How to scalably and accurately skip past streams

Supratik Bhattacharyya
Sprint ATL
supratik@gmail.com

André Madeira, S. Muthukrishnan
Rutgers University
[amadeira,muthu]@cs.rutgers.edu

Tao Ye
Sprint ATL
tao.ye@sprint.com

Abstract

Data stream methods look at each new item of the stream, perform a small number of operations while keeping a small amount of memory, and still perform much-needed analyses. However, in many situations, the update speed per item is extremely critical and not every item can be extensively examined. In practice, this has been addressed by only *examining every N^{th} item from the input; decreasing the input rate by a fraction $1/N$, but resulting in loss of guarantees on the accuracy of the post-hoc analyses.*

In this paper, we present a technique of skipping past streams and looking at only a fraction of the input. Unlike traditional methods, our skipping is performed in a principled manner based on the "norm" of the stream seen. Using this technique on top of well-known sketches, we show several-fold improvement in the update time for processing streams with a given guaranteed accuracy, for a number of stream processing problems including data summarization, heavy hitters detection and self-join size estimation.

We present experimental results of our methods over synthetic data and integrate our methods into Sprint's Continuous Monitoring (CMON) system for live network traffic analyses. Furthermore, aiming at future scalable stream processing systems and going beyond state-of-art packet header analyses, we show how the packet contents *can be analyzed at streaming speeds, a more challenging task because each packet content can result in many updates.*

1. Introduction

A challenge data management applications face is processing and analyzing massive "streams" of data where items arrive at a fast rate. Monitoring IP network traffic data is such an application where data stream management systems (DSMSs) monitor each IP packet sent on a communication link and perform detailed statistical analysis. In the past few years, a number of *sketching* and *selection* methods have been proposed that look at each new item, perform a small number of operations while keeping a small amount of memory (aka *sketches* or *samples*), and still perform much-needed analyses on streams including data summarization, finding heavy hitters and quantiles, estimating self-join and statistical moments, etc. Operational DSMSs such as Gigascope [9] at AT&T and CMON [16] at Sprint are able to monitor hundreds of thousands of packet headers with these algorithms. This is essential for nearly every aspect of network management, including fault diagnosis, verifying service level agreements on network performance and most importantly, network security.

One of the most critical elements of a DSMS is the rate at which updates may be processed. In particular, in the IP network management application, there are three developments that force ever greater rates of updates.

New technology. In the backbone of Internet Service Providers (ISPs), existing links operate typically at OC-48 speeds, or 2.5 Gbits/s. Increasingly, ISPs are using higher speed links (OC-192's or even OC-768's) that operate at up to 16 times that speed. DSMSs already struggle dealing with current rates and are required to scale up several-fold.

New functionality. Existing DSMSs typically analyze IP packet *headers*, which has source/destination IP addresses as well as port numbers among other things. Motivated by worm, virus, and application detection, DSMSs are faced with increasing demands to do "deep packet inspection", that is, analyze the *contents* of IP packets. Since IP contents are several tens of factors larger than the size of the headers, DSMSs must now do more computing per packet.

Network events. Flash crowds and attack events spike up the network traffic levels. DSMSs must handle these very high peak rates for prolonged periods of time. ∎

Therefore, there is a great need to study how to speed up streaming algorithms. In practice, to the extent DSMSs face these challenges, it is solved by downsampling the input, say 1 in N, so the input rate is reduced to manageable levels. But this results in loss of guarantees on the accuracy of many of the post-hoc analyses of interest.

In this paper, we propose an approach towards streaming that obtains speedup by skipping over portions of the stream. As a result, our approach behaves like downsampling in reducing the rate of data to be processed by DSMSs. However, unlike existing methods, the skipping is done in a principled manner dependent on the *norm* of the data seen so far and the intended applications. Consequently, we prove accuracy guarantees and still obtain provable speedups. Precisely, our contributions are as follows:

1. We present a way to skip over the portions of the stream so that known sketching algorithms only process a subset of the input. We apply this to algorithms for point queries with summaries, detecting heavy hitters and self-join estimation to obtain significant speedups for per-item processing time while maintaining the accuracy guarantees of the analyses. All these analyses have several applications in DSMSs.

2. We present detailed experimental study of our skipping-based sketching algorithms over synthetic data and integrate our methods into Sprint's Continuous Monitoring (CMON) system for live network traffic analysis, showing that skipping significantly improves performance; the improvement is in the order of two- to three-fold over standard sketches.

3. As a concrete application of skipping, we present experimental studies of heavy hitters summarization on the q-grams within IP packet contents, which is of interest for early worm and virus detection. We show accurate analyses and obtain impressive, order of 10-fold speedups. Our method not only skips portions of packet contents but even entire packets! This leads to substantial improvements and shows for the first time stream analyses of IP packet contents that is practical at backbone speeds. Existing deep-packet inspection methods in research and industry either relies on specialized hardware or looking at partial traces offline.

Section 2 presents standard sketches and streaming problems. In Section 3, we present our skipping framework and show its applicability in Section 4. In Section 5, we present our experimental results. Finally, we review some related work in Section 6 and conclude in Section 7.

2. Preliminaries

In the input *data stream*, items $v_1, v_2, \ldots$ arrive sequentially, describing an underlying vector $\boldsymbol{V}$ on domain $[N] = [0, \ldots, N-1]$. Item t is the update $v_t = (i_t, c_t)$, $c_t \geq 0$. Formally, on seeing an update v_t, $\boldsymbol{V}_t[i] = \boldsymbol{V}_{t-1}[i] + c_t$, where $\boldsymbol{V}_t$ is the state of vector $\boldsymbol{V}$ after the tth update. Note that multiple v_t's could increment the same value $\boldsymbol{V}[i]$.

Examples. Consider network traffic monitoring application. Say $\boldsymbol{V}_t$ is the total number of bytes sent by each IP address after seeing t IP packets. Each IP packet p_t has a source IP address s_t and size S_t in bytes that may be thought of as an update (s_t, S_t), ie., $\boldsymbol{V}_t[s_t] = \boldsymbol{V}_{t-1}[s_t] + S_t$. As another application, say $\boldsymbol{V}_t[i]$ is the number of occurrences of q-gram (substring of length q) after seeing t IP packets. Now each IP packet p_t entails updating a lot of $\boldsymbol{V}_t[i]$'s, one for each distinct q-gram in p_t. ■

We denote the L_1 and L_2 norms of vector $\boldsymbol{V}$ by $|\boldsymbol{V}| = |\boldsymbol{V}|_1 = \sum_i \boldsymbol{V}[i]$ and $||\boldsymbol{V}|| = ||\boldsymbol{V}||_2 = \sqrt{\sum_i (\boldsymbol{V}[i])^2}$ respectively.

2.1. The Count-Min Sketch

Sketches are small space data structures to maintain on data streams. Many sketches have been proposed in DSMSs. Throughout this paper, we adopt the Count-Min (CM) sketch [7] which has nearly the best time and space performance for a variety of stream analyses.

The data structure is a two-dimensional array of counters with width w and depth d: $count[1,1] \ldots count[d,w]$. Each counter is initially zero. Additionally, d hash functions $h_1 \ldots h_d : \{1 \ldots N\} \to \{1 \ldots w\}$ are chosen uniformly at random from a pairwise-independent family. Whenever an update (i_t, c_t) arrives, meaning that item $\boldsymbol{V}_t[i]$ should be updated by a quantity of c_t, the value c_t is added to only one count in each row of the sketch. These counters are determined and updated as follows: $\forall 1 \leq j \leq d$,

$$count[j, h_j(i_t)] \leftarrow count[j, h_j(i_t)] + c_t.$$

The update procedure is illustrated in Figure 1(a).

The update time depends on the depth d of the sketch, i.e. on the number of hash functions used per item. The space used by the CM sketch is the array of wd counters plus the d hash functions (both counters and hash functions can be stored with constant words of memory). The output procedure varies with the application as well as the choice of w and d. These parameters dictate the desired guarantees and are typically $w = O(1/\varepsilon)$ and $d = O(\log(1/\delta))$ for a wide range of applications of interest.

2.2. Data Streams Problems

We focus on three quintessential problems in data stream management and show how the CM sketch solves them. We discuss extensions to other problems of interest in Section 7.

Data Summarization. The stream is summarized in small space using a sketch as the "index". At any time, given a query item $i \in [N]$, the problem is to estimate $\boldsymbol{V}[i]$ from the sketch. Data summarization is of great interest in DSMSs. For example, in network traffic analysis, once the stream is summarized, we can support queries such as how many bytes/packets were sent/received by a given IP address.

The CM sketch algorithm for this problem works as follows. The update procedure is the same as described

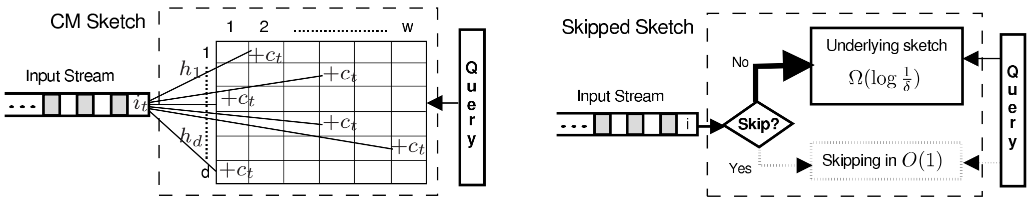


Figure 1. Update procedures in both CM sketch and *skipped sketch* with confidence parameter δ.

previously, while the output estimate is simply $\hat{\boldsymbol{V}}[i] = \min_j count[j, h_j(i)]$. The *width* of the CM sketch is $\lceil e/\varepsilon \rceil$ (where e is the natural logarithm base) and the *depth* is $\lceil \log 1/\delta \rceil$. Hence, the CM sketch uses $O(\log(1/\delta)/\varepsilon)$ space and update time $O(\log(1/\delta))$. The CM sketch guarantees that $\hat{\boldsymbol{V}}[i] \geq \boldsymbol{V}[i]$ with certainty and $\hat{\boldsymbol{V}}[i] \leq \boldsymbol{V}[i] + \varepsilon|\boldsymbol{V}|_1$ with probability $1 - \delta$ (see [7] for a detailed proof).

Self-Join size estimation. Given a relation $\boldsymbol{R}$ with an attribute on domain $1, \ldots, N$, let f_i be the frequency of attribute value i. Then, the *self-join size* is $\sum_{i \in [N]} f_i^2$. Estimating it is one of the earliest queries studied in DSMSs since it is used to estimating the skew in the data and as an upper bound to the join size of any number of relations [2].

If one represents the frequency vector of items in $\boldsymbol{R}$ by a vector $\boldsymbol{V}$, i.e. $\boldsymbol{V}[i] = f_i$, the self-join size is the square of the L_2-norm of $\boldsymbol{V}$. Variants of CM sketch (similar update and query procedure as before) set $w = \lceil e/\varepsilon^2 \rceil$ and $d = \lceil \log(1/\delta) \rceil$ and thus use $O\left(\log(1/\delta)/\varepsilon^2\right)$ space and $O(\log(1/\delta))$ update time to output an estimate $||\tilde{\boldsymbol{V}}||^2 \in (1 \pm \varepsilon)||\boldsymbol{V}||^2$ with probability at least $1 - \delta$ (see [8] for a detailed description and proof).

Heavy Hitters. The $\langle \varepsilon, \delta, \phi \rangle$-approximation problem for heavy hitters consist of returning all items i such that $\boldsymbol{V}[i] \geq \phi|\boldsymbol{V}|$ and no i such that $\boldsymbol{V}[i] < (\phi - \varepsilon)|\boldsymbol{V}|$ for some specified $\varepsilon < \phi$, with probability at least $1 - \delta$. The returned items are the heavy hitters that are larger than a specified fraction of the entire norm. It is commonly used in network traffic analysis, for example, in finding addresses that account for a large fraction (in bytes or number of packets) of a network link utilization in a time window.

The approach in [7] is as follows. The algorithm updates $v_t = (i_t, c_t)$ to a regular CM sketch instance and then poses a query for $\hat{\boldsymbol{V}}[i]$. If the returned estimate $\hat{\boldsymbol{V}}[i]$ is a heavy hitter up to time t, i.e., $\hat{\boldsymbol{V}}[i] \geq \phi|\boldsymbol{V}_t|$, item i is added to a heap. The heap is kept small by deleting any item with count less than $\phi|\boldsymbol{V}_t|$ at each time t. At the end, all items in the heap which still exceed $\phi|\boldsymbol{V}_t|$ are output. Clearly, the algorithm retrieves all heavy hitters (ie., no false negatives), since the CM sketch guarantees $\hat{\boldsymbol{V}}[i] \geq \boldsymbol{V}[i]$. This algorithm uses $O(\log(N/\delta)/\varepsilon)$ space and $O(\log(N/\delta) + \log(1/\phi))$ update time per item (again see [7] for a detailed proof). ■

3. Skipped Sketches

Standard sketch-based solutions to queries described above involve computing several hashes for each new stream item and in some cases, maintaining auxilary information such as the heap. These hash computations can easily dominate the overall algorithm processing time in a DSMS. Thus, minimizing such functions or not executing them at all is critical to achieving higher processing rates.

Skipped sketches allow standard sketches to avoid the bulk of hash computations on certain items. More specifically, skipped sketches avoid processing a given item (ie., *skip* an item) if the desired accuracy of the underlying standard sketch will not be violated. Items not skipped are processed thereafter as in a standard sketch. The query procedure is similar as in standard sketches, while the update procedure is novel and comprises of two alternating phases:

Sketching Phase. Each item is added to the underlying standard sketch, thus performing hash computations.

Skipping Phase. If we decide to skip an item, it is not processed further.

The *crux* is the determination of when to switch between phases. This should ideally be done in constant time independent of the number of hash functions in the standard sketch. We do this determination by a variety of ways that track the vector norms of the skipped and the sketched parts.

Recall that $\boldsymbol{V}$ represents the vector of all updates in the input stream. Let vectors $\boldsymbol{L}$ and $\boldsymbol{R}$ represent the sketched and skipped updates respectively. Clearly, $|\boldsymbol{V}| = |\boldsymbol{L}| + |\boldsymbol{R}|$. Now, let s denote the time when the last phase transition occurred, and t be the current time. The time subscript on a vector norm denotes its value at that time. A phase transition occurs if one of the following conditions is met. Upon processing update $v_t = (i_t, c_t)$, the algorithm:

- moves from the sketching phase to the skipping phase if $|\boldsymbol{L}_{t-1}| + c_t > |\boldsymbol{L}_s| + T$;
- moves from the *skipping* phase to the *sketching* phase if $|\boldsymbol{R}_{t-1}| + c_t > \varepsilon'|\boldsymbol{V}_t|$; and remains in the current phase otherwise.

Here ε' is a specified *skipping rate* parameter and T is a threshold (the value of T does not affect the analysis and thus can be thought of a system parameter). We refer to *conservative* skipping when $\varepsilon' < 1$. When the skipping rate $\varepsilon' \geq 1$, we refer to it as *agressive* skipping. In this case, the algorithm changes slightly and the transition from the skipping phase to the sketching phase happens whenever $|\boldsymbol{R}_{t-1}| + c_t > |\boldsymbol{L}_{t-1}|$ instead.

Figure 1(b) shows conceptually how the update procedure works. The first (few) update(s) will always be sketched. When an item is sketched, it is treated as an update to the standard sketch. If the update can be skipped, it is processed by an (ideally) constant procedure that (possibly) summarizes $\boldsymbol{R}$. For example, estimating $|\boldsymbol{R}|$ requires only one counter, and is used to determine when to switch between phases. Also, queries can make use of both the sketch and the statistics on the skipped items. Observe that for a specified parameter ε', $|\boldsymbol{R}| \leq \varepsilon'|\boldsymbol{V}|$ when $\varepsilon' < 1$ and $|\boldsymbol{R}| \leq \frac{\varepsilon'}{1+\varepsilon'}|\boldsymbol{V}|$ when $\varepsilon' \geq 1$, at any time.

The following is an example depicting the update procedure of a *skipped* sketch that summarizes $|\boldsymbol{R}|$ through a simple counter. Such approach is used in the data summarization and heavy hitters applications of the next section.

Example 1. Let $\varepsilon' = 0.2$, $T = 50$, $N = \{a, b, c\}$ and the start of the input stream $\boldsymbol{V}$ is: $(a, 100)$, $(b, 20)$, $(a, 40)$, $(c, 60)$, $(b, 10)$, $(c, 10)$, $(a, 20)$, $\ldots$ The algorithm first sketches $(a, 100)$ and then skips items summing up to at most $\varepsilon'|\boldsymbol{V}| = 20$. Update $(b, 20)$ is thus skipped, but $(a, 40)$ and $(c, 60)$ are sketched. Now, $|\boldsymbol{L}| = 200$ and $|\boldsymbol{R}| = 20$ allowing the system to skip up to $\varepsilon'(|\boldsymbol{L}| + |\boldsymbol{R}|) = 44$. But since it has already skipped 20, it can only skip up to 24 next. It does so by skipping $(b, 10)$ and $(c, 10)$. The last item is sketched. At the end, $|\boldsymbol{L}| = 220$ and $|\boldsymbol{R}| = 40$. ■

Skipped sketches are thus *norm-aware* in processing the input stream. The high level description and example above used the L_1 norm. In certain cases, we may need to use other norms for the purpose of switching between phases, but estimating these norms without incurring the cost of a standard sketch update may itself become challenging! Carefully constraining the norm of the skipped part with respect to the entire stream will let us provide accuracy guarantees equivalent of standard sketches but improved amortized update cost. The more one skips, the smaller the amortized update cost, but the less accurate the results are. Hence, we need to balance the tradeoff among these factors to get the best results. In addition, the cost of the computation to switch phases is an overhead to be minimized.

4. Applications

We present and analyze skipped sketches for the three applications mentioned in Section 2.2.

4.1. Data Summarization

This is the simplest case of skipped sketch, which monitor the L_1 norm of $\boldsymbol{L}$ and $\boldsymbol{R}$ with just a counter each. We refer to this algorithm as the Skipped Count-Min (SCM).

SCM Update Procedure: The SCM update procedure follows the same skeleton presented in Section 3 with the underlying sketch being a CM sketch (Figure 1(b)).

SCM Query Procedure: Let $\hat{\boldsymbol{L}}[i]$ denote the estimated count for item i output by the underlying CM sketch. The SCM query simply outputs $\tilde{\boldsymbol{V}}[i] = \hat{\boldsymbol{L}}[i]$. If desired, the results can be scaled by $(\hat{\boldsymbol{L}}[i]/|\boldsymbol{L}|) \cdot |\boldsymbol{R}|$, or similar, if the distribution of input items is reasonably known.

Accuracy Analysis. To analyze the accuracy of queries on a SCM sketch, we need only to consider the moment at the end of the skipping phase. At any other time, vector $\boldsymbol{L}$ approximates vector $\boldsymbol{V}$ even tighter.

Theorem 1. The estimate $\tilde{\boldsymbol{V}}[i]$ output by SCM for any query item $i \in [N]$ satisfies: $\tilde{\boldsymbol{V}}[i] \geq \boldsymbol{V}[i] - \varepsilon'|\boldsymbol{V}|$; and, $\tilde{\boldsymbol{V}}[i] \leq \boldsymbol{V}[i] + \varepsilon|\boldsymbol{V}|$ with probability at least $1 - \delta$, for $0 < \varepsilon' < 1$ and $0 < \varepsilon < 1$.

PROOF. The amount skipped is at most $|\boldsymbol{R}| \leq \varepsilon'|\boldsymbol{V}|$. In the worst case, we have $\tilde{\boldsymbol{V}}[i] \geq \boldsymbol{V}[i] - \varepsilon'|\boldsymbol{V}|$, because the underlying CM sketch guarantees $\tilde{\boldsymbol{V}}[i] \geq \boldsymbol{L}[i]$ and all items skipped could be comprised solely of item i. For the other direction, the error term comes directly from the underlying CM sketch guarantees (see Section 2.2). ■

Notice that there are two approximations in the guarantee above, one due to the underlying CM Sketch (ε) and the other due to skipping (ε').

Time & Space Analysis. We present an amortized analysis for the update procedure running time. Recall that the skipping technique is norm-aware and thus the number of items skipped (and consequently its update time) depends on the distribution of counts in the stream. We analyze the case where all updates are of unit count ($c_t = 1$ for all updates (i_t, c_t)) since it is easier to see the improvements in update times, and this case has a direct application we study in Section 5.3. Although a similar analysis can be done for updates with counts other than 1, it is more difficult to interpret the improvements in update times and the tradeoff involved in choosing the skipping rate.

Theorem 2. The SCM sketch guarantees query time in $O\left(\log \frac{1}{\delta}\right)$ and space $O\left(\frac{1}{\varepsilon} \log \frac{1}{\delta}\right)$. Assuming all updates have unit counts, the amortized update time per element is $O\left((1 - \varepsilon') \log \frac{1}{\delta} + \varepsilon'\right)$ for skipping rate $0 < \varepsilon' < 1$, and $O\left(\log(1/\delta)/\varepsilon'\right)$ for $\varepsilon' \geq 1$.

PROOF. The SCM sketch only requires two extra counters for $|\boldsymbol{L}|$ and $|\boldsymbol{R}|$ beyond the space needed for the CM sketch.

Similarly, the query procedure requires only one extra operation if one wishes to scale the output of the CM sketch. Thus space and query time bounds follow from that of the CM sketch (see Section 2.2).

For the update time, consider without loss of generality, the moment when the algorithm switches to the sketching phase. In *conservative* skipping where $0 < \varepsilon' < 1$, the algorithm has sketched $|\boldsymbol{L}_t| = |\boldsymbol{V}_t| - \varepsilon'|\boldsymbol{V}_t|$ and skipped $|\boldsymbol{R}_t| = \varepsilon'|\boldsymbol{V}_t|$ items. It takes $\lceil \log \frac{1}{\delta} \rceil$ hash computations to sketch an item and constant time (say one unit time) to skip it. Therefore, the amortized time is: $[(1-\varepsilon')|\boldsymbol{V}_t| \cdot \lceil \log \frac{1}{\delta} \rceil + \varepsilon'|\boldsymbol{V}_t| \cdot 1)]/|\boldsymbol{V}_t| = O\left((1-\varepsilon')\log\frac{1}{\delta} + \varepsilon'\right)$. In *aggressive* skipping where $\varepsilon' \geq 1$ and $|\boldsymbol{R}_t| \leq \frac{\varepsilon'}{1+\varepsilon'}|\boldsymbol{V}_t|$ at the start of the sketching phase, the amortized time is: $[\frac{1}{1+\varepsilon'}|\boldsymbol{V}_t| \cdot \log\frac{1}{\delta} + \frac{\varepsilon'}{1+\varepsilon'}|\boldsymbol{V}_t| \cdot 1]/|\boldsymbol{V}_t| = \frac{1+\log 1/\delta}{1+\varepsilon'} = O\left(\log(1/\delta)/\varepsilon'\right)$.■

Theorem 1 shows that the total worst case approximation error is $(\varepsilon + \varepsilon')|\boldsymbol{V}|$ and that the update processing time varies roughly linearly with the skipping rate ε'. In other words, if we choose $\varepsilon' = 1/10$, we can conclude that for conservative skipping, the decrease in update time is essentially ε' of the original processing time. Although this is a good improvement, more can be achieved with aggressive skipping. The lower bound $\boldsymbol{V}[i] - \varepsilon'|\boldsymbol{V}| \leq \tilde{\boldsymbol{V}}[i]$ represents a unrealistic worst-case when all the skipped items fall into one of the hashed buckets that i falls into in the CM sketch for *each* hash function. Typically, one expects to skip items that fall into different buckets. We can formally analyze this assuming when the updates are uniformly random. Then, in the expected case, $\boldsymbol{V}[i] - \varepsilon'|\boldsymbol{V}|/(e/\varepsilon) = \boldsymbol{V}[i] - (\varepsilon\varepsilon'/e)|\boldsymbol{V}|$, the total skipped count is divided evenly among the width $w = \lceil e/\varepsilon \rceil$ of the CM sketch. Hence the error now is $\varepsilon + \varepsilon\varepsilon'/e$ and the update rate is $O\left(\log(1/\delta)/\varepsilon'\right)$. To achieve a total error $t = \varepsilon + \varepsilon\varepsilon'/e$, one can vary ε and ε' in different ways. For example, for $t = 0.1$, one can choose ε to be much smaller than t (say $\varepsilon = t/10$), consequently setting ε' to a large value ($\varepsilon' \approx 20$) and thus saving per-item processing time by a factor of nearly 20 without affecting the total expected error. Experimentally we will show that aggressive skipping does not decrease the accuracy significantly on real streams while still improving the update time.

To summarize, *conservative* skipping provides guarantees in the worst case and improves the performance of update time by a small factor; while *aggressive* skipping gives guarantees for uniform update streams only but improves the update time substantially. This summary holds for each of the other analyses we apply skipping to. Hence, we do not discuss it again until we present experimental studies.

4.2. Self-Join Size Estimation

The L_2-norm is defined as $||\boldsymbol{V}|| = \left(\sum_i \boldsymbol{V}[i]^2\right)^{1/2}$, while the self-join size, or the second frequency moment, is defined as $F_2 = (L_2)^2 = ||\boldsymbol{V}||^2$. The CM sketch itself provides an $\langle \varepsilon, \delta \rangle$-approximation for estimating F_2 [8]. The goal of any skipping method is thus to approximate the terms within double brackets in $||\boldsymbol{V}||^2 = ||\boldsymbol{L}||^2 + [\![||\boldsymbol{R}||^2 + 2\langle \boldsymbol{L} \cdot \boldsymbol{R} \rangle]\!]$, because the first term can be approximated by the underlying sketch, i.e., $||sk(\boldsymbol{L})||^2 = (1 \pm \varepsilon)||\boldsymbol{L}||^2$ [8]. However, all known methods to $\langle \varepsilon, \delta \rangle$-approximate either $||\boldsymbol{R}||^2$, the second frequency moment, or $\langle \boldsymbol{L} \cdot \boldsymbol{R} \rangle$, the inner product of two vectors in the streaming model require $\Omega(\log(1/\delta))$ processing time per item. Our goal is to achieve a $O(1)$ update time for the skipping phase. Here, we present a new skipping algorithm that guarantees a constant factor approximation to F_2 with such goal in mind.[1] Later, we briefly outline other heuristics for the same task that perform well experimentally, but do not have any theoretical guarantees.

4.2.1 Constant factor self-join approximation

We show a roughly factor 4 approximation for F_2 with skipping. The algorithm follows the skipping framework as before, except that the sketching phase does not start until $|\boldsymbol{R}|^2 > \varepsilon'||sk(\boldsymbol{L}_s)||^2$. The output of the algorithm $||\tilde{\boldsymbol{V}}||^2$ is simply the output of the underlying CM sketch plus $|\boldsymbol{R}|^2$, i.e., $||\tilde{\boldsymbol{V}}||^2 = ||sk(\boldsymbol{L})||^2 + |\boldsymbol{R}|^2$.

Accuracy Analysis. The following theorem summarizes the accuracy guarantees of this algorithm.

Theorem 3. The algorithm above outputs $||\tilde{\boldsymbol{V}}||^2$ as an approximation to $||\boldsymbol{V}||^2$ such that $(1/2-\varepsilon)||\boldsymbol{V}||^2 \leq ||\tilde{\boldsymbol{V}}||^2 \leq (2+2\varepsilon)||\boldsymbol{V}||^2$, with probability at least $1-\delta$ for $0 < \varepsilon < 1$.

PROOF. Recall that the underlying CM sketch guarantees $||sk(\boldsymbol{L})||^2 \in (1 \pm \varepsilon)||\boldsymbol{L}||^2$. Since $|\boldsymbol{R}|^2 \leq \varepsilon'||sk(\boldsymbol{L})||^2$,

$$\begin{aligned} ||\tilde{\boldsymbol{V}}||^2 &= ||sk(\boldsymbol{L})||^2 + |\boldsymbol{R}|^2 \\ &\leq (1+\varepsilon)||\boldsymbol{L}||^2 + \varepsilon'(1+\varepsilon)||\boldsymbol{L}||^2 \leq (2+2\varepsilon)||\boldsymbol{V}||^2. \end{aligned}$$

For the lower bound,

$$\begin{aligned} ||\tilde{\boldsymbol{V}}||^2 &= ||sk(\boldsymbol{L})||^2 + |\boldsymbol{R}|^2 \geq (1-\varepsilon)||\boldsymbol{L}||^2 + |\boldsymbol{R}|^2 \\ &= (1-\varepsilon)(||\boldsymbol{V}||^2 - ||\boldsymbol{R}||^2 - 2\langle \boldsymbol{L} \cdot \boldsymbol{R} \rangle) + |\boldsymbol{R}|^2 \\ &\geq (1-\varepsilon)(||\boldsymbol{V}||^2 - ||\boldsymbol{R}||^2 - ||\boldsymbol{V}||^2/2) + |\boldsymbol{R}|^2 \\ &\geq (1/2-\varepsilon)||\boldsymbol{V}||^2 - ||\boldsymbol{R}||^2 + |\boldsymbol{R}|^2 \geq (1/2-\varepsilon)||\boldsymbol{V}||^2. \end{aligned}$$

The third and fourth line follow because $\langle \boldsymbol{L} \cdot \boldsymbol{R} \rangle \leq ||\boldsymbol{V}||^2/4$ and $|\boldsymbol{R}|^2 \geq ||\boldsymbol{R}||^2$ respectively. ■

Time & Space Analysis. As in data summarization, we can analyze the skipped sketch solution for self-join estimation. The factor 4 approximation above uses space and query

[1]Note that well-known F_2 estimation methods including the classical Alon-Matias-Szegedy [3] sketches and others [7], provide $(1 \pm \varepsilon)$ guarantees, but have larger update times. With $O(1)$ update time independent of δ, previously, no guarantees were known for F_2 estimation.

Algorithm	Outputs $(1 \pm \varepsilon)\|\|\boldsymbol{L}\|\|^2 + \dots$
SCM$^{\text{c}}$	$\|\boldsymbol{R}\|_1^2$
SCM$^{\text{prod}}$	$\|\boldsymbol{L}\|_1 \|\boldsymbol{R}\|_1$
SCM^{F_0}	$\|\boldsymbol{R}\|_1^2 / \log \|\boldsymbol{R}\|_0$
SCM$^{\text{hash}}$	$\begin{cases} \|\|\boldsymbol{R}_{\text{h}}\|\|^2 + \|\boldsymbol{R}_{\text{h}}\|_1^2/\log\|\boldsymbol{R}_{\text{s}}\|_0 & \\ \quad +\|\boldsymbol{L}\|_1\|\boldsymbol{R}_{\text{h}}\|_\infty & \text{when } \varepsilon' < 2 \\ \text{SCM}^{F_0} & \text{when } \varepsilon' \geq 2 \end{cases}$

Table 1. Self-join size estimation procedures.

time $O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)$ as its non-skipped version [8]. Assuming all updates are unit counts, the amortized update time per element is $O\left((1-\varepsilon')\log\frac{1}{\delta}+\varepsilon'\right)$ for skipping rates $0 < \varepsilon' \leq 1$, which is an improvement over $O(\log\frac{1}{\delta})$ in the standard sketch. Further, the rate can be significantly improved with aggressive skipping, but with accuracy guarantees only on the expected case.

4.2.2 Heuristics

In addition to the constant factor approximation, we considered different heuristics to approximate $||\boldsymbol{R}||^2 + 2\langle \boldsymbol{L}\cdot\boldsymbol{R}\rangle$. As before, $||sk(\boldsymbol{L})||^2$ is used to approximate $||\boldsymbol{L}||^2$. The first heuristic is similar to the constant factor approximation algorihtm earlier, while the others keep an approximation to $|\boldsymbol{R}|_0$ (the zero-th frequency moment) of $\boldsymbol{R}$ as well. Unfortunately, all known methods to $\langle\varepsilon,\delta\rangle$-approximate $|\boldsymbol{R}|_0$ require $\Omega(\log(1/\delta))$ time per item. We therefore use a constant factor approximation to $|\boldsymbol{R}|_0$ with $O(1)$ update time per skipped item. The algorithm is that of Bar-Yossef et al [4], using only 72 buckets and no probabilistic amplification, which leads to a factor 4 approximation to $|\boldsymbol{R}|_0$.

Table 1 lists the constant factor approximation algorithm and the heuristics used. These heuristics differ from the former only on the query procedure. For a more detailed explanation on the choice of such heuristics, we refer the reader to a more comprehensive version of this work [5].

4.3. Heavy Hitters

We described an existing algorithm that used the underlying CM sketch with a heap of candidate heavy hitters in Section 2. We denote this as CMHeap. In this section, we present a new algorithm CM+MG for finding heavy hitters based on the CM sketch that is particularly designed to minimize the update time. Later, we show skipped sketches for both CMHeap and CM+MG. There are a number of other solutions to detecting heavy hitters on streams including that of Manku-Motwani [13] and its variants. Our skipping framework can be applied to those as well, and we expect similar improvements as in our sketch-based algorithms.

4.3.1 CM+MG (Count-Min sketch + Misra-Gries)

The idea behind the new CM+MG algorithm is to keep the identifier of the heavy hitter in each of the buckets in the CM sketch to avoid maintaining an expensive heap of candidates. This shifts the burden of heavy hitter identification from update time to the query time.

Misra and Gries [14] present a streaming algorithm that outputs the identifier of the item with absolute majority, if any; if there is no absolute majority, the identifier returned is arbitrary. It works by maintaining a few counters only. We will execute their algorithm within each bucket of the CM sketch to get the CM+MG algorithm. The main idea behind this is that a heavy hitter will most likely be a majority item in at least one bucket to which it is hashed.

The CM+MG algorithm stores the current item $item[j][h_j(i)]$ and a frequency counter $freq[j][h_j(i)]$ per bucket of the CM sketch besides the total count $count[j][h_j(i)]$ kept (the sum of all item counts that hashed into the bucket). All variables in the sketch are initialized to 0. CM+MG works as follows.

Update procedure: Upon seeing an update (i_t, c_t), it updates $count[j][h_j(i_t)]$ as the CM sketch does, and performs one of the following actions for each hash table j: **(A)** $freq[j][h_j(i_t)]$ += c_t if $i_t = item[j][h_j(i_t)]$; **(B)** $freq[j][h_j(i_t)]$ -= c_t if $i_t \neq item[j][h_j(i_t)]$ and $c_t \leq freq[j][h_j(i_t)]$; or **(C)** $freq[j][h_j(i_t)] = c_t - freq[j][h_j(i_t)]$ and $item[j][h_j(i_t)] \leftarrow i_t$ otherwise.

Query procedure: At query time, for each $count[i][j] \geq \phi|\boldsymbol{V}|$, it performs a point query $\hat{\boldsymbol{V}}[item[i][j]]$ and if $\hat{\boldsymbol{V}}[item[i][j]] \geq \phi|\boldsymbol{V}|$, that item is added to a heap. At the end of this linear search, it outputs all items from the heap after removing any duplicates. ■

Theorem 4. The CM+MG algorithm above $\langle\varepsilon,\delta,\phi\rangle$-approximates the heavy hitters problem using $O\left(\frac{1}{\varepsilon}\log\frac{N}{\delta}\right)$ memory space, $O\left(\log\frac{N}{\delta}\right)$ update processing time per item and $O\left(\frac{1}{\varepsilon}\log\frac{N}{\delta}\right)$ query time.

PROOF. Similarly to the analysis in [7], consider j hash tables and indicator variables $I_{i,j,k}$, which are 1 if $(i \neq k) \wedge h_j(i) = h_j(k)$, and 0 otherwise. Then, by pairwise independence of the hash functions, $\mathbf{E}[I_{i,j,k}] \leq \mathbf{Pr}[h_j(i) = h_j(k)] \leq 1/\text{range}(h_j) = \varepsilon/e$. Now, consider the non-negative random variables $X_{i,j}$ to be $\sum_{k=1}^{n} I_{i,j,k}\boldsymbol{V}[k]$. Then, $\mathbf{E}[X_{i,j}] \leq \sum_{k=1}^{n}\boldsymbol{V}[k]\mathbf{E}[I_{i,j,k}] \leq \frac{\varepsilon}{e}|\boldsymbol{V}| < \frac{\phi}{e}|\boldsymbol{V}|$ because $\varepsilon < \phi$. Thus, for a given heavy hitter i (note its count must be greater than $\phi|\boldsymbol{V}|$) and hash table j, the probability that the sum of all other items in the same bucket is $\geq \phi|\boldsymbol{V}|$ is $\mathbf{Pr}[X_{i,j} \geq \phi|\boldsymbol{V}|] \leq \mathbf{Pr}[X_{i,j} > e\mathbf{E}[X_{i,j}]] \leq 1/e$ by the Markov inequality. Thus $\lceil\log 1/\delta\rceil$ independent hash functions suffice to reduce the probability to δ. Since the CM sketch has a one-sided error, i.e. $\boldsymbol{V}[i] \leq \hat{\boldsymbol{V}}[i] \leq$

$\hat{\boldsymbol{V}}[i]+\varepsilon|\boldsymbol{V}|$, all heavy hitters are output. However, we must scale the parameter δ with N as in the analysis of CMHeap [7] to avoid including non-heavy hitters in the output. Because Misra-Gries requires only constant number of counters per bucket, the space is asymptotically the same as the CM sketch. Query time is the same as space due to the linear search. Finally, the update time is trivial from the algorithm and number of hash functions used. ■

In sum, CMHeap requires $O(\log(N/\delta)+\log(1/\phi))$ update time while CM+MG requires only $O(\log(N/\delta))$.

4.3.2 Skipped Sketches for Heavy Hitters

We create skipped versions SCMHeap and SCM+MG for the two algorithms. They both follow the skeleton from before where the standard sketch is either the CMHeap or CM+MG sketch and the skipping condition guarantees $|\boldsymbol{R}| \leq \varepsilon'|\boldsymbol{V}|$. The query procedure is left unchanged, i.e., the output of heavy hitters is the output of the heavy hitters in sketched vector $\boldsymbol{L}$ with threshold $\geq \phi|\boldsymbol{V}|$. The following theorem easily follows from the discussion thus far.

Theorem 5. SCMHeap and SCM+MG algorithms output all heavy hitter items such that $\boldsymbol{V}[i] \geq (\phi+\varepsilon')|\boldsymbol{V}|$ and no items such that $\boldsymbol{V}[i] \leq (\phi-\varepsilon)|\boldsymbol{V}|$, for approximation parameter $0 < \varepsilon < 1$ and skipping rate $0 < \varepsilon' < 1$. For SCMHeap, the update time per stream item is $O\left((1-\varepsilon')[\log(N/\delta)+\log(1/\phi)]+\varepsilon'\right)$ amortized. The query time and space bounds are $O\left(\log(1/\phi)\right)$ and $O\left(\frac{1}{\varepsilon}\log\frac{N}{\delta}\right)$, respectively. For SCM+MG, the update time per stream item is $O\left((1-\varepsilon')\log\frac{N}{\delta}+\varepsilon'\right)$ amortized, and the query time and space bounds are both $O\left(\frac{1}{\varepsilon}\log\frac{N}{\delta}\right)$.

As before, the theorems above apply to conservative skipping and use the worst-case scenario for a point query on $\boldsymbol{V}[i]$. However, suppose the amount skipped $\varepsilon'|\boldsymbol{V}|$ is uniformily distributed across all buckets of the CM sketch, then a point query estimation of $\hat{\boldsymbol{V}}[i]$ would guarantee, in expectation, that $\hat{\boldsymbol{V}}[i] \geq \boldsymbol{V}[i]-\varepsilon\varepsilon'|\boldsymbol{V}|$ (assuming the number of buckets is $1/\varepsilon$). Thus, heavy hitters would be output if $\boldsymbol{V}[i] \geq (\phi+\varepsilon\varepsilon')|\boldsymbol{V}|$ in the expected case, yielding a much tighter bound. Hence, if the distribution is close to uniform, we can use aggressive skipping to achieve better speed up as before.

5. Experimental Study

We carried out several experiments for data summarization, self-join size estimation, and all the heavy hitters algorithms described above along with their skipped versions.

On all comparisons, unless otherwise noted, we fixed the total desired approximation error $\varepsilon = 0.01\%$ which is reasonable for network monitoring applications at Sprint. The confidence parameter δ was set to 10%, which is common if few hash functions are desired ($\lceil\log 1/\delta\rceil = 4$ hash functions only). The combination of parameters created sketch sizes of around 450k. Each result shown is averaged out over more than 10 different runs with each run processing 10 million consecutive packets.

5.1. Input Data

The experiments were carried out on *real* and *synthetic* data. For the former, we incorporated our algorithms into Sprint's CMON monitoring system [16], which processes live network traffic data from the Sprint Internet backbone. For the latter, we generated the data based on known input distributions as described in detail below.

Real network traffic data. We consider Internet Protocol (IP) packet headers and content traces collected from an OC-48 link in the operational Sprint IP backbone [10] as our input streams. The Sprint CMON system is capable of processing traces offline as well as sniffing live IP packets online. Our streaming update procedures were implemented as statistical function modules within the system and are called back for every packet. A large portion of the performance study is done by offline processing, while the calibration of timing is done online. In the online capacity, we used the treplay [18] system to replay the traces at the same rate they were collected. This allowed us to simulate the live traffic environment in the lab, as well as produce repeatable input streams for the study.

Recall that in packet header analysis, a packet header contains the origin and destination IP addresses as well as its data length. For data summarization, the count $\boldsymbol{V}[i]$ is the number of bytes received by IP address i; in the heavy hitters problem consist of outputing the IP addresses that received over ϕ fraction of the entire data length $|\boldsymbol{V}|$ transferred during the same measured epoch. Self-join size estimation experiments were carried out only on synthetic data.

For the packet content analysis of Section 5.3 we consider the content of each packet as a string. We focus on the heavy hitters application in q-grams, that is, finding the most prevalent substrings of length q, for each 1-byte shift. We processed many traces, analyzing approximately 350 thousand packets on each. We set the q-gram size to 32-bytes to match the lowest possible packet size and the fingerprint size to 32-bits for convenience. We also set the approximate error for these experiments to $\varepsilon = 0.001\%$, the confidence parameter to $\delta = 10\%$, and the fraction parameter ϕ to be around 0.005% (see Section 2.2) because there weren't many heavy hitters in the traces tested.

Synthetic data. We created the synthetic data using standard routines to draw values from known skewed distributions. We tuned the distribution parameters as to mimic high-speed network packet-level traces. Our data stream generator created several traces of 1 million items each, drawn from a Zipfian distribution with varying skew parameter. The respective counts of items are drawn from a pareto distribution as a large fraction of items have typically small packet sizes and only a small fraction of them are large.

5.2. Packet Header & Synthetic Data Analysis

5.2.1 Data Summarization

Let the input stream $\boldsymbol{V}$ contain n distinct elements and $\boldsymbol{V}[i]$ represent the number of bytes sent to a destination IP address i. The accuracy error is calculated as the absolute difference of the actual and estimated values scaled by $|\boldsymbol{V}|$. We compare the accuracy of each algorithm by analyzing the 90th percentile of error (i.e., sorting the errors in descending order and getting the $(90/100)n$ error) because both sketches allow a probability of error of 10%. We also plot the maximum error for comparison purposes.

Figure 2(a) shows an accuracy comparison between CM and SCM algorithms summarizing real traffic data from Sprint IP backbone (we omit details for synthetic data for this application due to the similarity of results). As shown, the accuracy of SCM is comparable with that of the CM sketch on all metrics shown. For the 90% error metric, SCM even outperforms the CM accuracy results. This is because skipping tends to estimate a count of 0 for infrequent items, which is often a good estimate.

Regarding the update processing time, Figure 2(b) compares the relative gain in bits per second rate between CM and SCM, when using 4 and 10 hash functions for the sketch. Note that the performance gains are over 50% and 140% for skipping rates over 10, for 4 and 10 hash functions respectively. The plot also confirms the intuitive notion that the slower the update processing time of an underlying sketch, the greater the possibility of performance gain by using a skipped version of that sketch.

5.2.2 Heavy Hitters

Similarly to data summarization, we compare the accuracy (true positive and false negative rates) and performance of all heavy hitter algorithms and their skipped versions. We measure the true positive rate by the *precision* of the set of heavy hitters, i.e., the number of claimed heavy hitters by the algorithm over the correct number of heavy hitters. Similarly, the false negative rate is measured by the *recall* of the set of heavy hitters, i.e., the number of claimed heavy hitters that are indeed real heavy hitters divided by the real heavy hitters. For both rates, offline analysis of the same trace was done to extract the correct list of heavy hitters.

Figure 2(c) and 2(d) plot the precision rate of all heavy hitters algorithm and their skipped version on real traffic data. We omit the recall rate in the interest of space since the decay in accuracy as the skipping rate increases is very similar. The fraction ϕ of the norm was set to 0.1%, or 0.001, yielding on average 150 heavy hitters in over 10 different traces tested, each with over 10 million packets.

Observe that the precision rate of the skipped method remains at reasonable levels, 85% accurate, for large skipping rates such as 200. Similar to the data summarization estimation, we compare the sketch processing rate as shown in Figure 2(d). The improvements exceed 50% for skipping rates $\varepsilon' \geq 5$ and over 100% for $\varepsilon' \geq 10$.

5.2.3 Self-Join Size estimation

To study the self-join size estimation, we use our generated synthetic data. We compare the accuracy and performance of the heuristics described in Section 4.2.2. Figure 3(a) shows the accuracy comparison with varying skipping rate ε'. The relative error is the absolute difference between the estimated value $||\tilde{\boldsymbol{V}}||^2$ and the actual value $||\boldsymbol{V}||^2$ over $||\boldsymbol{V}||^2$. First, note that the estimates for the first two heuristics deteriorate quicker than the others as the skipping rate ε' increases. In other words, as skipped vector $\boldsymbol{R}$ plays a larger role for the estimate of $||\boldsymbol{V}||^2$, neither $|\boldsymbol{L}|_1|\boldsymbol{R}|_1$ nor $|\boldsymbol{R}|_1^2$ becomes a good estimate. On the other hand, estimating $|\boldsymbol{R}|_0$ (used in the last two heuristics) helps significantly, specially for large skipping rates. Remarkably, our best heuristic $\mathsf{SCM}^{\mathsf{hash}}$ gives estimates below 25% error for the entire range of skipping rates tested!

The performance improvement gained by skipping elements is shown on Figure 3(b). Logically, the heuristics depending solely on $|\boldsymbol{R}|_1$ demonstrate the larger gains because the overhead per skipped item is very small (only one counter is needed). Moreover, the other heuristics, which require keeping track of $|\boldsymbol{R}|_0$, also show sizable gains.

Another interesting plot is shown on Figure 3(c). The x-axis varies the Zipfian z parameter (from less to highly skewed distributions), while the skipping rate is fixed at 0.5. First, the plot shows that all heuristics behave similarly for $z \geq 1.5$, staying below error levels of 20%. Moreover, the algorithm $\mathsf{SCM}^{\mathsf{c}}$, for which we have proven bounds, perfoms nicely for the entire range of skipping rates.

5.3. Packet Content Analysis

Our experiments focus on obtaining the heavy hitters list of q-grams in the packet content stream. This application is quite challenging compared to packet header analysis because each packet now generates multiple updates to the stream. In networking, authors in previous work [1, 15] have studied the problem of determining the most prevalent substrings in the network, motivated by worm and virus detection. A popular approach is to use uniform *random sampling* to extract a subset of all q-grams. Since this is rarely

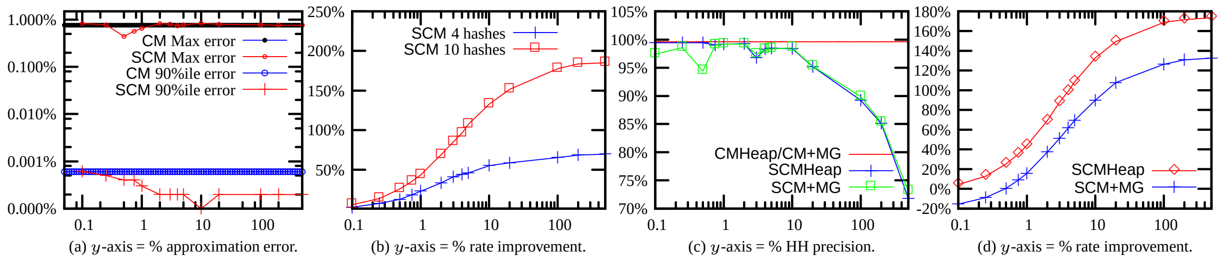


Figure 2. Data summarization (a and b) and Heavy Hitters (c and d) accuracy and performance. The x-axis varies the skipping rate ε'.

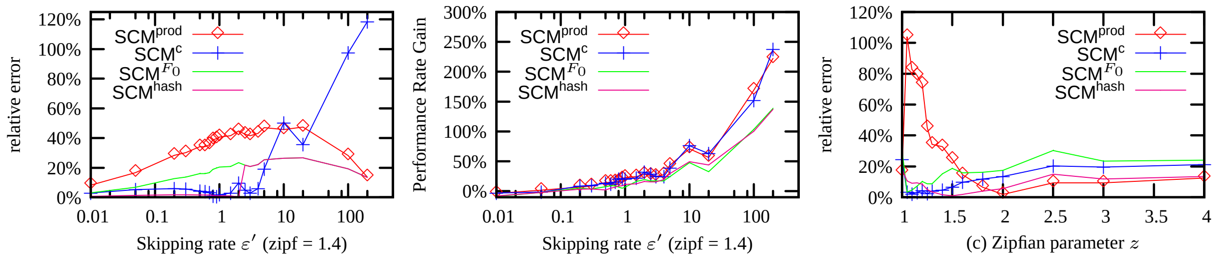


Figure 3. Accuracy (a and c) and performance (b) comparison of the heuristics listed on Table 1. Plot (c) shows the accuracy behavior as the distribution gets skewed.

accurate, the authors designed *value sampling* to select only those q-grams whose hash matches a certain bit mask.

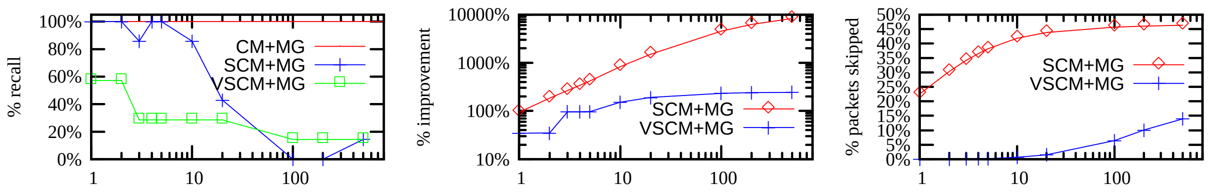
We show that a skipping-based approach susbtantially improves on the performance of value sampling for this problem. We compare the accuracy of the CM+MG and SCM+MG, its skipped version, as well as VSCM+MG based on value sampling. We implement the fingerprint method [6], which allows efficient q-gram hashes computations over a sliding window. The item/count stream pair comprise of these hashes and a value of 1 respectively. We compare the recall (true positive) rate. Since VSCM+MG requires a pre-determined mask, we averaged the results out over several runs, choosing a random mask each time.

Figure 4 shows the accuracy and performance of the algorithms tested. Note that the accuracy of our skipping method SCM+MG is consistently better than value sampling VSCM+MG specifically when $\varepsilon' < 50$. However, for some very large skipping rates, $\varepsilon' > 100$, the story is reversed. Value sampling has a better chance to extract a few set of heavy hitters since it tracks all occurences of q-gram hashes that matched its mask. However, it comes at a price. Note that the VSCM+MG algorithm falls significantly short in performance compared to our method (note the logarithmic scale!). This is expected because the value sampling approach must compute a fingerprint and match it against a mask for *every* q-gram before deciding whether to sketch it or not. Moreover, the plot shows that the rate improvement increases *linearly* as the skipping rate increases. Additionally, Figure 4(c) compares the number of packets skipped. Note that our proposed approach SCM+MG skips *significantly* more packets than VSCM+MG.

To exemplify, for a skipping rate ε' of only 10, our method achieves accuracies over 80% with 45% of skipped packets. Moreover, the packet processing rate increased from around 5k to 50-440k packets/s (for ε' ranging from 10 to 500 resp.), a 10- to 90-fold speed-up. These experiments show for the first time that packet content processing can indeed be done at near OC-48 line speeds (note that due to the tri-modal distribution of packet sizes in the internet, most packets —around 40% —are smaller than 40 bytes and thus usually irrelevant to deep content analysis).

6. Related Work

There has been a lot of work on summarization of data streams in small space, generating typically either selection- or sketch-based approaches. Selection-based algorithms such as [12, 13] process each item into a data structure (say a heap) and have to periodically prune them; their amortized update times are quite efficient, still, they do not skip over items the way our methods do leading to our improvements. The same difficulty is faced by selection-based algo-



(a) Recall (true positives) comparison. (b) Improvement over CM+MG. (c) Number of packets skipped.

Figure 4. Packet contents heavy hitters accuracy and performance comparison.

rithms [1, 15] developed for networking streaming analysis.

The idea of skipping past the input is quite natural. It has been previously used in *reservoir sampling* by Vitter [17] where a probabilistic calculation determines how many items may be ignored for the next sample. This corresponds to unit count increments in our case. Our skipping methods are more general and deterministic. In particular, they are based on the various norms of the sketched and skipped parts of the stream, and are tuned for particular analysis (heavy hitters and self-join estimation) for which reservoir sampling does not yield accurate results.

7. Concluding Remarks & Extensions

We have presented a framework to skip past streams to achieve several-fold higher processing rates while still providing strong accuracy guarantees for post-hoc analyses including data summarization and point queries, estimating the self-join size of a relation, and finding the heavy hitters. We validated our proposed algorithms with experiments on synthetic data and live network traffic from Sprint's IP backbone. Our experiments are the first known that analyze the stream of packet contents rather than just the headers at backbone speeds. The skipping framework we have introduced of using constant processing time per-item for norm estimation to guide the reduction of data for sketching algorithms can be applied easily to other underlying summarization techniques (such as the sampling algorithm for quantiles [12], the selection algorithm for heavy hitters [13], and the distinct sampling [11] algorithm) and analyses of interest (such as quantile estimation, detecting changes using deltoids, and finding anomalies such as deviants.). Moreover, one can modify our approach to the case where streaming updates comprise of inserts *and* deletes.

References

[1] P. Akritidis, K. Anagnostakis, and E. Markatos. Efficient content-based fingerprinting of zero-day worms. In *Proceedings of the ICC*, May 2005.

[2] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of PODS*, pages 10–20. ACM Press, June 1999.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of STOC*, pages 20–29, 1996.

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proceedings of RANDOM*, London, 2002. Springer-Verlag.

[5] S. Bhattacharrya, A. Madeira, S. Muthukrishnan, and T. Ye. How to (accurately) skip past streams. Sprint ATL Research Report RR06-ATL-031609, Sprint ATL, March 2006.

[6] A. Broder. Some applications of rabin's fingerprinting method. In *Methods in Communications, Security, and Computer Science*, pages 143–152. Springer-Verlag, 1993.

[7] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *Proceedings of LATIN*, pages 29–38, 2004.

[8] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *SDM*, 2005.

[9] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proceedings of SIGMOD*, pages 647–651, New York, NY, USA, 2003. ACM Press.

[10] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the sprint IP backbone. *IEEE Network*, 2003.

[11] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *The VLDB Journal*, pages 541–550, 2001.

[12] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of VLDB*, August 2002.

[13] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of VLDB*, 2002.

[14] J. Misra and D. Gries. Finding repeated elements. In *Sci. Comput. Programming*, pages 143–152, November 1982.

[15] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of OSDI*, pages 45–60. USENIX, 2004.

[16] K. To, T. Ye, and S. Bhattacharyya. CMON: A general-purpose continuous ip backbone traffic analysis platform. Research Report RR04-ATL-110309, Sprint ATL, 2004.

[17] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

[18] T. Ye, D. Veitch, G. Iannaccone, and S. Bhattacharyya. Divide and conquer: PC-based packet trace replay at OC-48 speeds. In *Tridentcom*, Trento, Italy, February 2005.