The Eternal Sunshine of the Sketch Data Structure

Xenofontas Dimitropoulos
IBM Zurich Research Laboratory
xed@zurich.ibm.com

Marc Stoecklin
IBM Zurich Research Laboratory
mtc@zurich.ibm.com

Paul Hurley
IBM Zurich Research Laboratory
pah@zurich.ibm.com

Andreas Kind
IBM Zurich Research Laboratory
ank@zurich.ibm.com

Abstract

In the past years there has been significant research on developing compact data structures for summarizing large data streams. A family of such data structures is the so-called *sketches*. Sketches bear similarities to the well-known Bloom filters [2] and employ hashing techniques to approximate the count associated with an arbitrary key in a data stream using fixed memory resources. One limitation of sketches is that when used for summarizing long data streams, they gradually saturate, resulting in a potentially large error on estimated key counts. In this work, we introduce two techniques to address this problem based on the observation that real-world data streams often have many transient keys that appear for short time periods and do not re-appear later on. After entering the data structure, these keys contribute to hashing collisions and thus reduce the estimation accuracy of sketches. Our techniques use a limited amount of additional memory to detect transient keys and to periodically remove their hashed values from the sketch. In this manner the number of keys hashed into a sketch decreases, and as a result the frequency of hashing collisions and the estimation error are reduced. Our first technique in effect slows down the saturation process of a sketch, whereas our second technique completely prevents a sketch from saturating[1]. We demonstrate

[1]The phrase "eternal sunshine" in the title reflects that our techniques mitigate or halt the saturation process of a sketch.

the performance improvements of our techniques analytically as well as experimentally. Our evaluation results using real network traffic traces show a reduction in the collision rate ranging between 26.1% and 98.2% and even higher savings in terms of estimation accuracy compared to a state-of-the-art sketch data structure. To our knowledge this is the first work to look into the problem of improving the accuracy of sketches by mitigating their saturation process.

1 Introduction

In the past several years there has been significant research on developing data structures for summarizing massive data streams and on computing desired statistics from the summaries created. One statistic that is commonly sought is the total count associated with a key in a data stream, i.e., the sum of the values that have appeared for a key. Sketches are summary data structures that can estimate the count associated with a key, a process that is also called answering *point queries*. In networking, sketches have known applications in estimating the size of the largest traffic flows in routers [5] and NetFlow/IPFIX collectors [9], in detecting changes in traffic streams [10, 16], in adaptive traffic-sampling techniques [12], and in worm fingerprinting [17]. More generally, sketches find applications in systems that require online processing of large data sets.

Sketches are a family of data structures that use the same underlying hashing scheme for summarizing data. They differ in how they update hash buckets and use hashed data to derive estimates. Among the different sketches, the one with the best time and space bounds is the so called *count-min* sketch [4]. Sketches bear similarities to the well-known Bloom filters in that both employ hashing to summarize data, have fixed memory requirements, and provide approximate answers to the queries they are designed for.

An inherent problem with sketches is that keys may collide, namely, hash to the same bucket, producing errors in the estimated counts. This particularly affects long data streams, for which sketches gradually saturate, leading to more collisions and to lower estimation accuracy. In this work we introduce two techniques to address this problem. Our techniques are based on the observation that for real-world data streams, the saturation of sketches is driven by a large number of transient keys that are hashed into the data structure and produce many collisions. These transient keys often are not interesting either because they have a small size or because they become inactive after a certain point in time. The techniques effectively detect such keys and remove their hashed values from the sketch, decreasing in this way the number of collisions and the estimation error. The first technique uses a small additional memory to detect certain transient keys and to remove their hashed values from the sketch. Transient keys that are not detected remain in the sketch, and thus sketch saturation slows down but does not halt completely. The second technique uses a larger additional memory to effectively halt the saturation process by posing an upper bound on the number of keys that are present in the data structure

at any given point in time.

Our techniques are useful for applications that are not interested in transient keys, but rather focus on the active or recent keys of a data stream. In the context of network monitoring and management, a number of such applications exist that, for example, monitor the active flows in a network, identify important events, like anomalies, and manipulate monitored active flows, namely, for traffic engineering purposes.

We describe and analyze our techniques with respect to the state-of-the-art count-min sketch[2]. Using traffic traces from a large enterprise network, we illustrate that, compared with the count-min sketch, our techniques result in lower collision rate and increased estimation accuracy.

We structure the remainder of this paper as follows. In the next section we provide background information on sketches and introduce our notation. Then, in Section 3 we summarize related research. In Section 4 we introduce our two techniques. We evaluate the performance of our techniques and compare it with that of existing schemes in Section 5. Finally, we conclude our paper and outline future directions in Section 6.

2 Preliminaries

In this section we first outline the data-streaming model that forms the input to a sketch data structure, and then provide a detailed description of the count-min sketch.

2.1 Data-streaming model

A data stream I of running length n is a sequence of n tuples. The t-th tuple is denoted as (k_t, u_t), where k_t is a key used for hashing and u_t is a value associated with the key:

$$I = (k_0, u_0), (k_1, u_1), \ldots, (k_t, u_t), \ldots, (k_{n-1}, u_{n-1})$$

The value u_t is a positive number, which results in monotonically increasing key counts. This data-streaming model is called the cash register model [15].

2.2 The count-min sketch

The count-min sketch is a two-dimensional array $T[i][j]$ with d rows, $i = 1, \ldots, d$, and w columns, $j = 1, \ldots, w$. Each of the $d \times w$ array buckets is a counter that is initially set to zero. The rows of T are associated with d pairwise independent hash functions $h_1, \ldots, h_d$ from keys to $1, \ldots, w$. In Figure 1 we illustrate the count-min sketch data structure.

[2]Though we focus on the count-min sketch, our techniques are generic and can be used with other sketch variants as well.

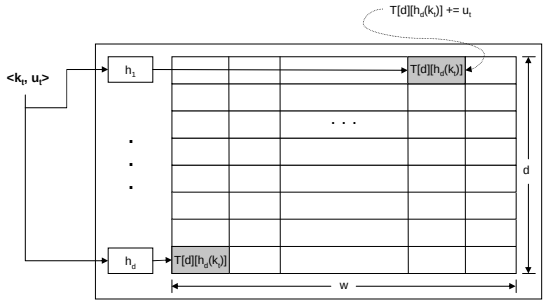

Figure 1: The count-min sketch data structure.

Update procedure. An incoming key k_t is hashed with each of the hash functions $h_1, \ldots, h_d$, and the value u_t is added to the counters $T[i][h_i(k_t)], i = 1, \ldots, d$.

Estimation procedure. Given a query for key k_t, the count-min sketch returns the minimum of the counters to which k_t hashes, so that the estimated count for a key k_t becomes $\min_{i=1,\ldots,d} T[i][h_i(k_t)]$.

Discussion. First, note that multiple keys may hash to the same bucket and thus the count of a bucket may *overestimate* the true size of a key. For this reason the estimation procedure returns the minimum value of the counters a key is hashed to. Assume for a moment that the count-min sketch has a single row, i.e., $d = 1$. If two keys hash to the same counter, the counter will overestimate the count of the two keys. By adding more hash buckets in each row, i.e., increasing the number of columns w, it becomes less likely that two keys collide and thus the number of collisions and the resulting *expected error* decrease. Accordingly, the parameter w primarily manipulates the expected error of the count-min sketch. If we increase the number of rows d of the sketch, then the probability that a given key has a large error, i.e., collides with many other keys, decreases. This is because as we increase the number of rows d, it becomes less likely that a given key collides with *many* other keys in all of the d buckets it is hashed to. Therefore, the parameter d mainly controls the probability that a key has a large error or, in other words, the tail of the error distribution.

3 Related work

A number of sketch variants have been introduced in recent years [1, 7, 8, 6, 4, 5, 10, 3]. These sketches use the same underlying array of counters for summarizing data streams and differ in how they update counters and use counters' values to derive estimates. Among the different sketches, the count-min sketch by Cormode and Muthukrishnan [4] has the tightest time and space bounds[3].

[3]For a comprehensive analysis on the differences between sketches and the resulting performance, the reader is referred to [4].

Sketches find a large number of applications in networking problems and in particular in analyzing streams of traffic data for extracting useful statistics, detecting anomalies, accounting, and other applications. The work by Estan and Varghese [5] introduced a data structure called *multi-stage filter*, which is a sketch variant designed to find traffic flows with size above a desired threshold. Krishnamurthy *et al.* [10] introduced the *k-ary sketch* for answering point queries and used this sketch to detect changes in traffic streams and signal potential anomalies. The use of sketches for identifying changes in traffic streams has also been explored by Schweller *et al.* [16] and Li *et al.* [14]. Kumar *et al.* [11] used a sketch with a single array of counters, i.e., $d = 1$, in combination with an expectation maximization method to estimate the distribution of flow sizes. The same sketch was also used by Kumar and Xu [12] in their design of an improved traffic-sampling technique offering better statistics than naive random sampling. The work by Zhang *et al.* [19] described a sketch-based solution to the problem of identifying hierarchical heavy hitters, i.e., high-volume clusters of IP addresses and/or other hierarchical attributes. Finally, the work by Lee *et al.* [13] proposed a generic method using linear least squares for improving the estimation accuracy of sketches. To the best of our knowledge, our work is the first to focus on the problem of improving estimation accuracy of sketches by mitigating their saturation.

4 Ageing sketches

In this section, we introduce the two techniques for mitigating the saturation process of sketches, but first describe certain characteristics of real-world data streams that we exploit in our techniques. Our main focus is network traffic data, thus we augment our description with examples based on traffic traces.

A common characteristic of real-world data streams is that they contain a large number of transient keys, i.e., keys that are active for a certain period and then become inactive. For example, network traffic data streams include many transient keys that result from short-lived http flows. In Figure 2, we plot the distribution of flow durations in a two-hour traffic trace collected from an enterprise network. We observe that a large portion of the flows are transient having a short duration, i.e., the duration of 78% of the flows is equal to or lower than one second. For many sketch applications, transient flows are not interesting, as they become inactive after a point in time and also because of their small size[4]. In addition, transient flows result in many hashing collisions and decrease the estimation accuracy of sketches. For these reasons, our techniques remove transient flows from sketches to prevent or slow down their saturation.

Both techniques split the input data stream into windows of a fixed number l

[4]One may argue that knowing small flows is important for certain applications such as identifying port scanning. Port scanning and similar behaviors are detected by counting the number of distinct destinations to which a source is communicating. Consequently, identifying port scanning does not require keeping state for small flows, which is an inefficient approach, but relies instead on distinct counting techniques [18].

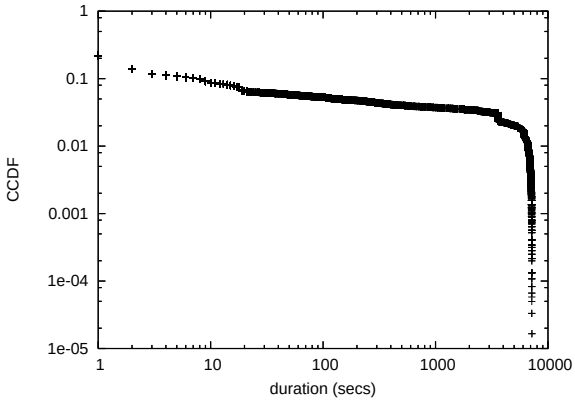

Figure 2: Complementary Cumulative Distribution Function (CCDF) of the duration of flows in a two-hour trace.

of (k_t, u_t) pairs. Then, at the end of each window, they remove structure state for keys that have been inactive, i.e., they age old state.

4.1 Bit marking

4.1.1 Description

The first technique, called *bit marking*, associates a bit with each counter in a sketch yielding an array of bits $b[i][j]$, where $i = 1, \ldots, d$ and $j = 1, \ldots, w$. Initially all bits are set to zero. When a counter is updated, its bit is set to one to reflect that the counter was active during the window. At the end of each window, all counters that still have their bit set to zero, i.e., the counters that were inactive during the last window, are reset. In this way, the values of the keys that have been inactive are removed from the sketch. Finally, all bits are set to zero to monitor the activity of the next window. In Algorithm 1 we summarize the operation of bit-marking.

4.1.2 Discussion

The extra bit monitors the activity of counters and using the window length l identifies counters that have been inactive for a desired period. These counters are then reset removing information for inactive keys. Most inactive keys are transient, as the latter are prevalent in real-world and network traffic data streams. All inactive keys are not necessarily removed from the sketch. An inactive key may hash to the same bucket with an active key and thus remain in the sketch. For this reason, bit-marking slows down the saturation of sketches but does not halt it completely.

Bit-marking disregards information for inactive flows and, thus, is useful for applications requiring statistics on active flows. A number of such applica-

Algorithm 1: Bit marking (update procedure)

```
Input: Stream of tuples (k_t, u_t)
Input: Window length l
Input: Number of sketch rows d and columns w

counter = 0;
forall (k_t, u_t) do
    for i = 1 to d do
        T[i][h_i(k_t)] += u_t;
        b[i][h_i(k_t)] = 1;
        counter += 1;
        if counter == l then
            counter = 0;
            for k = 1 to d do
                for n = 1 to w do
                    if b[k][n] == 0 then
                        T[k][n] = 0;
                    else
                        b[k][n] = 0;
                    end
                end
            end
        end
    end
end
```

tions exist that in the context of network traffic data streams mainly relate to monitoring flows, identifying interesting events, and manipulating flows. Specially, bit-marking can be used for monitoring and reporting active flows to an administrator, for identifying events involving active flows, like flash crowds, network anomalies, and attacks, and for manipulating active flows, namely, as part of traffic engineering. On the other hand, bit-marking is not useful for applications requiring size estimates on old flows like billing and network forensics applications.

The main advantages of bit-marking are that: 1) it is very simple to implement, 2) it requires only a small additional space of a single bit per counter, and 3) it substantially improves the performance of sketches, as we show in Section 5.

4.1.3 Analysis

If two distinct keys k and k' collide under hash function h_i, then the indicator variable $I^c_{k,i,k'}$ is 1, otherwise 0. The hash functions h_i are pairwise independent, from which we get:

$$E(I^c_{k,i,k'}) = Pr[h_i(k) = h_i(k')] = \frac{1}{w}.$$

In the worst case, a bucket is reset after $2l - 1 \approx 2l$ consecutive arrivals that result in no collision. Thus, the probability of reseting an inactive key in a single hash table has lower bound:

$$Pr[\text{reset inactive key}] \geq (1 - 1/w)^{2l}. \tag{1}$$

Conversely, the probability of an inactive key not being reset in all d hash tables has upper bound:

$$Pr[\forall i \text{ inactive key stays}] \leq (1 - (1 - 1/w)^{2l})^d.$$

Thus, the last equation gives a conservative estimate of the probability of an inactive key staying in the sketch. As an example, if $d = 16$, $w = 8192$, and $l = 5{,}000$, then the probability of an inactive key staying in the sketch is smaller than or equal to 0.0037.

The error associated with a key of the count-min sketch data structure is $X_{k,i} = \sum_{t=0}^{n-1} I^c_{k,i,k_t} u_t$. The count-min sketch paper [4] showed that the expected value of $X_{k,i}$ is:

$$E(X_{k,i}) = E(\sum_{t=0}^{n-1} I^c_{k,i,k_t} u_t) \leq \sum_{t=0}^{n-1} u_t E(I^c_{k,i,k_t}) \leq \frac{\| \mathbf{u}(0, n-1) \|_1}{w}, \tag{2}$$

where $\| \mathbf{u}(0, n-1) \|_1$ denotes the $L1$-norm of the data stream range between indices 0 and $n-1$, which, in this case, coincide with the complete data stream.

Assume an active key k that enters a sketch using bit-marking at the data stream index $t = f$. Before f, inactive keys that hash into the same bucket

with k may have been removed. After f, no such removals are possible, since the key k is active. Thus, the error accumulated after the index f is described by Equation 2 using f and $n-1$ as the range boundaries. To analyze the error in the data stream range between 0 and $f-1$, we use the indicator variable I_k^s that is equal to 0 if a key is inactive and removed; and equal to 1 otherwise. Let r denote the fraction of inactive keys, then using Equation 1 the expected value of I_k^s is:

$$\begin{aligned} E(I_k^s) &= \Pr[\text{key is inactive}]\Pr[\text{inactive key stays}] + \Pr[\text{key is active}] \\ &= r\Pr[\text{inactive key stays}] + (1-r) \leq 1 - r(1-1/w)^{2l}. \end{aligned}$$

The error during the first range of the data stream is $Y_{k,i} = \sum_{t=0}^{f-1} I_{k,i,k_t}^c I_{k_t}^s u_t$. The expected value of $Y_{k,i}$ is:

$$\begin{aligned} E(Y_{k,i}) &= E(\sum_{t=0}^{f-1} I_{k,i,k_t}^c I_k^s u_t) \leq \sum_{t=0}^{f-1} u_t E(I_{k,i,k_t}^c) E(I_k^s) \\ &\leq \frac{(1-r(1-1/w)^{2l}) \parallel \mathbf{u}(0,f-1) \parallel_1}{w}. \end{aligned} \tag{3}$$

Combining Equation 3 for the first portion of the data stream and Equation 2 for the last portion, we get the expected value for the total error $Z_{k,i}$ associated with an active key:

$$E(Z_{k,i}) \leq \frac{(1-r(1-1/w)^{2l}) \parallel \mathbf{u}(0,f-1) \parallel_1}{w} + \frac{\parallel \mathbf{u}(f,n-1) \parallel_1}{w}.$$

The $L1$-norm of a data stream portion between two indices a and b can be approximated as a fraction of the $L1$-norm of the complete data stream, i.e., $\parallel \mathbf{u}(a,b) \parallel_1 \approx (b-a+1)/n \parallel \mathbf{u}(0,n-1) \parallel_1$. Making this approximation, the last equation takes the simpler form:

$$E(Z_{k,i}) \leq \frac{1 - \frac{f}{n} r(1-1/w)^{2l}}{w} \parallel \mathbf{u}(0,n-1) \parallel_1$$

Compared to the expected error of the plain count-min sketch (Equation 2), bit-marking introduces an improvement by at least a factor of $\frac{f}{n} r(1-1/w)^{2l}$. Note that for long real-world data streams both $\frac{f}{n}$ and r are close to 1.

4.2 Sliding window

4.2.1 Description

The second technique, called sliding window, uses m identical sketches T_i, $i = 1, \ldots, m$, each having the same dimensions and hash functions. Initially, all

sketches are placed into a FIFO queue. The technique splits a window oflength l into m segments of length l/m and uses one sketch for each segment to summarize data received during the segment. At the beginning of each segment, a sketch is fetched from the FIFO queue and the counters of the sketch are initialized to zero. Then, incoming data is entered into to the sketch until the end of the segment. At the end of the segment, the sketch is inserted at the end of the FIFO queue and a new segment starts with fetching a new sketch from the top of the queue. Given a query for the count of a key k_t, our technique returns the sum of the estimated counts for k_t from the m sketches. We summarize the technique in Algorithm 2.

Algorithm 2: Sliding window (update procedure)

```
Input: Stream of tuples (k_t, u_t)
Input: Window length l
Input: Number of segments m
Input: Number of sketch rows d and columns w

counter = 0;
segmentlength = ⌊l/m⌋;
for i = 1 to m do push(fifoqueue,T_i);
forall (k_t, u_t) do
    T = fifoqueue[0];
    counter += 1;
    for i = 1 to d do T[i][h_i(k_t)] += u_t;
    if counter == segmentlength then
        counter = 0;
        tmp = pop(fifoqueue);
        push(fifoqueue,tmp);
        T = fifoqueue[0];
        for k = 1 to d do
            for n = 1 to w do T[k][n] = 0;
        end
    end
end
```

4.2.2 Discussion

Our technique implements a sliding window of length l and summarizes the data that arrived during the window in m sketches. The sliding window is implemented by splitting a data stream into equal-size segments and by keeping information only for the last m segments. The length of each segment determines how smoothly the window slides. For each segment, our technique keeps a distinct sketch. The sketches have identical hash functions, which reduces the number of hash operations required. This is because once a key is hashed, the resulting value can be used to index the appropriate buckets in all m sketches.

Compared to bit-marking, the sliding window technique is more aggressive in removing old information. Whereas bit-marking removes information for flows that have become inactive, the sliding window removes all information that arrived before the beginning of the window. Bit-marking effectively removes transient keys as these keys shortly become inactive. The sliding-window approach is more general in that it removes all old information along with transient keys. The sliding window and bit-marking techniques differ in their possible applications. Whereas bit-marking can be used for applications requiring information on the active keys, the sliding window can be used for applications that are interested on the last portion of a data stream. In practice, there is a wide range of such applications that relate to online network monitoring.

4.2.3 Analysis

The sliding window technique keeps information only for the last portion of a data stream. Otherwise the used sketches are identical to the count-min sketch. In this case the error associated with a key k is $X_{k,i} = \sum_{t=n-l}^{n-1} I^c_{k,i,k_t} u_t$. The expected value of $X_{k,i}$ is:

$$E(X_{k,i}) = E(\sum_{t=n-l}^{n-1} I^c_{k,i,k_t} u_t) = \sum_{t=n-l}^{n-1} u_t E(I^c_{k,i,k_t}) = \frac{\parallel \mathbf{u}(n-l, n-1) \parallel_1}{w}.$$

Comparing the last equation with Equation 2 for the unmodified count-min sketch, we find an improvement in expected estimation error by a constant factor of $\frac{\|\mathbf{u}(0,n-1)\|_1 - \|\mathbf{u}(n-l,n-1)\|_1}{\|\mathbf{u}(0,n-1)\|_1}$, which can be approximated by $\frac{n-l}{n}$.

4.3 Parameter Configuration

All sketches have two main parameters, the number of rows d and columns w of the two-dimensional array. One possibility is to set d and w based on the theoretical analysis in [10] that derived parameter values from a desired upper bound on estimation accuracy. This approach is rigorous and straight-forward to use. On the other hand, the resulting values for d and w are independent of the characteristics of the input data stream and thus are likely to be conservative. An alternative approach is to test different choices of d and w on data from the target environment and select the configuration that yields the best performance. This method dimensions a sketch taking into account the characteristics of the data stream and, thus, is less conservative than the previous one. It results in using fewer memory resources for achieving a desired accuracy. This benefit, though, comes at the cost of necessary testing experiments.

The two techniques use the window length l parameter. In bit-marking, the window length is the period of inactivity after which a key is removed from the data structure. In the second technique, l is the length of the sliding window. In both cases, the window length is associated with the removal of old information from the sketch. For this reason, the selection of l depends on

how an application identifies old information. For example, a network traffic profiling application might be interested in finding the largest traffic flows in the last 30 minutes. For this purpose, it could use the sliding-window technique and select l to keep information for 30 minutes. In addition, an online traffic engineering system might need to monitor and re-route large flows, however it might afford losing information for flows that have become inactive. For this goal, the application could use bit-marking and set l say to 5 minutes to ignore information for inactive flows.

The sliding-window technique has one additional parameter, the number of segments m into which a window is split. The number of segments determines how smoothly the window slides. A large value of m result in the window sliding more smoothly at the cost of additional memory. Thus, the choice of m depends on the available memory resources and on the sliding smoothness that an applications needs to deliver to the end-user.

Overall, the parameters l and m are determined by the specific requirements of an application using bit-marking or the sliding-window technique. On the other hand, the parameters d and w affect the estimation accuracy and the memory consumption of the data structure and could be derived either analytically or through testing experiments.

5 Evaluation

5.1 Data Sets

In our experiments we use a trace of NetFlow records collected from the IBM Research campus at Zurich. The campus hosts more than three hundred employees and at least as many networked workstations. NetFlow statistics are collected on the two border routers of the campus and reported to a NetFlow collector in our lab. For our experiments, we use a two-hour trace collected on November 5, 2006. The trace contains 8,850 NetFlow version 9 packets, 271,302 flow records, 120,504 unique flows (keys), 2,985 unique source IP addresses, and 67,235 unique destination IP addresses. We parse the dataset and extract a stream of flow records that we use as input for our experiments. The key associated with each flow record is the 4-tuple: source IP address, source port number, destination IP address, and destination port number. The value of a key is the number of packets reported for a flow.

5.2 Evaluation Metrics

Metrics. We define the *collision rate* as the average number of collisions per flow entered into a sketch, where a collision is the event of hashing a key into a bucket that includes information on at least one other key. For a sketch with d rows the maximum collision rate is equal to d, as in the worse case each key collides in all the d buckets it is hashed to. For this reason, we normalize the collision rate by d to enable the comparison of the collision rates of different

sketches with different values of d. The collision rate is useful for monitoring the saturation process of a sketch.

Secondly, we analyze the information that is removed from and that stays in the sketch data structure. Since our techniques are truncating information, it is necessary to understand how much information is removed, and what are the qualitative differences between the information that stays in the sketch and the information that is removed. For this purpose we compute different statistics on the flows that are removed and that remain in the sketch.

Next, we evaluate the estimation accuracy of our techniques and compare to the estimation accuracy of the unmodified sketch data structure. In evaluating estimation accuracy, it is necessary to take into account that our techniques truncate old information and that the useful data in the three approaches are different. In particular, bit-marking removes information for inactive flows and thus can only be used for estimating the size of active flows. On the other hand, the sliding-window technique removes all information that arrived before the beginning of the window. For this reason, it can only be used for estimating the number of flow packets or bytes that appeared during the window. Finally, the original sketch data structure does not remove any information and can be used for estimating the size of all flows in the data stream. For these reasons, we compare 1) the true and estimated sizes of the flows that were not removed with the bit-marking technique; 2) the true and estimated number of flow packets that appeared during the sliding window; and 3) and the true and estimated sizes of all flows in the data stream. In all cases we measure the size of a flow in terms of packets, periodically compute the estimation error, and construct a sample of error values. In characterizing the constructed samples we had to deal with that the error distribution was highly skewed. In particular, most error values in a sample were equal to zero, whereas some values (especially in the case of the unmodified sketch) were as high as 108,540,800%[5]. In order to compare the periodically constructed samples, we used the 95-percentile filtering technique[6], removed from each sample very large error values, and computed the mean error of each sample. Finally, we compare how the mean error of the three approaches changes with time.

5.3 Experimental Results

In our experiments we explored different settings of the sketch parameters w and d that yield the same memory consumption. As described in each experiment bellow, we report results on the configuration settings that had the best and worst performance. In addition, we set the application-dependent parameters l and m to 10,000 tuples and 5 segments, respectively. The window length

[5]Such outliers are the result of hashing collisions between flows of small and of very large size. For example, if a small flow, say of size one packet, collides in all the buckets it is hashed to with large flows, say of size larger than 20,000 packets, then the relative error on the size of the small flow is larger than 20,000%.

[6]In practice, our filtering resulted in attenuating the estimation accuracy improvements of our techniques.

choice corresponds on average to a five minute period after which a flow or piece of information becomes old, whereas the selected number of segments results in a sliding step on average equal to one minute.

We first fed our dataset to the count-min sketch and to the count-min sketch coupled with either the bit-marking or the sliding-window technique. For each experiment we measured the collision rate over every 1,000 input flows versus the total number of flows entered into the data structures. The goal of this set of experiments was to illustrate that the count-min sketch gradually saturates, whereas our two improvements effectively address this problem. We tried different combinations for the parameters $d = (1, 2, 4, 8, 16)$ and $w = (128K, 64K, 32K, 16K, 8K)$ that yield a total memory consumption of 512 KBytes (using 32-bit counters), and illustrate in Figures 3 and 4 the plots corresponding to the parameter choices that resulted in the worst and best collision rate improvements for our techniques. In Figure 3, observe that for $d = 16$ and $w = 8K$ the count-min sketch saturates quite fast, while the bit-marking and sliding-window techniques exhibit a lower collision rate, which remains relatively stable as the number of flows entered in the data structure increases. The average collision rate of the count-min sketch, of bit-marking, and of the sliding window was 0.934, 0.69 (26.1% reduction), and 0.089 (90.5% reduction), where the reduction in indicated with respect to the count-min sketch. In Figure 4, we show the collision rate for the choice of parameters ($d = 1$ and $w = 128K$), for which our improvements resulted in the highest benefits with respect to the count-min sketch. We see that in this case the bit-marking and sliding-window techniques substantially reduce the collision rate of the count-min sketch. The average collision rate was 0.34 for the count-min sketch, 0.091 (73.2% reduction) for bit-marking, and 0.006 (98.2% reduction) for the sliding window.

The parameter that differentiates the performance in the two settings is the length of the hash arrays w, i.e., the number of columns of the sketch. This is because the parameter d decreases the probability that a *given* key collides in multiple hash tables, but does not affect the overall number of collisions that occur in the hash tables. In the first setting $w = 8K$ is shorter and each hash array saturates faster resulting in more collisions. For this reason the collision rate of the unmodified sketch reaches its maximum almost immediately. On the other hand, for $w = 128K$ the hash array saturates slower and at the end of the experiment the collision rate has not reached the maximum value. This difference results in the higher performance benefits for our two techniques. For $w = 128K$, bit-marking is more effective in removing inactive flows, as there are fewer collisions between active and inactive flows. As a result the average collision rate of bit-marking is 0.091, whereas it is 0.69 for $w = 8K$. In both settings the sliding-window technique achieves the lowest collision rate. This is because the sliding-window keeps a portion of the data stream and thus does not saturate.

In the case of the sliding-window technique, the lack of saturation is expected because at any point in time a sliding window imposes an upper bound on the number of keys present in the data structure. In the case of the bit-marking technique, there are no analytical guarantees that rule out the possibility of

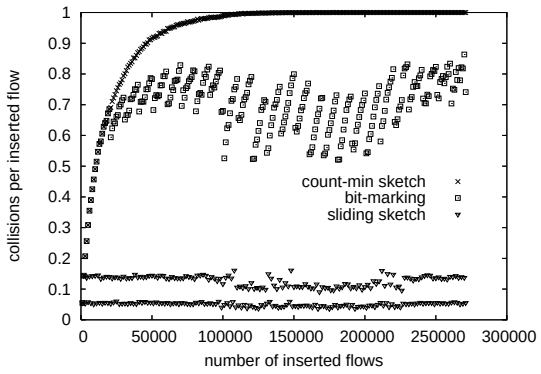

Figure 3: Collision rate of the count-min sketch with and without using the bit-marking or sliding-window techniques. The plot corresponds to the parameter setting $d = 16$ and $w = 8$K, which resulted in the worst collision rate improvement for our techniques.

saturation. However, we observe that in practice the collision rate stays stable over time and does not tend to saturate. This happens because bit-marking effectively removes most inactive flows from the sketch. As we show in the following experiment a large number of flows is removed from the sketch and the collision rate stays low.

Secondly, we analyze the data that are removed and that remain in the sketch with our two techniques. Using bit-marking and the parameter configuration that had the worst performance in the previous experiment, we first find that out of the 120,504 flows in the data stream, at the end of the experiment 114,609 (95.1%) were removed from the sketch, i.e., at least one of their counters was reset, and 5,895 (4.9%) remained in the sketch. In Figure 5 we plot the duration distribution of the removed and remaining flows, where the duration is reported in terms of windows. We see that the two distributions are substantially different. For 70% percent of the flows in the sketch the duration was smaller than 24.4 windows, whereas for the same percentage of removed flows the duration was smaller than 0.02 windows. Clearly, this verifies our expectation that most of the inactive flows removed with bit-marking are transient flows. In addition, the tail of the distribution denotes that among the inactive removed flows exist few flows with long duration, i.e., 0.6% of the removed flows had duration longer than 20 windows.

In contrast to bit-marking, the sliding-window technique removes all information that arrived before the beginning of the window. In other words, it includes information only on the last part of the data stream. At the end of our experiment using the last configuration, we found that: 1) 115,516 (95.9%) flows were removed from the sketch, 2) 2,076 (1.7%) flows were removed from the sketch but re-entered it later, and 3) 2,912 (2.4%) flows only entered the sketch. The total length of the data stream was 27.1 times larger than the win-

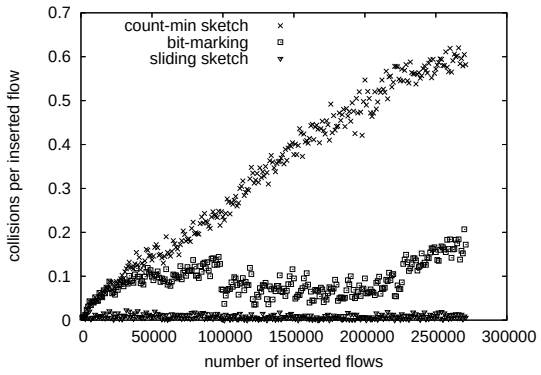

Figure 4: Collision rate of the count-min sketch with and without using the bit-marking or sliding-window techniques. The plot corresponds to the parameter setting $d = 1$ and $w = 128$K, which resulted in the best collision rate improvement for our techniques.

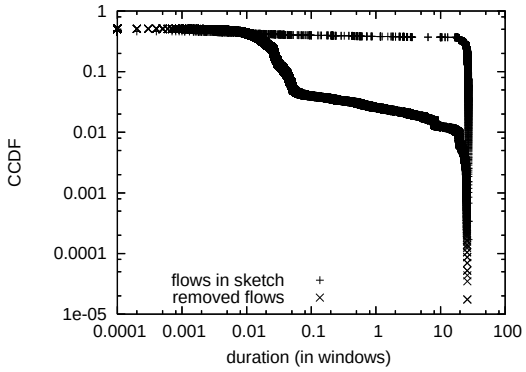

Figure 5: Duration CCDF of removed flows and of flows remain in the sketch using bit-marking. The duration is reported in terms of windows and the sketch had the parameters $d = 16$ and $w = 8$K that exhibited the worst collision rate improvements.

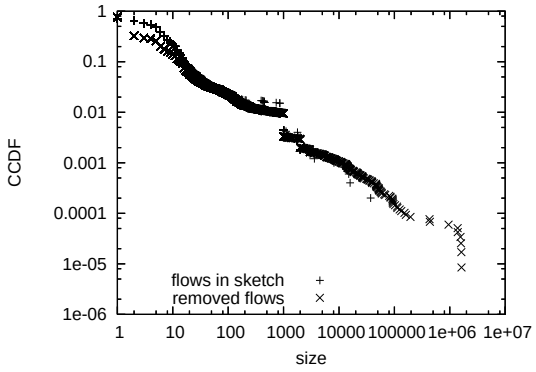

Figure 6: Size CCDF of flows that were in the sketch and of flows that were not in the sketch at the end of the experiment. The size is reported in terms of packets. The sketch configuration was $d = 16$ and $w = 8$K and had the worst improvements in the collision evaluation experiments. The bodies of the two distributions collide, whereas the tail of the distribution for removed flows is longer.

dow length l. The duration of the flows in the sliding window was bounded by the length of the window and on average was equal to 0.52 windows. On the other hand, the duration of the removed flows was bounded by the length of the data stream and on average was equal to 0.78 windows. In Figure 6 we plot the size distribution of flows that were in the sketch and that were removed from sketch at the end of the experiment. We see that the bodies of the two distributions almost collide, which means that the size properties of the two types of flows are not substantially different. This happens because the sliding window technique keeps a portion of the data stream, within which the size properties of the flows do not change significantly. In addition, we observe the the tail of the distribution for the removed flows is significantly longer. This happens because all information that appeared in the first 26 windows was removed, and only information for the last window was kept. Thus, in the first 26 windows we naturally find more large flows than in the last window.

In our third experiment, we measured the mean error over chunks of 1,000 input flows versus the number of flows that enter the data structure. In Figure 7, we plot the mean error of the count-min sketch with and without one of the two ageing techniques. Among the different parameters we explored, the illustrated setting ($d = 1$ and $w = 128$K) resulted in the lowest average mean error, i.e., the average of all the mean error points, for the plain count-min sketch. Observe that the sliding-window and bit-marking techniques significantly improve the estimation accuracy of the count-min sketch. The mean error for the sliding-window technique was always lower than 0.6%, whereas for the bit-marking technique lower than 7.4%. Moreover, note that our techniques exhibit a stable mean error regardless of the number of flows processed. On the other

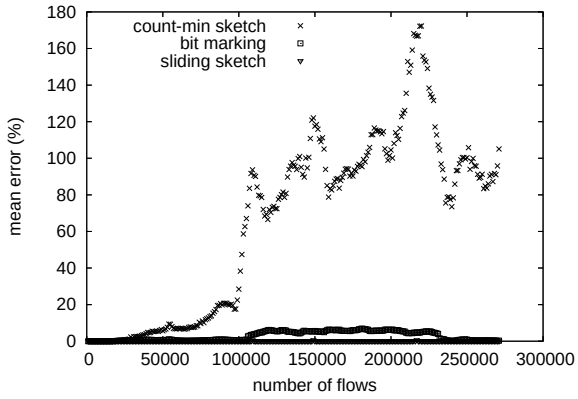

Figure 7: Mean error of the count-min sketch and of the two improvements versus the number of input flow arrivals. In the bottom of the figure, the points that correspond to bit-marking and to the sliding-window technique partially collide.

hand, in the unmodified sketch data structure the mean error was as large as 167.8%. These significant performance improvements demonstrate that keeping the collision rate of the data structure low effectively improves its estimation accuracy.

In Table 1 we summarize a set of aggregate statistics for our experiments. We compute and report the average collision rate and the average mean error over the sets of collision rates and mean error samples, respectively, of an experiment. In addition, we report the memory consumption for the three data structures. We observe that at the cost of a small amount of additional memory, the bit-marking technique provides notable improvements in the average collision rate and the estimation accuracy of the count-min sketch. The sliding-window technique uses more memory resources, but effectively achieves the best performance in terms of the two metrics.

Finally, we ask the question if the same results hold for different datasets. To answer this question, we repeat our experiments using 12 different two-hour traces collected from the same environment. For each dataset we compute the average collision rate and average mean error of the plain count-min sketch and of the count-min sketch coupled with each of our two techniques. In Table 2 we illustrate the five-number summary statistics, i.e., the minimum, maximum, mean, first and third quartile, of the two metrics for the parameter choice $d = 1$ and $w = 128$K. Observe that for a given sketch configuration, the two metrics vary with the input dataset. This is because, different datasets correspond to different diurnal periods and thus contain varying numbers of flow keys. Nevertheless, observe that sliding-window and bit-marking techniques consistently improve the performance of the count-min sketch, regardless of the input dataset.

Table 1: Aggregate statistics on collision rate and estimation accuracy.

		count-min sketch	bit-marking	sliding window
memory usage (KBytes)		512	528	2560
average collision rate	$d = 1$, $w = 128K$	0.350	0.091	0.006
	$d = 16$, $w = 8K$	0.935	0.691	0.089
average mean error	$d = 1$, $w = 128K$	64.941%	2.721%	0.004%
	$d = 16$, $w = 8K$	443.791%	0.0547%	0%

Table 2: Performance variation over 12 different two hour datasets.

		count-min sketch	bit-marking	sliding window
average collision rate	minimum	0.179	0.091	0.006
	1st quartile	0.259	0.117	0.006
	mean	0.380	0.145	0.006
	3rd quartile	0.472	0.182	0.006
	maximum	0.560	0.184	0.007
average mean error	minimum	11.975	0.474	0
	1st quartile	38.846	0.637	0
	mean	81.813	1.841	0.142
	3rd quartile	125.605	2.612	0.055
	maximum	153.297	4.113	1.238

6 Conclusions

In this work we introduced two techniques that remove old information from the sketch data structure in order to reduce the number of hashing collisions and improve its estimation accuracy. The bit-marking technique removes most information for keys that have become inactive and can be used for deriving size estimates on active keys. The sliding-window technique keeps information only for keys that arrived within a recent window and removes all old information. It can be used for estimating characteristics on the last portion of a data stream. Compared to the traditional approach of summarizing a complete data stream in a sketch, our techniques reduce the stored information and the number of collisions, improve the estimation accuracy of the sketch, and find many applications in data streaming problems. In particular, the two techniques can be used by the various applications interested in monitoring and manipulating the keys that are still active or the keys that appeared recently in a data stream. Finally, our two techniques are simple to implement and use a limited amount of additional memory.

References

[1] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, 1999.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, 1970.

[3] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, 2002.

[4] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[5] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, 2002.

[6] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and K.D. Ullman. Computing iceberg queries efficiently. In *Proceedings of the 24th International Conference on Very Large Databases*, 1998.

[7] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of the 27th International Conference on Very Large Databases*, 2001.

[8] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[9] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. QuickSAND: Quick summary and analysis of network data. Technical Report 43, DIMACS, November 2001.

[10] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *IMC '03: Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2003.

[11] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distributions. In *Proceedings of ACM SIGMETRICS*, 2004.

[12] A. Kumar and J. Xu. Sketch guided sampling using on-line estimates of flow size for adaptive data collection. In *Proceedings of IEEE INFOCOM*, 2006.

[13] G. M. Lee, H. Liu, Y. Yoon, and Y. Zhang. Improving sketch reconstruction accuracy using linear least squares method. In *IMC '05: Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2005.

[14] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina. Detection and identification of network anomalies using sketch subspaces. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 147–152, New York, NY, USA, 2006. ACM Press.

[15] S. Muthukrishnan. Datastreams: Algorithms and applications, 2003.

[16] R Schweller, A Gupta, E Parsons, and Y Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2004.

[17] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of USENIX OSDI*, 2004.

[18] S. Venkataraman, D. Song, P. B. Gibbonsy, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *Proceedings of Network and Distributed System Security Symposium NDSS*, 2005.

[19] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications. In *IMC '04: Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference*, pages 101–114, New York, NY, USA, 2004. ACM Press.