are good at satisfying some of the requirements, system operators end up using different algorithms/pipelines for different tasks. For example, one to accurately capture the distribution, and another to accurately report the frequent items. However, fine-tuning unrelated data sketches that share the same pool of constrained resources is non-trivial [40, 62]. Moreover, given the inherent challenges in identifying tails using relative quantiles, system operators end up using a static threshold to identify heavy hitters. However, as previous work [3] has shown, static thresholds are unresponsive to distribution shifts, and their usage affects the ability to report tail contributors accurately.

In this paper, we present the design and implementation of Panakos that makes the best use of limited available resources to summarize the feature distribution and identify tail contributors. Panakos leverages the skewness inherent to most high-dimensional feature streams in the real world to disentangle the feature stream into cold, warm, and hot features based on their metric values and uses a combination of data structures to track them. To the best of our knowledge, Panakos is the first data sketch to accurately report *both* the quantile approximation of a feature distribution and the identity tail contributors, i.e., contributors to the tail portions of the feature distribution *in resource constrained-environments*. Prior approaches either address one of the problems or are inefficient since cold and hot features are mixed. Our main contributions can be summarized as follows:

- Design of a data-processing sketch, Panakos, that accurately captures the frequency distribution of a data stream and reports contributors to its tail in resource-constrained settings.
- Provide mathematical analysis of Panakos and discuss the trade-offs between resources and accuracy.
- Implementation of Panakos for both general-purpose CPUs and programmable switches/NICs.
- Comparison of the proposed system with existing solutions using both synthetic and production datasets.

The paper is organized as follows. Section 2 provides the background and motivation. Section 3 introduces our proposed solution, Panakos, discusses the rationale behind each component, and gives intuitions on the improved performance by integrating these components. Section 4 presents the mathematical analysis of Panakos. Section 5 discusses the implementations of Panakos on different platforms. Section 6 reports the experimental evaluations conducted using synthetic and real-world datasets and compares Panakos to state-of-the-art sketches. Finally, Section 7 summarizes our contributions and concludes this work.

2 BACKGROUND AND MOTIVATION

2.1 Target Applications

Panakos provides application developers, system analysts, and network operators with an end-to-end analytic engine capable of understanding different features over time. As examples of the types of workloads and compute environments we seek to support, we draw on two motivating use cases.

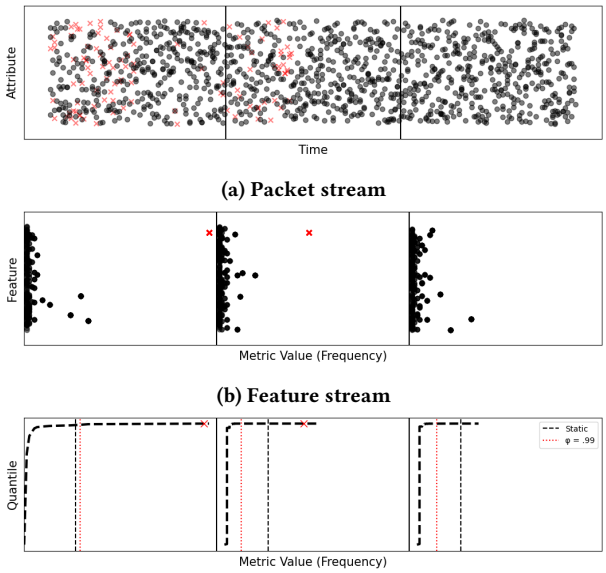


Figure 1: (a) Raw packet stream; (b) Extract Feature stream using the counts of source IP; (c) Feature distributions. Red cross(es) correspond to packets with source IP `92.160.2.154`**.**

Sensor data analytics. Wireless sensors are increasingly deployed across cities or rural areas to collect various metrics of interest [39, 61]. Due to energy, storage, and communication constraints, wireless sensor devices can only occasionally transfer small-sized summaries to end hosts over some period of time. Panakos can be used in such an environment to collect various statistics for analysis and satisfy the constraints in the low-power environment. In Section 6, we demonstrate Panakos's high performance over real-world sensor datasets.

Network monitoring. Network operators are interested in defending their networks against a wide range of cyberattacks [22, 41] while also optimizing the quality of service. They rely on streaming analytic systems to monitor the network's state [17, 69]. However, developing systems that can support network analytic queries at high data rates (e.g., 100 Gbps) is challenging. The rise of programmable switches [31] offers new opportunities to scale network streaming analytics. Modern network streaming analytic systems [29, 40, 54, 63] use network hardware to scale query execution. We consider deploying Panakos in such an environment. We aim to make the best use of available data-plane resources to offload most processing tasks to the programmable switch.

2.2 Problem Description

High-dimensional data streams. We consider multi-dimensional data streams for analysis from diverse resource-constrained sources. For the network streaming analytics [29, 40, 47], the data stream is the raw packets in the network's data plane, equipped with limited memory and compute resources [31]. For a data stream (σ), each data point contains a set of *attributes* (A) and *metrics* (M). Here, the set of *attributes* corresponds to associated identifiers and

Table 1: Panakos v.s. other solutions.

	Solutions	Feature Extraction	Feature Distribution	Tail Contributors	Hardware Friendly	Remark
(1)	KLL [32, 34, 66], DCS [48, 57, 67]	❍	●	◗	❍	Quantile Sketches require statistics not exist in the raw stream.
(2)	SpaceSaving [45, 56, 65], ASketch [52], CocoSketch [64], Count-Min [14]	●	❍	◗	●	Frequency Sketches only extract feature and metrics from raw stream
(3)	MRAC [37], Elastic [62], FCM [55]	◗	◗	❍	●	Synthesis focus on the general pattern and overlook the tail
(4)	Panakos (ours)	●	●	●	●	Navigate through the design space to satisfy all requirements

metadata, and *metrics* corresponds to key measurements. For the network packet stream, the set of attributes typically includes the packet's header fields (e.g., `sIP`, `dIP`, etc.) and any additional meta-information associated with the packet such as next-hop, virtual LAN (VLAN) identifier, application identifier, etc. The set of metrics includes various measurements associated with each packet, such as the number of appearances, sojourn time, etc.

Feature streams. Typically, a given application is interested in analyzing the characteristic of a subset of attributes of the data stream. We refer to this subset as *features*. Next, we consider the transformation of the raw (attribute, metric) data stream, i.e., σ: (A, M), into a feature stream, i.e., σ_π: (A', $\pi(M)$, w). We derive this feature stream by applying function π over metrics (M) for all elements with the same set of attributes, A', where $A' \subset A$, within a window w. For the network streaming analytics case, the feature stream can be the total number of bytes associated with each source IP address (`sIP`) in a one-millisecond window. The feature extraction function π is `sum`, and the set of features (A') is `sIP`, which is a subset of attributes (A) in the raw packet stream. Given the complexity of executing an arbitrary feature extraction function (π) on resource-constrained data sources, we focus on the `Count` operator and then discuss Panakos' generalization to `Sum` and `Average` operators. We leave the extension of Panakos to support more complex operators such as `Median` for future work.

Goal. Given this setup, our goal is to characterize the distribution of some subset of attributes in each window. More specifically, for each window, we aim to accurately report the feature quantile distribution where the features are a set of attributes (e.g., `sIP`). We also aim to report the anomalous data points for each window, i.e., ones that contribute to the tail portion of the feature distribution. We specify these anomalous data points as ones that exceed a pre-specified threshold, ψ. Note that the feature distribution changes over time; thus, we specify tail features using each window's relative threshold ψ. For instance, if $\psi = .99$, we report the features whose metric value is greater than the 99^{th} percentile value. Figure 1a illustrates an example of a high dimensional data stream from wide-area network raw packets over three milliseconds. This stream has a source IP address (`92.160.2.154`) with anomalously high packets in the first two milliseconds. The x-axis is time, and the y-axis is the hash values of the high dimensional attributes values. In Figure 1b, the raw stream is transformed into a source IP feature stream by applying the *Count* operator. The x-axis becomes the metric values, and the y-axis becomes the source IP feature. Figure 1c depicts the feature distribution in which the x-axis is the metric value and the y-axis is the quantile value. The dot and dash lines show the difference between using a static threshold and identifying the tail contributors based on $\psi = .99$, in which the static threshold can not adapt to the change in distribution.

2.3 Limitations of Existing Solutions

The goal of our data processing pipeline is to (1) extract desired features from the raw data stream and then, for each window, (2) accurately report the quantile approximation of the feature distribution and (3) identify anomalous data points, i.e., contributors to the tail portions of the feature distribution in (4) resource constrained-environments. We are not aware of any existing solution that satisfies all these four requirements to the best of our knowledge. Table 1 divides existing work into three categories and illustrates how they do not satisfy one of the four requirements.

Quantile sketches (#1). Quantile sketches [24, 27, 34, 43, 53, 57, 66] focus on making the best use of available (limited) memory to report quantile summaries of a data stream. At first glance, one may think quantile sketches would be strong candidates to address these challenges. Since features are drawn from a high-dimensional universe, the ordering among these features is not well defined. In addition, quantile sketches cannot be used to extract features (like aggregates based on counts). A naive solution would be to store and compute the true statistics, e.g., storing all items and their frequencies in a large hash table and then feeding this information into quantile sketches to obtain the feature distribution. Storing and computing the true statistics, e.g., finding the exact frequency of each item, would require significant memory space. Moreover, the efficient implementation of quantile sketches in resource-limited hardware remains an open problem [33].

Frequency estimation sketches (#2). Frequency estimation and frequent items sketches, including counter-based summaries [19, 35, 45, 46, 52, 65] and linear sketches [10, 14] extract important features by accurately estimating each feature's frequency and identifying heavy hitters. However, frequency estimation sketches alone cannot accurately approximate the entire feature distribution. The main reason is that these sketches ignore or underestimate the frequency of infrequent items, and infrequent items are necessary

for understanding the entire feature distribution. Frequency estimation sketches can be implemented efficiently on resource-constraint hardware [54, 64] after solving many challenges.

Synthesis (#3). The stream synthesis approach attempts to understand feature distributions using statistical tools. On the one hand, previous work [42] has shown that by sampling features from a stream and using the maximum likelihood estimation algorithm to analyze the samples, the tail behavior can be directly understood. However, these approaches overlook the overall feature distribution and are not data-driven since they require *a priori* knowledge of that data packet distribution (e.g., Zipfian Distribution). Alternatively, [37, 62] have shown that by creating a small sketch of the stream and then using the expected maximization algorithm to parse the sketch, it is possible to capture the entire feature distribution accurately. However, these approaches are inefficient for identifying the tail contributors, as the hot features are mixed with the cold features. While the expected maximization algorithm does not satisfy all our requirements, it helps synthesize the general trend of the feature distribution. Later, we will show how we leverage this algorithm to design Panakos.

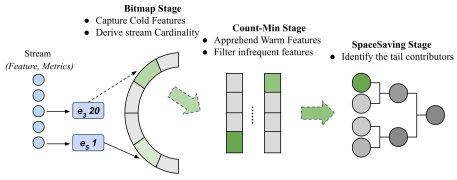
2.4 Observations and Opportunities.

Feature streams are heavy-tailed. We now explore the properties of feature streams we aim to characterize with Panakos. Figure 1 (c) shows the heavy-tailed behavior exhibited by a feature stream in each one-millisecond window. Specifically, we observe that the metric, i.e., the number of packets for each source IP address, is small for *most* IPs. However, a *few* features exchange a very high number of packets. We also observe similar trends in sensor analytics, and previous works have reported the heavy-tailed behavior for a wide range of real-world applications [15].

Disentangle cold and hot items. Given the challenges associated with the accurate estimation of heavy-tailed streams, separating cold and hot items in the skewed distribution is desirable to estimate infrequent and frequent features accurately. Previous work proposed disentangling cold and hot items to better assess the size of an equi-join [25]. Given the skewness in many real-world distributions, we have recently witnessed the development of several algorithms/systems that leverage the same intuition to strike a better balance among accuracy, update latency, and resource usage [52, 62, 68].

We observe that it is possible to leverage existing frequency sketches to summarize cold and hot features separately. In particular, we can use linear sketches, such as Count-Min [14], on cold and warm features, and use counter-based summaries, such as SpaceSaving [45], on hot features. While counter-based summaries use optimal space to store information about the hot features, they do not provide any insights into the cold features, as they always evict the feature with minimum frequency to make room for new arriving data. On the other hand, while linear sketches provide insights into all items in the universe using d hash functions to map features into d counters, they need more space to deliver similar performance compared to the counter-based algorithms. Also, linear sketches may fail with small probability, and cannot accurately identify the hot features due to hash collisions between cold and warm features with hot features. Thus, we conclude that linear sketches are more suitable for tracking the numerous cold and warm features with the additional advantage of using minimal memory accesses and requiring no comparisons (comparisons are often expensive in hardware). After filtering out the many cold and warm features using linear sketches, counter-based approaches can directly identify the tail contributors with a few number of comparisons (since cold items are filtered out) while using optimal space.

3 PANAKOS



(a)

Figure 2: Panakos Architecture

3.1 Rationale

There are two main goals that Panakos aims to deliver: 1) summarize the feature distribution, and 2) identify tail contributors based on quantile threshold ψ. We start by observing the connection between Top-K and the tail contributors. Initially, if the cardinality, which is the number of unique features in the windowed stream, $card$, is known, then, the Top-K items are the tail contributors above the relative threshold of $\psi = 1 - \frac{K}{card}$. In Section 2.3, we identified that SpaceSaving [45] is ideal for identifying the heavy tail contributors. However, the information missing from SpaceSaving are, namely, the data stream cardinality, $card$, and the cold and warm features, which are needed to construct the entire feature distribution. Therefore, additional data structures and algorithms are needed to extract this information. The Linear Counting algorithm [59] is commonly used to estimate the cardinality of a windowed stream, as it uses small space and allows fast processing. In Section 3.4.2, we show how Linear Counting can estimate singleton (features that only appear once) features by using 2 bits per counter. In addition, real-world data streams are often skewed where singletons and doubletons are a large fraction of all features. While singletons and doubletons can be identified using just 2 bits, warm features require more bits per counter to store their metric value. For a fixed memory budget, increasing the number of bits per counter will decrease the number of vector entries. Thus, we decided to use the Count-Min sketch, which consists of fewer entries but more bits per counter compared to a single array. Count-Min can also 1) obtain information on warm features, and 2) filter out cold and warm features. Although Count-Min has fewer entries in each counter vector, it uses multiple independent hash functions and counter vectors to reduce the probability of hash collisions and hence deliver satisfying results.

3.2 Basic Panakos Data Structures

As shown in Figure 2, Panakos consists of three components to disentangle cold, warm and hot features and to provide more accurate

Algorithm 1: Panakos Stream Processing Algorithm

```
for item from stream do
    e = extract(item);
    V = bitsVector[hash(e)];
    if V < 2 then
        bitsVector[hash(e)] += 1;
        return ;
    else if V == 2 then
        bitsVector[hash(e)] += 1;
    V = 2 + CM.query(e) ;
    if V < 1 + 2^T then
        CM.insert(e) ;
        return ;
    SpaceSaving.insert(e);
    return ;
end
```

estimates of the feature distribution, tail contributors, and other statistics. While each component is used for different purposes, the components are integrated coherently. The bitmap stage is a vector of 2-bit counters, followed by the Count-Min stage, and finally the SpaceSaving stage.

Bitmap Stage. The first bitmap stage is responsible for tracking the number of singletons (n_1), doubletons (n_2), and the cardinality of the current window stream. The bitmap uses 2 bits per counter entry for the following reasons: (1) We first note that one bit is not enough to differentiate a singleton feature from features with more than one occurrence. Using 1 bit, if a counter is flipped to 1, it is not clear if the counter is flipped because it encountered a singleton or otherwise. We need at least 2 bits, to be certain if a counter has value 1 then it must be a singleton which caused the counter to increment from 0 to 1. (2) Since 2-bit counters are allocated, we can use the available bits to maintain information not only on the number of singletons, n_1, but also the number of doubletons, n_2. (3) When a counter in the bitmap entry is 3, this indicates that the counter is overflowed. The reason that we need this overflow state is to avoid overestimating an item's frequency by too much, as without an overflow state, one can not be certain if the item proceeded to the next stage (i.e, a Count-Min Sketch) or not. Even if a feature is never inserted into Count-Min, Count-Min may largely overestimate the count due to hash collisions.

Count-Min Stage. Warm features are stored in the Count-Min with T bits counters and the maximum counter value is $2^T - 1$. Count-Min is an ideal candidate to track warm features from n_3 to n_{2+2^T}, as Count-Min projects the features into small counter vectors, and by using multiple vectors, it obtains more accurate estimates of the metric values. In addition to small space, Count-Min has two important properties, namely supporting high update throughput and causing no underestimation, which helps with processing fast streams and understanding tail behaviors [13]. Furthermore, Count-Min does not need to keep an overflow state as the next stage, SpaceSaving, stores each feature's identity and guarantees that if a feature is never inserted, its count would be zero.

SpaceSaving Stage. The final stage is a SpaceSaving summary to track all the heavy features that pass through the previous two stages. SpaceSaving stores both the feature's identity and estimated frequency while using optimal minimal space. Since each feature's identity is directly known, we can then be certain that these features lie in the heavy tail. To target specific hardware such as programmable switches, we use CocoSketch [64], an extension of SpaceSaving optimized for computation and memory accesses on special hardware.

3.3 Panakos Update Workflow

Initially, Panakos initializes all counter values to zero and the desired feature dimensions are defined by the user. For instance, in network monitoring applications, a representative feature can be defined as a 5-tuple, i.e, source IP address/port number, destination IP address/port number, and the protocol. Panakos digests arriving data packets in three logical steps, as shown in Algorithm 1:

Step 1: Extract the feature of interest, e, from each arriving data item, and use a uniform hash function h to map e to $h(e)$, an index in the 2-bit bitmap. If the counter value stored at index $h(e)$ equals 3, which is already in an overflow state, then forward e to the next step. If the counter value stored at index $h(e)$ equals 2, then the counter is incremented to 3 and e is forwarded to the next step. Otherwise, increment and return.

Step 2: Update Count-Min with the forwarded feature e. Count-Min maps feature e using d hash functions and increment the corresponding d counters. If a counter value is $2^T - 1$, which is the maximum possible value, then the increment is ignored. If all d counters are $2^T - 1$, then propagate the update to the next step.

Step 3: Update SpaceSaving using feature e. If e is monitored in the SpaceSaving structure, then increment the counter associated with the monitored key e. However, if e is not monitored, SpaceSaving replaces the item with the minimum count by e. This replacement operation uses $O(\log k)$ number of comparisons in a standard min-heap implementation of SpaceSaving where k is the size of SpaceSaving. In addition, SpaceSaving incurs large estimation errors when hot items are replaced by cold items. Panakos improves SpaceSaving's estimates because, with the filtering stages in Steps 1 and 2, cold items rarely reach SpaceSaving, hence reducing the update latency and enhancing estimation accuracy.

3.4 Feature Distribution

In this section we discuss how Panakos derives the feature distribution of a data stream. In particular, how Panakos provides an approximate quantile approximation on the distribution of feature metric values. We start by estimating the overall cardinality and the number of singletons and then develop the necessary machinery to estimate the cumulative quantile value of larger metric values.

3.4.1 Cardinality Calculation. Panakos' 2-bit bitmap vectors can provide accurate estimates of singletons and approximate the cardinality of the feature stream. To approximate the cardinality of the feature stream, *card*, we use the popular Linear Counting algorithm [59]. It has been shown in previous studies that Linear Counting can provide accurate estimates with around 1% standard error using less than $12 \cdot card$ bits while achieving fast processing time [30, 44, 63]. The linear counting algorithm maintains a vector

of length m and uses a uniformly random hash function to map a feature into an index in the vector. After processing the data stream, the probability that an entry in the vector equals 0 is $(1-1/m)^{card}$. Hence, the estimator for $card$ is $\widehat{card} = m \cdot ln(\frac{m}{m_0})$ where m_0 is the number of zero entries in the vector.

Panakos Optimizations. $\widehat{card}$ is an estimate of $card$. Since high-frequency features are directly stored in Panakos, to improve the cardinality estimation, we calculate the cardinality for cold features ($card_{cold}$) and hot features ($card_{hot}$) separately. $card_{hot}$ is heuristically estimated as the number of frequent features in the SpaceSaving data structure. $card_{cold}$ is estimated by resetting all vector entries associated with these large features to zero and using the linear counting algorithm to estimate the number of cold features in both the 2-bit bitmap vector and the count-min data structures. Hence, the final cardinality is obtained by summing $card_{cold}$ and $card_{hot}$.

3.4.2 Singletons Estimation. Estimating the number of singletons is an interesting statistic for cold features. In real-world data streams, a large fraction of the features only appear once [23] and estimating the number of singletons is challenging. In previous work [9], the number of singletons is approximated by sampling. However, this approach requires the sampling rate to be around 1% to provide an accurate approximation, which can cause significant overheads as features often need several bits to represent. We adopt Kumar et al. [37] approach, which was inspired by Linear Count, and provides an accurate estimator for n_1 using a vector of constant length m and a uniformly random hash function. After incrementing all vector entries based on the input stream, the number of singletons can be estimated directly based on the estimated cardinality, $\widehat{card}$, and the number of entries with value one, denoted as m_1, in which $\widehat{n_1} = m_1 e^{\widehat{card}/m}$.

3.4.3 Challenges with $n_{i\geq 2}$. While singletons can be directly estimated, doubletons and beyond are harder to estimate. If a vector entry has a value of 1, then it must be a singleton feature mapped into this entry. However, if a vector entry has a value of two, then there are two cases: 1) it is a result of a doubleton, or 2) two singletons hashed into the same location by chance. Moreover, for warm features, the counter values can span a wide range, and with larger values, there can be more erroneous instances due to hash collisions. Quantiles of feature distribution are cumulative, and hence errors are accumulated resulting in the potential for large estimation errors. To tackle this challenge, we use the maximum likelihood estimation described in Section 3.4.4 to derive information about n_2 and beyond.

3.4.4 Expectation Maximization. While Section 3.4.2 shows n_1 can be directly approximated, this method can not be generalized to estimate n_i for $i > 1$. The expectation maximization (EM) algorithm [20] is an iterative algorithm that derives the maximum likelihood estimation of the feature distribution. EM has been widely applied in many areas [18, 21, 37], and is known for its convergence to an empirically good estimator [60]. We use the EM algorithm to estimate the distribution of cold features stored in the 2-bit bitmap and the Count-Min sketch of T bits. The 2-bit bitmap provides information on n_1 and n_2, and the Count-Min sketch provides information on n_3 to n_{1+2^T}. Notice, with a limited space budget, different features may hash to the same entry in the bit vector and Count-Min, and they only provide a partial view of the feature distribution. EM can then maximize the likelihood of the feature distribution with respect to the unobserved hash collisions. For hot features with large frequencies above or equal to n_{1+2^T}, this information is contained in SpaceSaving.

The EM algorithm is executed in iterations of expectation and maximization steps. The expectation step of a given iteration uses the feature distribution from the previous iteration to guess the expectation of the log-likelihood of hash collisions. Note since hashing is uniform, the collision pattern can be approximated using Poisson distributions [7]. The maximization step computes the most likely new feature distribution to maximize the expectation of the log-likelihood of hash collisions. The output of the maximization step also serves as an input to the expectation step in the next iteration. EM repeats over a fixed number of iterations such as 10 iterations to derive an accurate approximation. In each iteration i, the EM algorithm takes in a pair of n^{i-1}, ϕ^{i-1} and outputs a new pair of n^i, ϕ^i. The approximate feature distribution outputted by iteration i consists of $n^i = \{n_1^i, n_2^i, \ldots\}$ and ϕ^i which is the feature distribution. The initial n^0 and ϕ^0 can be derived directly assuming there are no hash collisions in the observed counter values.

More details of the correctness of the EM algorithm are shown in [37] [1]. In addition, the computational complexity of running the EM algorithm is high. For an observation with a counter value of j, the time complexity of EM is $O(j^3)$. In practice, researchers heuristically reduce the complexity by limiting the number of different features hashing into the same entry to 7. Hence, EM often ignores hash collisions for large counter values. In Panakos, by disentangling cold and hot items, the computation complexity is reduced as the tail contributors are tracked solely based on SpaceSaving.

3.4.5 Panakos Optimizations for Feature Distribution. In Panakos, we first derive the accurate cardinality of cold features, $card_{cold}$, which reflects the cardinality of features stored in the 2-bit bitmap and Count-Min, as described in 3.4.2. Since the 2-bits vector is a single array, the 2-bits vector can be directly used as an input to the EM algorithm to obtain the number of doubletons (n_2). We can then subtract n_1 and n_2 to obtain the number of unique features captured by Count-Min. In addition, since Count-Min usually contains multiple rows, we need to transform Count-Min into a single vector as the input to the EM algorithm which requires a single vector. The transformation takes the average of the number of observations for each counter value except for the minimum and maximum counter values. The average of all the counter-value observations is taken to minimize the variance in the hash function randomness and obtain the most likely initial guess for the feature distribution. However, for the minimum and maximum counter values, an alternative approach is taken based on the following observation: Consider a Count-Min with two counter vectors that use two hash functions to map each feature into the corresponding vector, for example, the two counter vectors are $[1, 1, 3, 2]$ and $[1, 2, 2, 2]$. Let's first consider the minimum metric value. From the first counter vector, we can infer that there are at least two distinct features with a metric value of one, whereas the second vector implies at least one feature with a metric value of one. Hence for the minimum metric value, we

[1] The EM algorithm is described in Section 4.2 in [37].

should rely on the maximum number of appearances. Next, let's consider the maximum metric value. While the first vector tells us that there is a feature that might have appeared 3 times, based on the second vector we know that cannot be the case. If there exists a feature that appeared 3 times, then the second vector must have an entry with a counter value larger than or equal to three. Hence for the maximum metric values, we should rely on the minimum of the maximum metric values in all vectors.

3.5 Frequency Estimation

Algorithm 2: Panakos Frequency Estimation for e

```
1 V = bitsVector[hash(e)];
2 if V < 3 then
3 |   return V ;
4 V = 2 + CM.query(e) ;
5 if V < 1 + 2^T then
6 |   return V;
7 return 1 + 2^T + SpaceSaving.query(e);
```

Unlike the cumulative feature distribution, frequency estimation is point-wise and aims to estimate the frequency of a feature e. Hence, errors in frequency estimation are not cumulative and the point-wise error is often small. Panakos leverages the separation of cold items and hot items to deliver better estimation, in which the frequency of the cold items is captured by the bit vector and Count-Min, and the frequency of hot items is captured by SpaceSaving. The query operation on e involves three steps, as shown in Algorithm 2:

Step 1: Use a uniform hash function h to map e to $h(e)$, an index in the bit vector. If the counter value stored at index $h(e)$ is less than 3 (3 indicates the counter is overflowed), then report the counter value as the estimated frequency. Otherwise, forward the query to the next step.

Step 2: Query the Count-Min sketch with e. If the sum of 2 (due to the two initial hashes to the bit map) and the query result is less than $1 + 2^T$, report the sum as the estimated frequency.

Step 3: Query the SpaceSaving with e, and report the sum of $1 + 2^T$ (due to prior hashes to the bitmap and the Count-Min sketch) and the query result as the estimated frequency.

3.6 Identifying Tail Contributors

Instead of relying on a static threshold, such as in the Top-K problem, the ψ heavy tail report important features based on a relative quantile threshold ψ, in which setting $\psi = .99$ means to find the top 1%. Tail contributors can be efficiently identified by consolidating feature distribution and frequency approximations. First, the given quantile threshold ψ is mapped into a frequency threshold, $\mathscr{F}$, based on the approximated feature distribution. If the frequency threshold, $\mathscr{F}$, is larger than $1 + 2^T$, then all heavy tail features are stored in SpaceSaving. Features stored in SpaceSaving with an estimated frequency larger than $\mathscr{F}$ are the heavy tail features. On the other hand, if the frequency threshold, $\mathscr{F}$, is less than $1 + 2^T$, the heavy tail features consist of all features stored in the SpaceSaving and some additional features from the Count-Min Sketch.

3.7 From Count to Sum and Average

In this subsection, we discuss how Panakos can be further generalized to support *Sum* and *Average* operators. Streams may contain weighted metrics such as purchase quantity or importance [11]. To handle a feature with weight w, we can treat the weighted insertions as w inserts of a single feature. Although this approach is correct, it would undesirably increase the update times. A more efficient approach is to process the weights in a single pass. Fortunately, the data structures used in Panakos are already known to support the *Sum* of weighted inserts [14, 45, 56, 64]. As a result, for weight metrics, Panakos maintains the original logic, and if the weights exceed the counter capacity in the current stage, then the remaining weights can be incrementally applied to subsequent stages.

Supporting the *Average* statistic is challenging. Although the sum and count metrics have tight additive-error guarantees, their division can still incur large errors. To the best of our knowledge, we are unaware of any existing data sketches supporting the average operator. To support the *Average* operator, we propose two heuristic approaches i) data-independent Ratio Distribution, and ii) data-dependent Panakos with Alignment. First, we introduce the data-independent Ratio Distribution approach. In essence, the average of an item is its sum divided by its count. Assume the count and sum of any item are two random variables drawn from two independent distributions, i.e, ϕ_{count} and ϕ_{sum}, then we can compute the density distribution of the average metric, a, as $\phi_{avg}(a) = \sum_{c=1}^{\infty} c\phi_{count}(c)\phi_{sum}(c \cdot a)$, where a is the average metric, c is the count metric, and $c \cdot a$ is the sum metric. Now, we can use two independent Panakos sketches to approximate both ϕ_{count} and ϕ_{sum}. By approximating the distributions and assuming these two metrics are independent, we obtain the approximated distribution for the average.

Our second data-dependent method, Panakos with Alignment, uses the sum metric to guide the placement of the count metric. On one hand, Panakos with Alignment includes two bitmap and Count-Min stages corresponding to the sum and count operators, and these data structures share the same set of hash functions (to ensure a feature's count and weights share the same counter index). On the other hand, Panakos with Alignment uses one single SpaceSaving stage with one extra field that maintains (feature, weight, count) tuples. In addition, the SpaceSaving update algorithm takes (feature, weight, $c = 1$) as parameters. Note the count of a new insertion is always one, whereas the weight may vary. The update logic for SpaceSaving follows the same procedure as described in Section 3.3 Step 3, and the only additional step is, depending on which tuple is modified due to the inserted weight, its count field which is also increased by one. From the weights of the insertions, we can classify this insertion as cold, warm, or hot. Based on the cold, warm, or hot classification, we aim to place the count in the bitmap, Count-Min, or SpaceSaving stage, and, in case the counter reaches its capacity, the count will be stored in the next stage. The intuition is that to compute accurate average information, we need to place the count of the feature corresponding to its weights such that each feature's weight and count are near each other. Empirically, we find the alignment strategy produces accurate approximations.

4 PANAKOS ANALYSIS

In this section, we analyze Panakos's theoretical guarantees for understanding the general feature distribution of a data stream and identifying the tail contributors.

4.1 Panakos Feature Distribution Properties

We argue that Panakos provides more accurate feature distributions when using larger space. Panakos relies on the EM algorithm to derive the feature distribution of a data stream. The EM algorithm is well-studied as a *maximum likelihood estimator*. Asymptotically, the EM algorithm provides an unbiased estimation of the feature distribution [4]. Since the derived feature distribution from the EM algorithm is unbiased, we can analyze Panakos based on the estimation variance. For a fixed number of observations, the minimum variance of a maximum likelihood estimator is bounded by the well-known Cramer-Rao bound [16, 49], which states that the variance of any maximum likelihood estimator is lower bounded by the inverse of the Fisher information. The Fisher information [51] can be seen as the amount of information that a set of observable data points (the counter values) stores about the unknown distribution (feature distribution). By increasing Panakos' space budget, Panakos can thus store more observations. As a result, the Fisher information will increase, which is then translated into a finer lower bound for the variance. Hence increasing the space budget for Panakos improves the estimated feature distribution that Panakos provides.

Moreover, the feature distribution satisfies two constraints, namely $\sum_{i\geq 1}\hat{\phi}_i = 1$ and $\forall i, 0 < \hat{\phi}_i < 1$. It has been shown in [26] that constraints on the estimated distribution increase the Fisher information. The Fisher information can be calculated from the observations seen in the sketch and hence with a larger sketch size and more observations, the Fisher information will increase [42, 50, 58]. Hence, the additional constraints on the feature distribution increase the available information, thus reducing the lower bound on the variance, and ensuring more accurate estimates of the feature distribution given the same amount of space.

4.2 Analysis of Tail Contributors

To identify the tail contributors given the relative threshold ψ, the procedure discussed in Section 3.6 consists of two steps: 1) use the feature distribution to map a quantile threshold, ψ, into a frequency threshold $\mathscr{F}$, and 2) report all features with frequencies larger than or equal to $\mathscr{F}$. In Section 4.1, we argued that Panakos's feature distribution is unbiased and minimizes the variance when using more space. In this section, we focus on the second step and provide an analysis of Panakos' estimate of a feature's frequency.

Estimating the Number of False Positives. We observe that filtering out small and warm features can improve the performance of Count-Min and SpaceSaving as their estimation error depends on the number of features they digest. The bitmap of 2-bit counters and Count-Min of T-bit sketch can be seen as a counting Bloom filter [12] whose function is to filter out the small and warm features. The bitmap component is a Bloom filter that uses one hash function and the Count-Min component is equivalent to a Bloom filter using $O(log\frac{1}{\delta})$ hash functions where δ is the probability of Count-Min having a high estimation error. To analyze the false positive rate of reporting cold features as hot features, we assume the features in the stream are ordered randomly [28]. Under the random order assumption, a counting Bloom filter with m entries, H hash functions, and after digesting a windowed stream with N features, the false positive rate of classifying features with metric values less than $1 + 2^T$ as hot features is approximate:

$$FPR(threshold, H, N, m) = (1 - \sum_{l \leq threshold} b(l, HN, 1/m))^H \quad (1)$$

where $b(l, HN, 1/m) = \binom{HN}{l}\frac{1}{m}^l(1-\frac{1}{m})^{HN-l}$ is the binomial distribution [12, 36].

Based on Equation 1, the 2-bit bitmap with one hash function and m_{bm} counter entries has a false positive rate (denoted as BM_{fpr}) of not classifying features with a metric value less or equal to two as small features is:

$$BM_{fpr} = 1 - \sum_{i=1}^{2} b(i, N, 1/m_{bm}) \quad (2)$$

Hence, the approximate number of features that proceed from the first stage to the second stage can be approximated as:

$$N_{MedHvy} = BM_{fpr} \cdot (n_1 + 2n_2) + \sum_{i\geq 3} i \cdot n_i - sum(BM) \quad (3)$$

where some singletons and doubletons may encounter false positive cases and $sum(BM)$ is the number of features remaining in the 2-bit bitmap.

Similarly, we can calculate the false positive rate of Count-Min, with H hash functions, m_{cm} counters, and a metric threshold of $2^T - 1$ using T bits per counter, as

$$CM_{fpr} = (1 - \sum_{i=1}^{2^T-1} b(i, HN_{MedHvy}, 1/m_{cm})^H. \quad (4)$$

Let $sum(CM_{row})$ be the sum of counter values in one of the Count-Min vectors (all vectors in Count-Min have the same sum), and then let $N_{remain} = sum(BM) + sum(CM_{row})$ where N_{remain} represents the number of features remaining in both the 2-bit bitmap and Count-Min. Hence, we can calculate the number of features inserted into SpaceSaving as:

$$N_{Hvy} = CM_{fpr}(BM_{fpr}(n_1+2n_2)+\sum_{i=3}^{1+2^T} i\cdot n_i))+\sum_{i\geq 1+2^T} i\cdot n_i - N_{remain} \quad (5)$$

Tail Contributors Accuracy. Overestimating a feature's frequency has been shown to be a positive property in identifying interesting features based on frequency thresholds [13]. We observe that Panakos never underestimates a cold feature's frequency due to the hash collisions of different features in the 2-bit bitmap and Count-Min sketch. In addition, Panakos also provides similar properties for the hot features.

Lemma 1. *Panakos never underestimates the frequency of a monitored tail feature.*

Proof. Panakos' frequency estimation algorithm has three stages, i.e., bitmap, Count-Min, and SpaceSaving. The bitmap can only overestimate an item's count due to hash collisions. In addition, Count-Min and SpaceSaving are known to have the property

of overestimation [13]. Since all stages never underestimate a monitored tail item's frequency, it is clear that Panakos inherits the property of one-sided error. □

Frequency Estimation Guarantees. We assume the features in a data stream are randomly ordered and further utilize the analysis shown in Section 4.2 to provide different error guarantees for the frequency estimation of the cold, warm, and hot features stored in different stages. As shown in Algorithm 2, the estimated frequency is reported from one of three stages. Depending on the stage, the frequency error guarantees are slightly different:

Bitmap Stage: The feature's frequency is reported based solely on a 2-bit bitmap. This frequency is at most off by 2.

Count-Min Stage: The feature's frequency is reported based on Count-Min. This frequency is off by $2 + \epsilon N_1$ with high probability $1 - \delta$, and is at most off by 2^T.

SpaceSaving Stage: The feature's frequency is reported based on SpaceSaving with K counters. The frequency is at most off by $2^T + \frac{N_2}{K}$.

5 HARDWARE IMPLEMENTATION

In this section, we describe the hardware implementation of Panakos on Intel Tofino switch [31].

PISA programming model. In contrast to conventional switches, protocol-independent switch architecture (PISA) switches offer programmable parsing and customizable packet-processing pipelines, as well as general-purpose registers for stateful operations [6]. These features provide opportunities to realize flexible data (packet) processing at line rate (i.e., order of nanoseconds). Developers use the P4 language [5] to support user-defined packet headers, specify the matching fields and types (e.g., exact, range, and ternary matching), and configure supported actions (e.g., CRC32 hash, header field modification, register read/write via arithmetic logic unit (ALU), arithmetic operations, and metering).

P4 Implementation. We use the Tofino switch's stateful registers to implement Panakos's three critical data structures. More concretely, we implement the bitmap and Count-Min using multiple register arrays with different CRC hash functions. When a packet arrives at the switch, it will update the corresponding hash-indexed counters stored in the registers using ALUs. We adopt the hardware-friendly CocoSketch [64], an extension of SpaceSaving. It reduces the number of memory accesses for resource-constraint hardware. Lastly, we add a cloning stage to clone a packet using Tofino's *mirror* function and send the cloned packet to the control plane for offline analysis. The cloned packet contains information about counter values from Bitmap, Count-Min, and CocoSketch. With the cloning functionality, hardware-friendly Panakos ensures that the original packet remains unaffected.

Latency Improvement. The hardware implementation is a P4 program running on a Tofino switch while the software implementation is a python program running on a server with 2 Intel Xeon Silver 4214R Processor 12-Core 2.4GHz CPU and 192 GB memory. Panakos' hardware implementation is much **(15x) faster** than the software one. More concretely, an insert operation takes around 5-6 microseconds in software implementation and around 400 nanoseconds in hardware. This result demonstrates Panakos' suitability for network monitoring applications.

6 EVALUATION

This section demonstrates that Panakos outperforms existing state-of-the-art solutions. More concretely, for given memory resources, it offers better accuracy in (i) reporting the feature distribution; (ii) identifying the tail contributors; and (iii) extracting essential statistics, such as frequency, cardinality, and entropy, from the raw data stream. we demonstrate the generalizability of observed gains across different datasets and configurations. While we focus on the *Count* operator, we also provide experimental results for the *Sum* and *Average* operators for approximating feature distribution and identifying tail contributors.

6.1 Experimental Setup

For accuracy-related evaluation, we implement all sketches using Python as it enables faster prototyping and offers good readability. For Panakos, we use the standard priority queue data structure to implement the SpaceSaving stage instead of the linked-list data structure proposed in the original paper. Using a priority queue to implement SpaceSaving saves pointer space and avoids random memory accesses. The experiments compare the software implementation of Panakos to other state-of-the-art sketches:

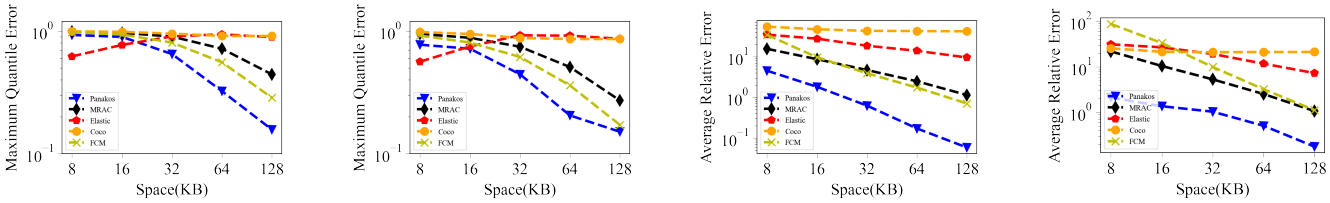
- **MRAC** [37]: MRAC is the seminal statistical summary technique that uses the EM algorithm to approximate the entire feature distribution.
- **CocoSketch** [64]: CocoSketch is an extension of SpaceSaving [45, 56, 65] with implementations on software and hardware platforms. CocoSketch achieves state-of-the-art performance in estimating a feature's frequency and identifying heavy features.
- **ElasticSketch** [62]: ElasticSketch can answer many statistical questions such as feature distribution. It tracks the heavy features first and then uses Count-Min to summarize the cold features.
- **FCMSketch** [55]: FCMSketch is an extension of Count-Min. It imposes a perfect tree hierarchical structure over multiple Count-Min sketches to extract feature distributions and other statistics.

Evaluation Metrics. We use the following three metrics to quantify accuracy.

- **Maximum Quantile Error (MQE)** measures the accuracy of the feature distribution in the quantile domain. MQE is the maximum deviation among all quantiles, also known as the Kolmogorov-Smirnov divergence [8], in which MQE is $max_{i=1}^{|F|_\infty}(abs(\phi_i - \hat{\phi}_i))$.
- **Average Relative Error (ARE)** evaluates a sketch's performance in the frequency domain. ARE is $\frac{1}{|\Upsilon|}\sum_{x\in\Upsilon}\frac{|f(x)-\hat{f}(x)|}{f(x)}$, where Υ is the set of all quantiles in a feature distribution or the set of all unique features when estimating frequencies.
- **F1 score** is the harmonic mean of the precision and recall $(2 \cdot \frac{precision \cdot recall}{precision + recall})$. We use the F1 score to evaluate the performance of each sketch in identifying the tail contributors.

Datasets We use one synthetic and two real-world datasets for evaluation.

- **Zipf Distribution**: We use this synthetic dataset to demonstrate how well Panakos performs for distributions with varying



(a) CAIDA Quantile Error **(b) Sensor Quantile Error** **(c) CAIDA Metric Error** **(d) Sensor Metric Error**

Figure 3: Accuracy of feature distribution approximation on real-world datasets with varying space budgets

skewness. Specifically, we curate multiple datasets with varying skewness. For each dataset, we draw the elements from a bounded universe such that their frequencies follow the Zipf Law [70], in which the frequency of an element with rank R: $f(R, s) = \frac{constant}{R^s}$. Here, s indicates skewness, which is different for different datasets. Unless specified otherwise, we use $s = 1.0$ for our experiments. Typically, real-world data tend to have skewness varying from 0.7 to 1.2, such as IP addresses in network traces, file transmission times, and webpage accesses [15].

- **CAIDA Dataset**: The CAIDA dataset contains anonymized and unsampled packet traces [1], which were captured from a large ISP's backbone link between Seattle and Chicago. We use this dataset to demonstrate how well Panakos performs in count, sum, and average operators for network monitoring use cases. We consider the most common feature, one that reports the metric values for each connection[2], for evaluation.
- **SensorScope**: The Sensorscope dataset contains environmental data from past SensorScope deployments. We use this dataset to demonstrate how well Panakos performs for environmental monitoring use cases. We consider the feature that reports the number of status packets for a group of sensors that share a set of transmission power, primary voltage, global current, and energy source attributes.

To ensure that we can compare results across different datasets, we consistently use *one million* distinct items from each datasets.

Parameter Settings. For a fair comparison, we must decide on appropriate resource allocation policies for different solutions.

ElasticSketch. ElasticSketch uses a hash table and a Count-Min sketch to track heavy and cold items. We use the heuristics described in [62] to decide how much memory to allocate to each data structure. Specifically, we assign 25% of the available memory (B) to the hash table and the rest to the Count-Min sketch.

FCMSketch. FCMSketch imposes a perfect tree structure over multiple Count-Mins. We use the heuristics described in [55] to set the parameters. Specifically, each Count-Min uses two rows and there are three Count-Mins, in total, using 8, 16, and 32 bits per counter and the tree has a branching factor of 8 which means the first Count-Min has 8 times more counters than the second Count-Min and 64 times more counter than the third Count-Min.

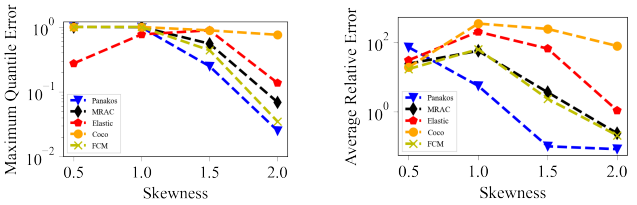
Panakos. Panakos uses bitmap, Count-Min, and SpaceSaving data structures to track cold, warm, and hot items. Let $\{bm, cm, ss\}$ denote the percentage of the memory allocated to the 2-bit bitmap, Count-Min with 4 bits per counter, and SpaceSaving stages. Unless specified otherwise, we use the configuration $\{35\%, 15\%, 50\%\}$ for evaluation. Section 6.4 shows how changing this resource allocation policy affects Panakos' performance.

[2]Network operators use five header fields, i.e., `sIP`, `dIP`, `sPort`, `dPort`, and `proto`, to group packets associated with a network connection.

6.2 Accuracy of Feature Distributions

Space Budget vs. Error. Figure 3(a)-(d) provide a comparison of the accuracy of different sketches in both the quantile and frequency domains on the CIADA and Sensor real datasets using space budgets from 8KB to 128 KB. Since hardware often has constraints imposed on the memory (e.g., NUCLEO-L476RG LoRaWAN board has 128KB SRAM [61]), we believe that 8KB to 128KB reflects state-of-the-art memory constraints on deploying sketches in the real world. Assuming the metric value v has quantile value ϕ_v, the quantile domain uses MQE to characterize the maximum error between the estimated and true quantile values of all ϕ_v. Similarly, the frequency domain uses ARE to characterize the error between the estimated and true frequencies based on ϕ_v. In Figure 3, the y-axis is either MQE or ARE which are used to measure the accuracy in the quantile or frequency domains respectively, and the x-axis is the total space budget used by each sketch. Lower MQE or ARE indicate a more accurate approximation.

As expected, CocoSketch (belongs to the family of counter-based summaries) is not designed to approximate feature distributions as it loses information on cold features and overestimates the frequencies of hot features. We also find that ElasticSketch does not perform well for feature distribution on the quantile domain. Unlike Panakos, ElasticSketch mixes the singletons and doubletons with other medium features. With limited resources, ElasticSketch can not calculate small features accurately and incurs large quantile estimation errors. Hence, ElasticSketch's performance is less robust compared to Panakos and MRAC. MRAC and FCMSketch are strong competitors to Panakos. As pointed out in FCMSketch [55], FCMSketch in general has better performance than MRAC by using layers of Count-Mins instead of storing all features in a single array. However, FCMSketch can not fully utilize the benefit of multiple Count-Mins as it requires all Count-Min sketches to share the same set of hash functions. Using the same set of hash functions and imposing the tree structure implies hot features are exponentially more likely to collide when the tree depth increase. By leveraging linear sketches to store cold and medium features and then using a counter-based summary to explicitly store hot features, Panakos has more accurate approximations than MRAC and FCMSketch in all experiments. Panakos outperforms all the other state-of-the-art

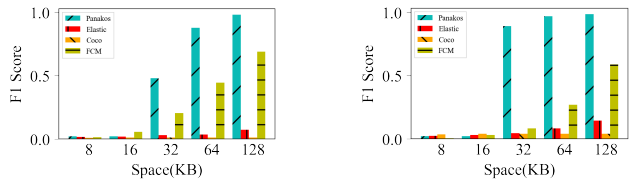


(a) Quantile Error (b) Metric Error

Figure 4: Accuracy of feature distribution approximation using 16KB on Zipf dataset with varying skewness.

sketches in the frequency domain. In the quantile domain, when using more than 32 KB of space, Panakos always provide the best accuracy. In addition, when Panakos uses larger space budgets, it provides a more accurate feature distribution approximation which is also shown in the theoretical expectation, as explained in Section 4.1. Therefore, we believe that Panakos offers better memory vs accuracy trade-off than existing solutions.

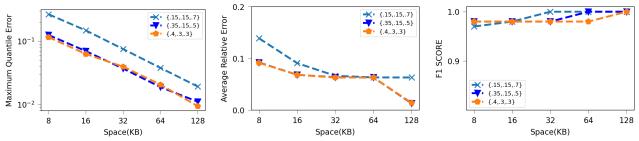
Skewness vs. Error. In Figure 4(a)-(b), we compare the accuracy of different sketches using 16KB space with Zipf datasets. The y-axis is either MQE or ARE to measure the accuracy in the quantile or frequency domains, respectively, and the x-axis is the skewness of the Zipf dataset from 0.5 (low skew) to 2.0 (high skew). Lower MQE or ARE indicate a more accurate approximation.



(a) CAIDA, ψ = .99 (b) Sensor, ψ = .99

Figure 5: F1 scores for identifying the 1% tail contributors on real world datasets with varying space budgets.

For the quantile domain, in general, MQE decreases as skewness increases. Since a higher skewness indicates more hot features, the sketches will have fewer collisions between cold and hot features. Moderate skew (1.0 - 1.5) may not help ElasticSketch as it often mixes small features with medium ones. On the other hand, Panakos can leverage the skewness increase to disentangle cold, warm, and hot features. For the frequency domain, moderate skew also may not help ElasticSketch and CocoSketch. Increasing skewness not only indicates more hot features but also means a lower cardinality. Recall that ARE computes the average distance in the metric error for all unique quantiles. In addition to ElasticSketch mixing small and medium features and CocoSketch only capturing the heavy features, the denominator for the ARE metric also decreases as skewness increases. When the data distribution is relatively uniform, we find that MRAC and FCMSketch have better performance. Small skew implies no hot features and, with a good hash function, all features can be mapped to counters evenly. In this particular



(a) Quantile. (b) Metric. (c) ψ = .99.

Figure 6: Panakos sensitivity analysis with different memory allocations using 10^5 items from Zipf Dataset of 1.5 skewness.

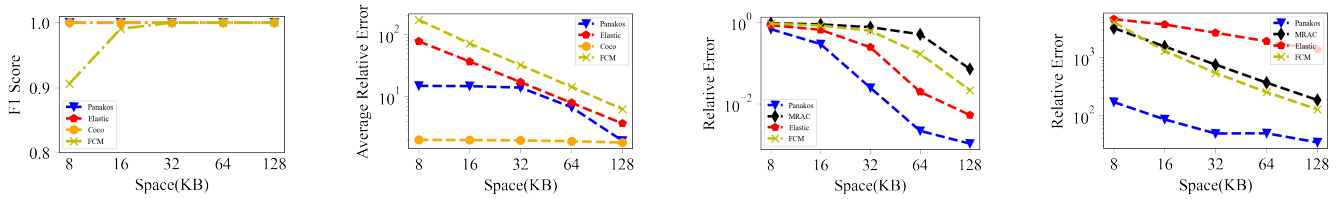
case, the SpaceSaving stage in Panakos becomes less beneficial and incurs higher overheads. When the skew is moderate or high, which is the more common case in real workloads, Panakos provides the best performance among existing solutions.

6.3 Tail Contributors

One of the main goals for Panakos is to use the relative threshold, ψ, in identifying the tail contributors. Figure 5(a) and (b) provide a comparison among sketches in identifying heavy tail contributors with ψ set to 0.99 over real-world CAIDA and SensorScope datasets, in which the experiments aim to identify all the top 1% features. The y-axis is the F1 score, and the x-axis is the space budget for each sketch. F1 is the harmonic mean of the precision and recall, and the closer an F1 score is to 1.0 indicates better accuracy. Moreover, since MRAC cannot identify feature identities, we exclude MRAC from the experiments in this task. Panakos acheives high F1 score in finding the tail contributors. Among all other sketches, FCMSketch is a strong competitor to Panakos. When using 128KB on CAIDA, Panakos achieves a 0.98 F1 score, while FCMSketch achieves a 0.69 F1 score. The drawback of FCMSketch in identifying the tail contributors is that the heavy features are exponentially more likely to collide when the tree depth increases. Panakos, on the other hand, can track hot features explicitly.

6.4 Sensitivity to Memory Allocation Policies

We now evaluate Panakos' sensitivity to memory allocation policies. We consider three different policies. Let an $\{bm, cm, ss\}$ denote the percentage of the total memory allocated to the bitmap, Count-Min, and SpaceSaving stages. We explore three different set of parameters, i.e., $\{.15, .15, .7\}$, $\{.35, .15, .5\}$, and $\{.4, .3, .3\}$. Figure 6(a)-(b) shows that allocating more memory percentage to bitmap and Count-Min stages enables summarizing cold and warm features efficiently, contributing to better MQE and ARE. Figure 6(c) shows that allocating more memory percentage (70%) for SpaceSaving at the cost of bitmap and Count-Min leads to a better F1 score in general. Interestingly, when the space budget is minimal, spending most of the memory on the SpaceSaving stage may not be very helpful, as the mapping between a given quantile threshold and metric threshold still requires some understanding of the entire feature distribution. As a result, we believe that parameter settings are subject to both system operator interest and data distribution. At one extreme of the spectrum, if the operator only cares about heavy hitters, allocating all memory to SpaceSaving would be optimal. At the other extreme of the spectrum, if the operator only cares about learning the number of singletons, then allocating all memory to the



(a) Frequent Items (b) Frequency Estimation (c) Cardinality Estimation (d) Entropy Estimation

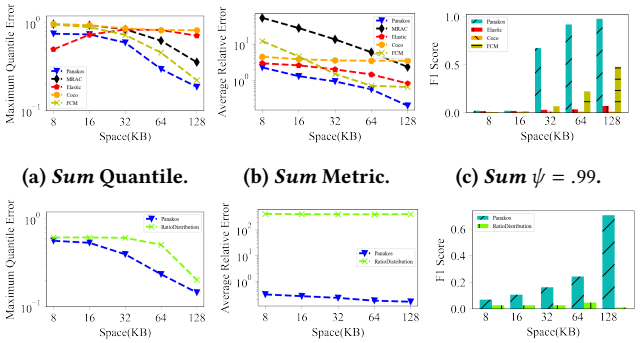
Figure 7: The performance comparisons of sketches on statistical queries over the Zipf 1.5 dataset with varying space budgets.

bitmap stage would be optimal. Our choice of allocating 15%, 35%, and 50% to bitmap, Count-Min, and SpaceSaving data structures attempt to strike a balance among all three metrics of interest. We leave the design of an algorithm that synthesizes optimal memory allocation policy for a given distribution as future work.

6.5 Additional Stream Statistics and Operators

Panakos also accurately estimates other stream statistics for a given memory budget. Figure 7 shows how well Panakos reports frequent items, frequency, cardinality, and entropy estimations for a given Zipf 1.5 stream. Note the entropy of the data stream is estimated from the feature distribution [38], as $-\sum_{i=1}^{\infty} i \cdot \frac{n_i}{card} log \frac{n_i}{card}$ where n_i is the number of features appeared i times, and $card$ is the cardinality. We report the F1 score for frequent items, average relative error (ARE) for frequency estimation, and relative error for cardinality and entropy estimations. For frequent items, we aim to report all hot features with a frequency above ϵN where $\epsilon = 10^{-3}$ and N are a million data items. Panakos performs better than or at par with existing solutions for frequent items, cardinality, and entropy estimations across different memory budgets. The only exception is a higher ARE in frequency estimation for Panakos. The observation is that the heavy features are the main contributors to errors in ARE. To provide better approximation across the entire feature spectrum, Panakos only allocates a fraction of memory to track heavy features, whereas CocoSketch uses all available memory to track heavy features. As memory increase, Panakos' frequency estimation error drastically diminish.

While we focused on the *Count* operator, we also conducted experiments to understand the performance of Panakos with *Sum* and *Average* operators. Note we assume the metric values are integers. The count metric is the occurrence of a 5-tuple in each packet. For the sum metric, we find that the minimum byte chunks in a packet stream are about 64 octets, and the sum metric is measured in units of 64 octets. The average metric is the sum metric divided by the count metric for each 5-tuple. Figure 8 (a)-(c) demonstrate that by using different data structures to track cold, warm, and hot features independently, Panakos offers better accuracy vs. memory usage trade-off than other sketches for the *Sum* operator. Figure 8 (d)-(e) compares the two possible approaches discussed in Section 3.7 to realize the *Average* operator. The result demonstrates the value of Panakos' "alignment-based" approach, which places an item's count near the item's sum, unlike the "ratio distribution" approach, in striking a better accuracy vs. memory usage trade-off.



(a) ***Sum*** **Quantile.** (b) ***Sum*** **Metric.** (c) ***Sum*** ψ = .99.

(d) ***Avg*** **Quantile.** (e) ***Avg*** **Metric.** (f) ***Avg*** ψ = .99.

Figure 8: Approximate feature distribution and identify tail contributors with *Sum* and *Average*.

7 CONCLUSION

This paper presents the design and implementation of Panakos. Our key idea is to leverage the skewness inherent to most feature streams in the real world. Specifically, we disentangle a feature stream into cold, warm, and hot items based on their feature values and use a combination of the bitmap, Count-Min, and SpaceSaving data structures to track them. Our design is generalizable to count, sum, and average operators, and strikes a good balance between accuracy and memory overhead. It is also amenable to modern programmable hardware targets. Our experiments showcase that Panakos is aligned with theoretical expectations, achieves state-of-the-art performance in approximating the feature distribution and identifying the tail contributors, and can provide very accurate estimations for a wide range of statistics. Panakos is applicable to many real-world systems, enabling them to utilize their resources effectively and report interesting statistics for analysis.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. This work is funded in part by NSF grants CNS-1703560 and CNS-1815733. Punnal and Arpit were supported by NSF awards OAC-2126327, OAC-2126281, and CNS-2003257 (co-sponsored by Intel). Liu was supported in part by NSF grants CNS-2107086, CNS-2106946, SaTC-2132643, and Red Hat Collaboratory.

REFERENCES

[1] [n.d.]. Anonymized Internet Traces 2015. https://catalog.caida.org/details/dataset/passive_2015_pcap. Accessed: 2022-10-5.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*. 1093–1110.

[3] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 541–556.

[4] Peter J Bickel and Kjell A Doksum. 2015. *Mathematical statistics: basic ideas and selected topics, volumes I-II package*. Chapman and Hall/CRC.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

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

[7] Tian Bu, Jin Cao, Aiyou Chen, and Patrick PC Lee. 2010. Sequential hashing: A flexible approach for unveiling significant patterns in high speed networks. *Computer Networks* 54, 18 (2010), 3309–3326.

[8] Francesco Paolo Cantelli. 1933. Sulla determinazione empirica delle leggi di probabilita. *Giorn. Ist. Ital. Attuari* 4, 421-424 (1933).

[9] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 2000. Towards estimation error guarantees for distinct values. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 268–279.

[10] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 693–703.

[11] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[12] Saar Cohen and Yossi Matias. 2003. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 241–252.

[13] Graham Cormode and Marios Hadjieleftheriou. 2008. Finding frequent items in data streams. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1530–1541.

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

[15] Graham Cormode and Shan Muthukrishnan. 2005. Summarizing and mining skewed data streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 44–55.

[16] Harald Cramér. 1999. *Mathematical methods of statistics*. Vol. 43. Princeton university press.

[17] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. 2003. Gigascope: A stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 647–651.

[18] Alexander Philip Dawid and Allan M Skene. 1979. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 28, 1 (1979), 20–28.

[19] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. 2002. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms*. Springer, 348–360.

[20] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22.

[21] Garry A Einicke, Gianluca Falco, and John T Malos. 2010. EM algorithm state matrix estimation for navigation. *IEEE Signal Processing Letters* 17, 5 (2010), 437–440.

[22] Seyed K Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. 2015. Bohatei: Flexible and Elastic {DDoS} Defense. In *24th USENIX security symposium (USENIX Security 15)*. 817–832.

[23] Michael Freitag and Thomas Neumann. 2019. Every row counts: Combining sketches and sampling for accurate group-by result estimates. *ratio* 1 (2019), 1–39.

[24] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. 2018. Moment-based quantile sketches for efficient high cardinality aggregation queries. *arXiv preprint arXiv:1803.01969* (2018).

[25] Sumit Ganguly, Phillip B Gibbons, Yossi Matias, and Avi Silberschatz. 1996. Bifocal sampling for skew-resistant join size estimation. In *Proceedings of the 1996 ACM SIGMOD international conference on management of data*. 271–281.

[26] John D Gorman and Alfred O Hero. 1990. Lower bounds for parametric estimation with constraints. *IEEE Transactions on Information Theory* 36, 6 (1990), 1285–1301.

[27] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record* 30, 2 (2001), 58–66.

[28] Sudipto Guha and Andrew McGregor. 2009. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM J. Comput.* 38, 5 (2009), 2044–2059.

[29] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 357–371.

[30] Hazar Harmouch and Felix Naumann. 2017. Cardinality estimation: An experimental survey. *Proceedings of the VLDB Endowment* 11, 4 (2017), 499–512.

[31] Intel. 2022. Barefoot Tofino. https://barefootnetworks.com/products/brief-tofino/. [Online; accessed 19-July-2022].

[32] Nikita Ivkin, Edo Liberty, Kevin Lang, Zohar Karnin, and Vladimir Braverman. 2022. Streaming quantiles algorithms with small space and update time. *Sensors* 22, 24 (2022), 9612.

[33] Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. 2019. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 285–291.

[34] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 71–78.

[35] Richard M Karp, Scott Shenker, and Christos H Papadimitriou. 2003. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)* 28, 1 (2003), 51–55.

[36] Kibeom Kim, Yongjo Jeong, Youngjoo Lee, and Sunggu Lee. 2019. Analysis of counting bloom filters used for count thresholding. *Electronics* 8, 7 (2019), 779.

[37] Abhishek Kumar, Minho Sung, Jun Xu, and Jia Wang. 2004. Data streaming algorithms for efficient and accurate estimation of flow size distribution. *ACM SIGMETRICS Performance Evaluation Review* 32, 1 (2004), 177–188.

[38] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun Xu, and Hui Zhang. 2006. Data streaming algorithms for estimating entropy of network traffic. *ACM SIGMETRICS Performance Evaluation Review* 34, 1 (2006), 145–156.

[39] Alexandru Lavric and Valentin Popa. 2017. Internet of things and LoRa™ low-power wide-area networks: a survey. In *2017 International Symposium on Signals, Circuits and Systems (ISSCS)*. IEEE, 1–5.

[40] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 101–114.

[41] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A {High-Performance}{Switch-Native} Approach for Detecting and Mitigating Volumetric {DDoS} Attacks with Programmable Switches. In *30th USENIX Security Symposium (USENIX Security 21)*. 3829–3846.

[42] Patrick Loiseau, Paulo Gonçalves, Stéphane Girard, Florence Forbes, and Pascale Vicat-Blanc Primet. 2009. Maximum likelihood estimation of the flow size distribution tail index from sampled packet data. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. 263–274.

[43] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1998. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record* 27, 2 (1998), 426–435.

[44] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2008. Why go logarithmic if we can go linear? Towards effective distinct counting of search traffic. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. 618–629.

[45] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *International conference on database theory*. Springer, 398–412.

[46] Jayadev Misra and David Gries. 1982. Finding repeated elements. *Science of computer programming* 2, 2 (1982), 143–152.

[47] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 85–98.

[48] Rasmus Pagh and Mikkel Thorup. 2022. Improved Utility Analysis of Private CountSketch. *arXiv preprint arXiv:2205.08397* (2022).

[49] C Radhakrishna Rao. 1945. Information and the accuracy attainable in the estimation of statistical parameters. *Reson. J. Sci. Educ* 20 (1945), 78–90.

[50] Bruno Ribeiro, Don Towsley, Tao Ye, and Jean C Bolot. 2006. Fisher information of sampled packets: an application to flow size estimation. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 15–26.

[51] Jorma J Rissanen. 1996. Fisher information and stochastic complexity. *IEEE transactions on information theory* 42, 1 (1996), 40–47.

[52] Pratanu Roy, Arijit Khan, and Gustavo Alonso. 2016. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International*

Conference on Management of Data. 1449–1463.

[53] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. 2004. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*. 239–249.

[54] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. 2017. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*. 164–176.

[55] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. 2020. FCM-sketch: generic network measurements with data plane support. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 78–92.

[56] Daniel Ting. 2018. Data sketches for disaggregated subset sum and frequent item estimation. In *Proceedings of the 2018 International Conference on Management of Data*. 1129–1140.

[57] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. 2013. Quantiles over data streams: an experimental study. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 737–748.

[58] Pinghui Wang, Xiaohong Guan, Junzhou Zhao, Jing Tao, and Tao Qin. 2014. A new sketch method for measuring host connection degree distribution. *IEEE transactions on information forensics and security* 9, 6 (2014), 948–960.

[59] Kyu-Young Whang, Brad T Vander-Zanden, and Howard M Taylor. 1990. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems (TODS)* 15, 2 (1990), 208–229.

[60] CF Jeff Wu. 1983. On the convergence properties of the EM algorithm. *The Annals of statistics* (1983), 95–103.

[61] Mingran Yang. 2020. *Joltik: enabling energy-efficient" future-proof" analytics on low-power wide-area networks*. Ph.D. Dissertation. Carnegie Mellon University.

[62] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 561–575.

[63] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software {Defined}{Traffic} Measurement with {OpenSketch}. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 29–42.

[64] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. 2021. CocoSketch: high-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 207–222.

[65] Fuheng Zhao, Divyakant Agrawal, Amr El Abbadi, and Ahmed Metwally. 2021. SpaceSaving$^{\pm}$: An Optimal Algorithm for Frequency Estimation and Frequent Items in the Bounded Deletion Model. *arXiv preprint arXiv:2112.03462* (2021).

[66] Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. 2021. KLL±: Approximate quantile sketches over dynamic datasets. *Proceedings of the VLDB Endowment* 14, 7 (2021), 1215–1227.

[67] Fuheng Zhao, Dan Qiao, Rachel Redberg, Divyakant Agrawal, Amr El Abbadi, and Yu-Xiang Wang. 2022. Differentially private linear sketches: Efficient implementations and applications. *arXiv preprint arXiv:2205.09873* (2022).

[68] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. 2018. Cold filter: A meta-framework for faster and more accurate stream processing. In *Proceedings of the 2018 International Conference on Management of Data*. 741–756.

[69] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491.

[70] George Kingsley Zipf. 2016. *Human behavior and the principle of least effort: An introduction to human ecology*. Ravenio Books.