The Bloom Paradox: When *not* to Use a Bloom Filter?

Ori Rottenstreich and Isaac Keslassy
Technion
{or@tx,isaac@ee}.technion.ac.il

***Abstract*—In this paper, we uncover the *Bloom paradox*: sometimes, it is better to disregard the Bloom filter results, and in fact not to even query it, thus making the Bloom filter useless.**

We first analyze conditions under which the Bloom paradox occurs, and show that it depends on the *a priori* probability that a given element belongs to the represented set. We also show that the Bloom paradox applies to Counting Bloom Filters (CBFs), and depends on the product of the hashed counters of each element. In addition, both for Bloom filters and CBFs, we suggest improved architectures that deal with the Bloom paradox. We also provide fundamental memory lower bounds required to support element queries with limited false-positive and false-negative rates. Last, using simulations, we verify our theoretical results, and show that our improved schemes can lead to a significant improvement in the performance of Bloom filters and CBFs.

I. Introduction

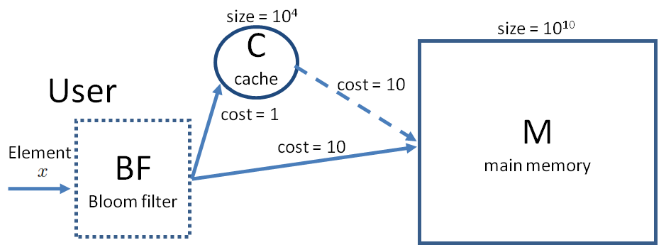
A. The Bloom Paradox

Bloom filters are widely used in many networking device algorithms, in fields as diverse as accounting, monitoring, load-balancing, policy enforcement, routing, filtering, security, and differentiated services [1]–[7]. Bloom filters are probabilistic data structures that can answer set membership queries without false negatives (if they indicate that an element does not belong to the represented set, they are always correct), but also with low-probability false positives (they might sometimes indicate that an arbitrary element is a member of the represented set although it is not). In addition, Bloom filters have many variants. In particular, Counting Bloom Filters (CBFs) add counters to the Bloom filter structure, thus also allowing for deletions within counter limits.

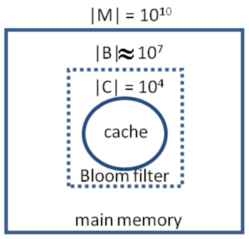
Networking devices typically use Bloom filters as *cache directories*. Bloom filters are particularly popular among designers because a Bloom filter-based cache directory has no false negatives, few false positives, and $O(1)$ update complexity.

In this paper, we show that the traditional approach to this Bloom-based directory forgets to take into account the *a priori set-membership probability* of the elements, i.e. the set-membership probability *without such a directory*. Surprisingly, forgetting this *a priori* probability can actually make the directory more harmful than beneficial.

Figure 1(a) illustrates the intuition behind the importance of the *a priori* set-membership probability. Consider a generic system composed of a user, a main memory containing all the data, and a cache with a subset of the data. When the user needs to read a piece of data, it can simply access the main memory directly, with a cost of 10. Alternatively, it can also access the cache first, with a cost of 1. If the cache owns this piece of data, there is no additional cost. Else, it also needs to access the main memory with an additional cost of 10. This is a generic problem, where the costs may correspond to *dollar* amounts (e.g. for an ISP customer that either accesses a cached Youtube video at the ISP cache, or the more distant Youtube server), to *power* (e.g. in a two-level memory system or a two-level IP forwarding system within a networking device), or to *bandwidth* (e.g. in a data center, with a local cache in the same rack as a server versus a more distant main memory).



(a) Illustration of the importance of the *a priori* set-membership probability. When the user needs element x (illustrated as an arrival of a query for x), there are two options. First, to access the main memory with a fixed cost of 10. Second, to look for it in the cache. With some probability, it is indeed there and the cost is only 1. With the complementary probability, it is not there and the user has to also access the main memory with a total cost of $1+10=11$.



(b) Illustration of the elements in the cache C, those with a positive membership indication in the Bloom Filter B, and those in the memory M. With a false positive probability of 10^{-3}, a positive indication of the Bloom filter is *incorrect* w.p. $\frac{|B\setminus C|}{|B|} \approx \frac{10^7-10^4}{10^7} = 1-10^{-3} \approx 1$.

Fig. 1. Illustration of the Bloom paradox.

Assume that the user holds a Bloom filter to indicate which elements are in the cache, and this Bloom filter has a false positive probability of 10^{-3}. Further assume that this Bloom filter indicates that some arbitrary element x is in the cache. It would seem intuitive to always access the cache in such a case. If the user does access the cache, it would seem that it pays just above 1 on average, since it will most often pay 1 (with probability $1-10^{-3}$), and rarely $1+10=11$ (with probability 10^{-3}). If instead it directly accesses the main memory, it always pays 10.

However, *this approach completely disregards the* a priori *probability*, and it is particularly wrong if the *a priori* probability is too small. For instance, assume that the main memory contains 10^{10} elements, while the cache only contains 10^4 elements. For simplicity, further assume that x is drawn uniformly at random from the memory, i.e., the *a priori* probability that it belongs to the cache is $10^4/10^{10} = 10^{-6}$. This is the probability *before* we query the Bloom filter. Then the probability that x is in the cache *after the Bloom filter says it is in the cache* is only about $\approx 10^{-6}/10^{-3} = 10^{-3}$ (the exact computation is in the paper).

This is the *Bloom paradox*: with high probability $(1-10^{-3})$, x is actually *not* in the cache, even though the Bloom filter indicates that x *is* in the cache. More generally, if the *a priori* probability is low enough before accessing the Bloom filter, *it is better to disregard the Bloom filter results* and always go automatically to the main memory — in fact, it is better to *not even query the Bloom filter*. Taken to the extreme, when the Bloom paradox applies to all elements, it means in fact that *the entire cache is useless*.

Figure 1(b) provides a more formal view to this Bloom paradox. Let B be the set of elements with a positive membership indication from the Bloom filter. Then, $|B| = 10^4 + 10^{-3}\cdot(10^{10}-10^4) \approx 10^7$. While the false positive rate of the Bloom filter is $\frac{|B\setminus C|}{|M\setminus C|} = 10^{-3}$, the probability that a positive indication is *incorrect* is significantly larger and equals $\frac{|B\setminus C|}{|B|} \approx \frac{10^7-10^4}{10^7} = 1-10^{-3}$.

B. Contributions

The main contribution of this paper is pointing out the Bloom paradox, and providing a first analysis of its consequences on Bloom filters and Counting Bloom Filters (CBFs).

First, in Section IV, we provide simple criteria for the existence of a Bloom paradox in Bloom filters. In particular, we develop an upper bound on the *a priori* probability under which the Bloom paradox appears and the Bloom filter answer is irrelevant. Based on this observation, we suggest improvements to the implementation of both the insertion and the query operations in a Bloom filter.

Then, in Section V, we focus on CBFs. We observe that we can calculate a more accurate membership probability based on the exact values of the counters provided in a query, and provide a closed-form solution for this probability. We further show how to use this probability to obtain a decision that optimizes the use of a CBF in a generic system.

Next, in Section VI, we adopt a more fundamental view, beyond the specific example of Bloom-based structures, and provide lower bounds on the memory of a general data structure used to represent a set with limited false positive and false negative rates.

Last, in Section VII, we evaluate our optimization schemes, and show how they can lead to a significant performance improvement. Our evaluations are based on synthetic data as well as on real-life traces.

II. Related Work

In [1], [2], [4], [6], [7], design schemes and applications of Bloom filters and CBFs are presented. In all these works, false negatives are prohibited and only false positives are allowed.

In [8], Donnet el al. presented the Retouched Bloom Filter (RBF), a Bloom Filter extension that reduces its false positive rate at the expense of random false negatives by resetting selected bits. The authors also suggested several heuristics for selectively clearing several bits in order to improve this tradeoff. For instance, choosing the bits to reset such that the number of generated false negatives is minimized, or alternatively, the number of cleared false positives is maximized. They also show that randomly resetting bits yields a lower bound on the performance of their suggested schemes. Unfortunately, calculating the optimal selection of bits can be prohibitive (for instance, it requires going over all the elements in the universe several times), and in practice only approximated schemes are used. For example, the selection of cleared bits is based only on estimated numbers of generated false negatives.

Laufer et al. presented in [9] a similar idea called the Generalized Bloom Filter (GBF) in which in each insertion, several bits are set and other are reset according to two sets of hash functions. To examine the membership of an element a match is required in all corresponding hash locations of both types. False negatives can occur in case of bit overwriting in the insertions of later elements. On the one hand, increasing the number of hash functions reduces the false positive rate since more bits are compared while on the other hand it increases the false negative rate due to higher probability of bit overwriting.

The issue of wrongly considering the *a priori* probabilities is a known problem in diverse fields. For instance, the *Prosecutor's Fallacy* [10] is a known mistake made in law when the prior odds of a defendant to be guilty before an evidence was found are neglected. The same problem is also known as the *False Positive Paradox* in other fields such as computational geology [11], and is also related to *Probabilistic Primality Testing* [12]. Our results might apply to such problems when the costs of false negatives and false positives are taken into account. We leave these to future work.

III. Model and Notations

We consider a Bloom filter (or alternatively a Counting Bloom Filter (CBF)) representing a set S of n elements taken from a universe U of N elements. The Bloom filter uses m bits, and relies on a set of k hash functions $H = \{h_1,\ldots,h_k\}$.

For each element $x \in U$, we denote by $\Pr(x \in S)$ the *a priori* probability that $x \in S$. As explained earlier, this probability function is not based on the Bloom filter itself and is not necessarily uniform over all the elements. We further denote by $\Pr(x \in S | BF = 1)$ the probability that $x \in S$ given that the Bloom filter indicates so, where *BF* is the indicator function of the answer of the Bloom filter to the query of whether x is a member of S.

We assume that the cost function of an answer to a membership query can have four possible values. They are summarized in Table I, which illustrates these costs for a query of an element $x \in U$. If $x \in S$, the cost of a positive (correct) decision is

TABLE I
MEMBERSHIP QUERY DECISION COSTS FOR AN ELEMENT $x \in U$

	Positive Membership Decision	Negative Membership Decision
$x \in S$	$W_P = 0$	$W_{FN} = \alpha \cdot W_{FP}$
$x \notin S$	W_{FP}	$W_N = 0$

W_P while the cost of negative (incorrect) decision is W_{FN}. Similarly, if $x \notin S$, the costs are W_{FP} and W_N for a positive and negative decision, respectively. In the most general case, the costs of the two correct decisions, W_P and W_N might be positive. However, we can simply reduce the problem to the case where $W_P = W_N = 0$ by considering only the marginal additional costs of a negative incorrect decision and a positive incorrect decision $(W_{FN} - W_P)$ and $(W_{FP} - W_N)$. Finally, for $W_{FP} > 0$ let α denote the ratio W_{FN}/W_{FP}. The variable α represents how expensive a false negative error is in comparison with a false positive error.

Finally, for simplicity, we assume that the optimal number k of hash functions is used in the Bloom filter, and that we can model the Bloom filter such that for $x \notin S$, $\Pr(BF = 1) = (1/2)^k = (1/2)^{\ln(2)\cdot(m/n)}$ as often done in the literature. This relies on the simplifying assumptions that the k hashes are distributed uniformly at random over k sub-tables, that half the bits are set, and that the number of hash functions k is equals to $\ln(2) \cdot (m/n)$.

Our goal is to *minimize the expected cost* in each query decision, therefore we return a negative answer iff its expected cost is smaller than the cost of a positive answer.

IV. THE BLOOM PARADOX IN BLOOM FILTERS

In this section we develop conditions for the existence of the Bloom paradox in Bloom filters. We also provide improvements to the implementation of both the insertion and the query operations in a Bloom filter.

A. Conditions for the Bloom Paradox

The next theorem expresses the maximal *a priori* set-membership probability of an element such that the Bloom filter is irrelevant in its queries. This bound depends on the error cost ratio α and on the bits-per-element ratio of the Bloom filter, which impacts its false positive rate.

Intuitively, in cases where the Bloom filter indicates that the element is in the cache, a smaller $\alpha = W_{FN}/W_{FP}$ means that the cost of a false negative is relatively smaller, and therefore we would prefer a negative answer in more cases, i.e., even for elements with a higher *a priori* probability. Therefore, a smaller α allows for the Bloom paradox to occur more often, and in particular also given a higher *a priori* probability.

Theorem 1: The Bloom filter paradox occurs if and only if

$$\Pr(x \in S) < \frac{1}{1 + \alpha \cdot 2^{\ln(2)\cdot(m/n)}} \quad (1)$$

Proof: The Bloom paradox occurs when a negative answer should be returned even though the Bloom filter indicates a membership. In order to choose the right answer, we first calculate the conditioned membership probability when $BF = 1$. First,

$$\Pr(x \in S|BF = 1) = \frac{\Pr(x \in S, BF = 1)}{\Pr(BF = 1)} = \frac{\Pr(x \in S)}{\Pr(BF = 1)},$$

because by definition a Bloom filter always returns 1 for an element in the set S, i.e. $\Pr(BF = 1|x \in S) = 1$. Likewise,

$$\Pr(x \notin S|BF = 1) = 1 - \Pr(x \in S|BF = 1) = \frac{\Pr(BF = 1) - \Pr(x \in S)}{\Pr(BF = 1)}.$$

For $BF = 1$, let $E_1(x)$ denote the expected cost of a positive decision for an element x, and $E_0(x)$ for a negative decision. Then,

$$E_1(x) = \Pr(x \notin S|BF = 1) \cdot W_{FP} = \frac{\Pr(BF = 1) - \Pr(x \in S)}{\Pr(BF = 1)} \cdot W_{FP},$$

and

$$E_0(x) = \Pr(x \in S|BF = 1) \cdot W_{FN} = \frac{\Pr(x \in S)}{\Pr(BF = 1)} \cdot W_{FN}.$$

The Bloom paradox occurs when $E_1(x) > E_0(x)$, i.e

$$\frac{\Pr(BF = 1) - \Pr(x \in S)}{\Pr(BF = 1)} \cdot W_{FP} > \frac{\Pr(x \in S)}{\Pr(BF = 1)} \cdot W_{FN},$$

which can be rewritten as

$$\Pr(BF = 1) > (\alpha + 1) \cdot \Pr(x \in S).$$

We use our model assumption that $\Pr(BF = 1) = (1/2)^{\ln(2)\cdot(m/n)}$ if $x \notin S$. Also, $\Pr(BF = 1) = 1$ if $x \in S$. Then, the left part of the last condition can be rewritten as $\left((1/2)^{\ln(2)\cdot(m/n)} \cdot \Pr(x \notin S) + 1 \cdot \Pr(x \in S)\right)$, and we finally have

$$(1/2)^{\ln(2)\cdot(m/n)} \cdot (1 - \Pr(x \in S)) > \alpha \cdot \Pr(x \in S),$$

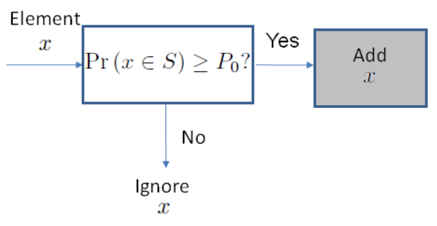
which provides the requested result. ■

B. Analysis of the Bloom Paradox

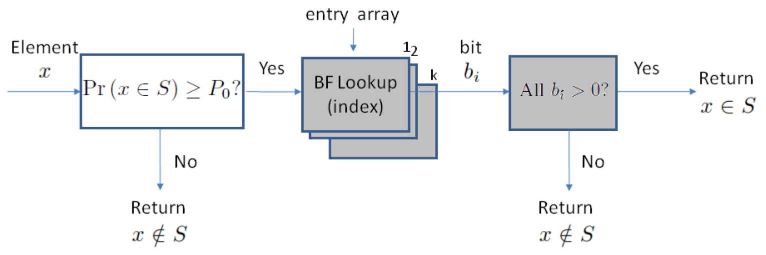
We now provide an illustration of the impact of various parameters on the Bloom paradox.

Figure 3(a) illustrates the probability that a Bloom filter is indeed correct when it indicates that an element x is a member of the set. This probability, $\Pr(x \in S|BF = 1)$, depends on the *a priori* set-membership probability of the element $\Pr(x \in S)$ as well as on the false positive rate of the Bloom filter. For instance, if $\Pr(x \in S) = 10^{-6}$ and the false positive rate is 10^{-3}, the Bloom filter is correct w.p. $\Pr(x \in S|BF = 1) = \frac{\Pr(x\in S)}{\Pr(BF=1)} = \frac{10^{-6}}{10^{-3}\cdot(1-10^{-6})+1\cdot 10^{-6}} \approx 10^{-6}/10^{-3} = 10^{-3}$.

Figure 3(b) plots the boundaries of the Bloom paradox. It presents the minimal bits-per-element ratio m/n needed to avoid the Bloom paradox, as a function of the *a priori* probability, given $\alpha = 0.1, 1, 10, 100$. For instance, if $\alpha = 1$, i.e. the costs of the two possible errors are equal, and the *a priori* probability is $\Pr(x \in S) = 10^{-6}$, at least $m/n = 28.7$ memory bits per element are required to consider the Bloom

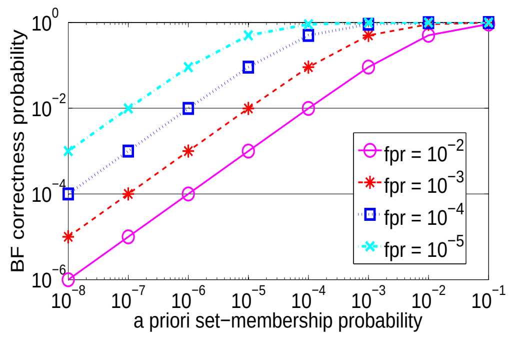


(a) Selective Bloom Filter Insertion. Elements with low *a priori* set-membership probability are not inserted into the Bloom filter.

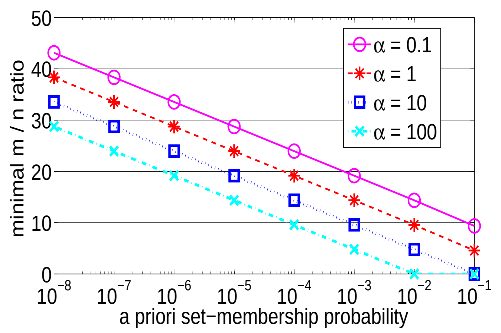


(b) Selective Bloom Filter Query. Elements with low *a priori* set-membership probability are not even queried, as shown in the first rectangle, and a negative answer is always returned for them no matter what the Bloom filter would have actually stated.

Fig. 2. Logical view of the Selective Bloom Filter implementation. Components that also appear in the Bloom filter are presented in gray. (a) shows a first possible improvement during insertion, where elements that satisfy the Bloom paradox are not even inserted in the Bloom filter. (b) displays a second possible improvement during query, where elements that satisfy the Bloom paradox are not even queried.



(a) The Bloom filter correctness probability as a function of the *a priori* set-membership probability.



(b) Boundaries of the Bloom paradox: minimal number of bits-per-element for a Bloom filter to avoid the Bloom paradox, as a function of the *a priori* set-membership probability.

Fig. 3. Analysis of the Bloom paradox. (a) shows that a lower *a priori* probability makes the Bloom filter increasingly irrelevant, because the *a posteriori* membership probability after the Bloom filter is positive is also lower. This favors the Bloom paradox. (b) provides the exact borders of the region in which the Bloom paradox occurs as a function of the *a priori* probability, the Bloom filter load, and the relative weights of false-positive and false-negative errors.

filter and avoid the Bloom paradox. If this ratio is smaller, the Bloom paradox occurs, so we should return a negative answer for all the queries of x, independently of the answer of the Bloom filter.

C. *Bloom Filter Improvements Against the Bloom Paradox*

Based on the observation in Theorem 1, we suggest the two following improvements to Bloom filters, as illustrated in Figure 2:

Selective Bloom Filter Insertion—If the *a priori* probability of an element x satisfies the Bloom paradox, we will not take the answer of the Bloom filter into account after the query. Therefore, *it is better not to even insert it in the Bloom filter*, so as to reduce the load of the Bloom filter. Therefore, the final number n^* of inserted elements may satisfy $n^* < n$.

Selective Bloom Filter Query—If the *a priori* probability of an element x satisfies the Bloom paradox, we do not want to take the answer of the Bloom filter into account, and therefore it is better to not even query it. Formally, if $\Pr(x \in S) < P_0 = (1 + \alpha \cdot 2^{\ln(2) \cdot (m/n^*)})^{-1}$, where n^* is the final number of inserted elements, then a negative answer should be returned for the queries of x, regardless of the Bloom filter.

Each of these two improvements can be implemented independently. Implementing the Selective Bloom Filter Insertion alone yields fewer insertions and therefore a lower Bloom filter load, leading to a lower false positive probability. In turn, implementing the Selective Bloom Filter Query alone makes a regular Bloom filter more efficient by discarding useless query results for elements with low *a priori* probability. Finally, implementing both the Selective Bloom Filter Insertion *and* Query results in the strongest improvement that combines the benefits of both approaches. All these approaches are further compared using simulations in Section VII.

Note that each of the two improvements requires knowing the *a priori* probabilities at different times (either during the insertion or during the query). Also, as expected, this approach may cause false negatives, since this may reduce the overall error cost.

D. *Estimating the* a Priori *Probability*

Access patterns to caches tend to have the *locality of reference* property, i.e. it is more likely that recently-used data will be accessed again in the near future. Therefore, the *a priori* probability distribution might be significantly non-uniform over U.

In such cases, we suggest to estimate the *a priori* probability by *sampling* arbitrary element queries and checking whether they belong to the cache. In practice, for 1% of element queries, we will check whether they belong to the cache, and use an exponentially-weighted moving average to approximate the *a priori* probability.

In addition, there might be several subsets of elements with clearly different *a priori* probabilities. For instance, packets originating from Class-A IP addresses might have distinct *a priori* probabilities from those with classes B and C. Then we will simply model the *a priori* probability as uniform over each class, and sample each class independently.

V. The Bloom Paradox in the Counting Bloom Filter

A. The CBF Based Membership Probability

In this section, we want to show the existence and the consequences of the Bloom paradox in Counting Bloom Filters (CBFs). To do so, we show how we can calculate the membership probability of an element in S based on the exact values of the counters of the CBF. We show again the existence of a *Bloom paradox*: in some cases, a negative answer should be returned even though the CBF indicates that the element is inside that set. Finally, we prove a surprisingly simple result that was interesting to us: to determine whether an element that hashes into k counters falls under the Bloom paradox, we only need to compare *the product of these counters* with a threshold, and do not have to analyze a full combinatorial set of possibilities.

We start by calculating $\Pr(x \in S | CBF(h_j(x)))$, the membership probability of x based on the j-th out of the k counters that the k hash values $h_i(x)$ for $i \in \{1, \ldots, k\}$ point to. We then present the probability based on all of these k counters, denoted by $\Pr(x \in S | CBF)$.

Let X be an indicator variable for the event $x \in S$ such that $X = 1$ iff $x \in S$. For $j \in \{1, \ldots, k\}$, let C_j denote the counter $h_j(x)$ and let c_j denote its value. Clearly, we can see that if $c_j = 0$, then $\Pr(x \in S | CBF(h_j(x))) = \Pr(X = 1 | C_j = c_j) = 0$.

Lemma 1: $\Pr(x \in S | CBF(h_j(x)))$

$$= \frac{m \cdot c_j \cdot \Pr(x \in S)}{m \cdot c_j \cdot \Pr(x \in S) + n \cdot k \cdot (1 - \Pr(x \in S))}.$$

Proof: For $c_j = 0$, $\Pr(x \in S | CBF(h_j(x))) = 0$ as explained earlier. For $c_j > 0$,

$$\Pr(x \in S | CBF(h_j(x))) = \Pr(X = 1 | C_j = c_j) = \frac{\Pr(X = 1, C_j = c_j)}{\Pr(C_j = c_j)} = \frac{\Pr(C_j = c_j | X = 1)\Pr(X = 1)}{\Pr(C_j = c_j | X = 1)\Pr(X = 1) + \Pr(C_j = c_j | X = 0)\Pr(X = 0)}$$

We now look at the expression $\Pr(C_j = c_j | X = 1)$ that appears in the numerator as well as in the denominator. If $X = 1$ then $x \in S$ is one of n elements that have been inserted into the data structure. Thus, $c_j - 1$ is the number of times that this counter was accessed by the other $n - 1$ elements in S. Since in each insertion, a counter is accessed w.p. k/m, we have that

$$\Pr(C_j = c_j | X = 1) = \binom{n-1}{c_j - 1}\left(\frac{k}{m}\right)^{c_j - 1}\left(1 - \frac{k}{m}\right)^{n - c_j}.$$

Likewise, when calculating $\Pr(C_j = c_j | X = 0)$, $x \notin S$ is not one among the n elements of S and then

$$\Pr(C_j = c_j | X = 0) = \binom{n}{c_j}\left(\frac{k}{m}\right)^{c_j}\left(1 - \frac{k}{m}\right)^{n - c_j}.$$

Merging the last equations, we have

$$\Pr(x \in S | CBF(h_j(x))) = \Pr(X = 1 | C_j = c_j) = \frac{\Pr(C_j = c_j | X = 1)\Pr(X = 1)}{\Pr(C_j = c_j | X = 1)\Pr(X = 1) + \Pr(C_j = c_j | X = 0)\Pr(X = 0)}$$
$$= \frac{m \cdot c_j \cdot \Pr(X = 1)}{m \cdot c_j \cdot \Pr(X = 1) + n \cdot k \cdot \Pr(X = 0)}$$
$$= \frac{m \cdot c_j \cdot \Pr(x \in S)}{m \cdot c_j \cdot \Pr(x \in S) + n \cdot k \cdot (1 - \Pr(x \in S))}$$

■

We can see that $\Pr(x \in S | CBF(h_j(x)))$ is a monotonically increasing function of the counter value c_j and that the ratio

$$\frac{\Pr(x \in S | CBF(h_j(x)))}{1 - \Pr(x \in S | CBF(h_j(x)))} = \frac{m \cdot c_j \cdot \Pr(x \in S)}{n \cdot k \cdot (1 - \Pr(x \in S))}$$

grows linearly with c_j.

We would like now to present the membership probability of an element $x \in U$ based on the k counters with indices $h_i(x)$ for $i \in \{1, \ldots, k\}$ pointed by the set of k hash functions. Since in an insertion or a deletion of this element other counters are not updated, their values do not have first order dependency on its membership probability and thus are not considered in our calculations.

Theorem 2: For $j \in \{1, \ldots, k\}$ let C_j denote the counter $h_j(x)$ and let $C = (C_1, \ldots, C_k)$. Let $c = (c_1, \ldots, c_k)$ denote the values of these counters. Then, $\Pr(x \in S | CBF)$

$$= \frac{m^k \cdot (\prod_{j=1}^{k} c_j) \cdot \Pr(x \in S)}{m^k \cdot (\prod_{j=1}^{k} c_j) \cdot \Pr(x \in S) + (n \cdot k)^k \cdot (1 - \Pr(x \in S))}.$$

Proof: Let X be an indicator variable for the event $x \in S$ such that $X = 1$ iff $x \in S$. If $c_j = 0$ for any $j \in \{1, \ldots, k\}$, then $\Pr(x \in S | CBF) = 0$. Otherwise, we use the independency among the different sub-arrays of the CBF. If $X = 1$ then $x \in S$ is one of n inserted elements. Thus, $c_j - 1$ is the number of times that the counter C_j was accessed by the other $n - 1$ elements in S. We now have that

$$\Pr(C = c | X = 1) = \prod_{j=1}^{k} \Pr(C_j = c_j | X = 1)$$
$$= \prod_{j=1}^{k} \binom{n-1}{c_j - 1}\left(\frac{k}{m}\right)^{c_j - 1}\left(1 - \frac{k}{m}\right)^{n - c_j}.$$

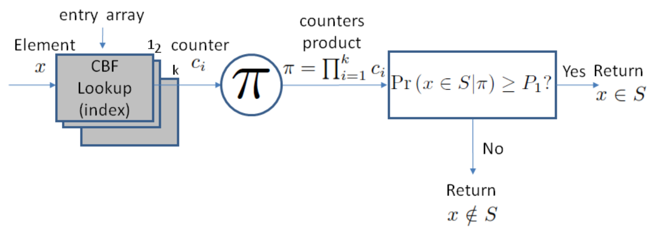


Fig. 4. Selective Counting Bloom Filter Logical View Implementation. Components that also appear in CBF are presented in gray. Membership probability is calculated based on the counters product. Negative answer is returned for elements with low calculated membership probability.

Likewise,

$$\Pr(C = c|X = 0) = \prod_{j=1}^{k} \Pr(C_j = c_j|X = 0)$$

$$= \prod_{j=1}^{k} \binom{n}{c_j} \left(\frac{k}{m}\right)^{c_j} \left(1 - \frac{k}{m}\right)^{n-c_j}$$

and again $\Pr(X = 1) = \Pr(x \in S)$ and $\Pr(X = 0) = (1 - \Pr(x \in S))$. Putting all together, we have

$$\Pr(x \in S|CBF) = \Pr(X = 1|C = c) =$$

$$\frac{\Pr(C = c|X = 1)\Pr(X = 1)}{\Pr(C = c|X = 1)\Pr(X = 1) + \Pr(C = c|X = 0)\Pr(X = 0)}$$

$$= \frac{m^k \cdot (\prod_{j=1}^{k} c_j) \cdot \Pr(x \in S)}{m^k \cdot (\prod_{j=1}^{k} c_j) \cdot \Pr(x \in S) + (n \cdot k)^k \cdot (1 - \Pr(x \in S))}.$$

■

Directly from the last theorem we can deduce the following corollary.

Corollary 3: For an element $x \in U$, the CBF based membership probability $\Pr(x \in S|CBF)$ is an increasing function of the *product* of the k counters pointed by $h_i(x)$ for $i \in \{1, \ldots, k\}$.

B. Optimal Decision Policy For A Minimal Cost

We now suggest an optimal decision policy for a query of an element $x \in U$ with conditioned membership probability, $\Pr(x \in S|CBF)$ and a general set of cost values.

Theorem 4: Let $\Pr(x \in S|CBF)$ be the CBF based membership probability for an element $x \in U$, and let the set of four cost values be as defined in Table I. In order to minimize the expected cost, the decision should be positive if and only if:

$$\Pr(x \in S|CBF) \geq P_1$$

for $P_1 = \frac{W_{FP}}{W_{FN}+W_{FP}} = \frac{1}{\alpha+1}$.

Proof: Let again $E_1(x)$ be the expected cost of a positive decision for an element x and $E_0(x)$ for a negative decision. For a positive decision, the cost is W_P if $x \in S$ (w.p. $\Pr(x \in S|CBF)$) and W_{FP} if $x \notin S$ (w.p $1 - \Pr(x \in S|CBF)$). Thus, we have that

$$E_1(x) = (1 - \Pr(x \in S|CBF)) \cdot W_{FP}.$$

Likewise, we have an expected cost for a negative decision,

$$E_0(x) = \Pr(x \in S|CBF) \cdot W_{FN}.$$

To minimize the expected cost, the scheme decides on a positive answer if $E_1(x) \leq E_0(x)$, i.e when

$$(1 - \Pr(x \in S|CBF)) \cdot W_{FP} \leq \Pr(x \in S|CBF) \cdot W_{FN}$$

With simple algebra, we can see that the last inequality holds if

$$\Pr(x \in S|CBF) \geq P_1.$$

■

Figure 4 illustrates the improved logical process of a query of an element x in the Selective Counting Bloom filter. It is similar to the query process of the Selective Bloom Filter that was presented in Figure 2(b). Here, the product of counters is used to calculate the membership probability.

C. Optimal Number of Hash Functions

Given the parameters m, n of a Bloom filter (and a CBF as well), the number of hash functions $k \approx \ln(2) \cdot (m/n)$ is typically chosen in order to minimize the false positive probability without false negatives. Practically, k must be an integer. In the suggested schemes, the goal is different. Instead of minimizing the false positive probability, we try to minimize the expected cost of a query decision. Thus, the optimal number of hash functions is not necessarily $k \approx \ln(2) \cdot (m/n)$ anymore. According to the decision policy in the previous subsection, we present the expected cost as the function of the number of hash functions k, such that a k that minimizes this cost should be selected.

To do so, we define several new notations. We generalize the function $\Pr(x \in S|CBF)$ to be a function of the number of hash functions k and of the product π of the k counters, while m and n are considered as constants, such that

$$P(x, k_0, \pi) = \Pr\left(x \in S|k = k_0, \left(\prod_{j=1}^{k} c_j\right) = \pi\right)$$

$$= \frac{m^{k_0} \cdot \pi \cdot \Pr(x \in S)}{m^{k_0} \cdot \pi \cdot \Pr(x \in S) + (n \cdot k_0)^{k_0} \cdot (1 - \Pr(x \in S))}.$$

Likewise, we generalize the function of the expected cost for both of the decisions.

$$E_1(x, k_0, \pi) = (1 - P(x, k_0, \pi)) \cdot W_{FP},$$

and

$$E_0(x, k_0, \pi) = P(x, k_0, \pi) \cdot W_{FN}.$$

We then define for $x \in U$ and number of functions k_0, the value $\pi_1(x, k_0)$ as the minimal value of π that yields a probability $P(x, k_0, \pi)$ not smaller than P_1, i.e.

$$\pi_1(x, k_0) = \min\{\pi \in N|P(x, k_0, \pi) \geq P_1\}.$$

Last, we define $P(k_0, \pi)$ as the probability that the product of k_0 counters, one in each sub-array, in a CBF (with parameters

$n, m, k_0)$ is exactly π. It can be calculated as the sum of the probabilities for the vectors of counters with this property.

Then, we have that the expected cost function for an element $x \in U$ is:

$$E(x,k_0) = \sum_{\pi'=0}^{\pi_1(x,k_0)-1} P(k_0,\pi') \cdot E_0(x,k_0,\pi') + \sum_{\pi'=\pi_1(x,k_0)}^{\infty} P(k_0,\pi') \cdot E_1(x,k_0,\pi').$$

Given the probability that a query element is $x \in U$, $P_Q(x)$, the expected cost for k_0 hash functions is

$$\begin{aligned} E(k) &= \sum_{x\in U} P_Q(x) \cdot E(x,k) \\ &= \sum_{x\in U} P_Q(x) \cdot \left(\sum_{\pi'=0}^{\pi_1(x,k_0)-1} P(k_0,\pi') \cdot E_0(x,k_0,\pi') \right. \\ &\qquad \left. + \sum_{\pi'=\pi_1(x,k_0)}^{\infty} P(k_0,\pi') \cdot E_1(x,k_0,\pi') \right). \end{aligned}$$

Thus, the optimal number of hash functions is the value of k that minimizes the last expression.

VI. A Memory Lower Bound on a Data Structure With False Positives and False Negatives

A. Related Work

As we have noticed so far, the suggested schemes may yield false positives as well as false negatives. In order to examine the efficiency of the solution, we would like to present lower bounds on the memory required to represent a set of a given size with limited false positive and false negative rates. Let $S \subseteq U$ be the represented set of n elements from a universe of size $|U| = N$. Likewise, let ϵ, δ denote the upper bounds of the false positive and false negative rates, respectively.

Carter et al. suggested in [13] a lower bound on the number of bits m required to represent such a set S without false negatives. Any string that represents S can accept at most $n+\epsilon(N-n)$ elements. Thus, any string s can represent correctly at most $\binom{n+\epsilon(N-n)}{n}$ sets. In order to represent all the possible sets by strings of length m, we have:

$$2^m \binom{n+\epsilon(N-n)}{n} \geq \binom{N}{n},$$

or alternatively (for $N \gg n$) the entropy lower bound,

$$m \geq \log_2 \left(\frac{\binom{N}{n}}{\binom{n+\epsilon(N-n)}{n}} \right) \approx n \log_2(1/\epsilon).$$

An important property of Bloom filters is that they are dynamic in a sense of keeping their succinct representation, while enabling a sequence of elements to be inserted one at a time. Recently, Lovett and Porat showed in [14] that any dynamic data structures for approximate membership cannot achieve this entropy lower bound.

This entropy lower bound was generalized in [15] for the case that both false positives and false negatives are allowed in the concept of limited-error dictionaries. To do so, they calculate again the number of sets that can be represented correctly by the same string of m bits.

B. Improved Memory Lower Bound

Although the last mentioned bound can also apply to Bloom filters, we would like to present a more accurate analysis for this case without some of their assumptions and approximations.

A string that represents a set of size n with limited error rates of ϵ, δ can accept at most $n+\epsilon(N-n)$ (due to the false positive bound) and at least $n - \delta \cdot n = (1-\delta)n$ (due to the false negative bound) elements.

Let $a \in R$ for $R = [(1-\delta)n, n+\epsilon(N-n)]$ be the number elements that are accepted by a string s. (For simplicity, we assume that the values $n+\epsilon(N-n)$ and $(1-\delta)n$ are both integers. Otherwise, we can instead use the rounded values $\lfloor n+\epsilon(N-n) \rfloor$ and $\lceil (1-\delta)n \rceil$, respectively.) In the following lemma we present the number of sets of size n that can be represented correctly by the same string that accepts exactly a elements.

Lemma 2: The number of sets of size n that can be represented with limited false positive and false negative rates of ϵ and δ, by a string that accepts exactly a elements is

$$X_a = \sum_{i=max(a-\epsilon(N-n),(1-\delta)n)}^{min(n,a)} \binom{a}{i} \binom{N-a}{n-i}.$$

Proof: The n represented elements can be selected among the a accepted elements and the other $(N-a)$ rejected elements according to the following constraints:

- The number of represented elements among the a accepted is at least $a - \epsilon(N-n)$. Otherwise, the number of false positives is greater than $\epsilon(N-n)$.
- The number of elements in the represented set among the a accepted is at least $(1-\delta)n$. Otherwise, the number of false negatives is greater than δn.
- Clearly, the number of represented elements among the a accepted is not greater than a and not greater than the total number of represented elements n.

By combining these constraints, we obtain the presented formula of X_a. ■

Based on the lemma we can deduce the following theorem.

Theorem 5: The number m of bits in a string representing a set S with the requested error rates from above satisfies

$$m \geq \log_2 \left(\binom{N}{n} / \left(\max_{a\in R} X_a \right) \right).$$

Proof: As in the calculation of the previous bound, all the possible $\binom{N}{n}$ sets must be represented correctly by one of the 2^m strings of m bits. Since each string represents at most $\max_{a\in R} X_a$ sets, the result follows. ■

VII. Simulations

A. Bloom Filter Simulations

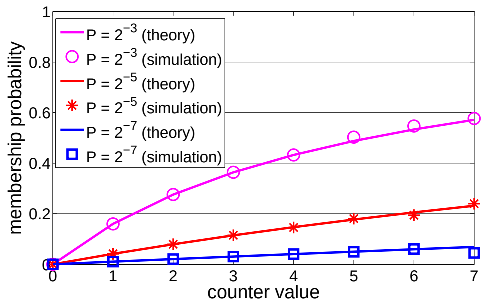
Table II compares the false positive rate (fpr), false negative rate (fnr) and the total cost for the Bloom Filter (BF) [1], Generalized Bloom Filter (GBF) [9], Retouched Bloom Filter (RBF) [8] and the suggested Selective Bloom Filter with its three variants.

We assume a set S composed of 256 elements from each of 13 types of elements, such that $n = |S| = 2^8 \cdot 13 = 256 \cdot 13 = 3328$. Each subset of 256 elements are selected homogenously among sets of sizes $2^{11}, 2^{12}, ..., 2^{23}$. Thus, for $i \in [1, 13]$ an element of the i-th type is member of S with *a priori* set-membership probability of $2^{-(i+2)}$ and $N = |U| = \sum_{i=1}^{13} 2^{i+10} = 16775168$. The numbers of bits per element (bpe) are $4, 6, 8$ and 10 such that $m = n\cdot$ bpe.

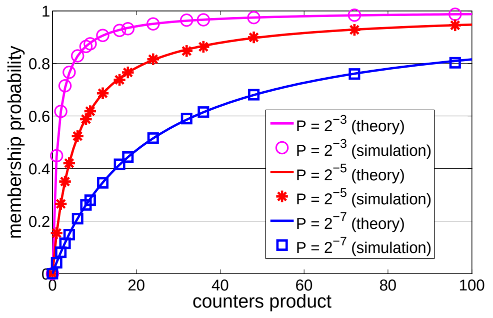
As usual, the false positive rate (fpr) is calculated among the $N - n$ elements of $U \setminus S$ and the false negative rate (fnr) is calculated among the n members of S. In the calculation of the total cost, we assume that $W_{FP} = 1$ and $W_{FN} = \alpha$ such that the cost equals $(N - n)\cdot$ fpr + $n\cdot$ fnr $\cdot\alpha$. The results are presented for the values $\alpha = 100$ and $\alpha = 5$ which illustrates two possible scenarios for the ratio of the two errors costs. In the Bloom Filter and in the three variants of the Selective Bloom Filter we use $k \approx \ln(2) \cdot (m/n)$ hash functions. In the Generalized Bloom Filter we use $k_1 = k$ hash functions to select bits to be set and $k_0 \in [1, k_1 - 1]$, the number of functions to select bits to reset, was chosen such that the total cost is minimized. For the Retouched Bloom Filter we used the *Ratio Selection* as the clearing mechanism. In this heuristic, shown to be the best scheme in [8], the bits to be reset are selected, such that the ratio of the additional false negatives and the cleared false positives is minimized.

We first note that when the *a priori* set-membership probabilities are available in the insertion process as well as in the query process, the Selective Bloom Filter always improves the total cost achieved in Bloom Filter, GBF and RBF even when $\alpha = 100$ and the cost of a false negative is relatively very high. For instance, when they are always available, with memory of 4 bits per element (and $\alpha = 100$) the total cost is 1.78e5 in comparison with 2.46e6, 6.37e5 and 3.32e5 in Bloom filter, GBF and RBF respectively. A relative reduction of 92.76%, 72.04%, 46.46%. If $\alpha = 5$, the cost of a false negative is relatively small, and the achieved false positive rate - false negative rate tradeoff is (1.87e-4, 5.38e-1) instead a tradeoff of (3.08e-3, 3.80e-1) (with larger false positive rate and smaller false negative rate) achieved for $\alpha = 100$. With this smaller value of α, the cost is 1.21e4 instead of 2.46e6 in Bloom filter. This is a relative reduction of 99.51% and a significant improvement in two orders of magnitude.

We can also see that, in this simulation, the contribution of the *a priori* probabilities is more significant in the query process than in the insertion process. For instance, with 4 bits per element and $\alpha = 100$, the cost is 9.49e5 if the probabilities are known only in the insertion, while it is only 1.90e5 when they are used only during the query. It can be explained by the fact that in our experiment, the set $U \setminus S$ is much larger than the set S itself. Thus, the effect of avoiding the false positives of elements with smaller *a priori* set-membership probability during the query is larger than the effect achieved by avoiding the insertion of elements with such probabilities.



(a) Membership probability based on *a single counter value* (out of $k = 6$ counters) in comparison with Lemma 1.



(b) Membership probability based on *k=6 counter values* in comparison with Theorem 2.

Fig. 5. CBF-based membership probability for elements with *a priori* set-membership probability $P = \Pr(x \in S)$.

B. *Counting Bloom Filter Simulations*

In this section we conduct experiments on CBFs. We first examine the CBF based membership probability in comparison with Lemma 1 and Theorem 2. Then, we try to use these probabilities to further reduce the expected cost of a query.

The set S is defined exactly as in the previous simulation. It again includes $n = 13 \cdot 256 = 3328$ elements of 13 types with *a priori* probabilities of $2^{-3}, 2^{-4}, ..., 2^{-15}$. Here, since CBFs with four bits per entry are used, we consider bits per elements values of $16, 24, 32$ and 40.

First, Figure 5(a) presents the simulated membership probability based on one counter in comparison with Lemma 1. The results are presented for counters values of up to 7, since the probabilities for larger values are quite negligible. As expected, the membership probability grows with the counter value. For example, if the counter value is 2, the membership probability are $0.27586, 0.07921, 0.02057$ for the elements with *a priori* set-membership probabilities of $2^{-3}, 2^{-5}, 2^{-7}$, respectively. For each type of elements, larger membership probabilities of $0.57143, 0.23140$ and 0.06846 are achieved for counter value of 7. The simulated results are slightly less close to the theory for larger values of the counters, since, according to their distribution the number of simulated counters with these values is much smaller.

Figure 5(b) displays the membership probability based on the values of the $k = 6$ counters. According to Theorem 2, the probability can be described as a function of the product of these k counters. The figure presents the results for up to a product of 100, since larger products were encountered in the simulations with a negligible probability. The simulated

(a) $\alpha = 100$

		Bloom Filter			Generalized Bloom Filter			Retouched Bloom Filter		
bpe	m	fpr	fnr	cost	fpr	fnr	cost	fpr	fnr	cost
4	13312	1.47e-1	0.00	2.46e6	2.52e-2	6.46e-1	6.37e5	0.00	1.00	3.32e5
6	19968	5.63e-2	0.00	9.44e5	5.06e-3	7.35e-1	3.29e5	0.00	1.00	3.32e5
8	26624	2.17e-2	0.00	3.64e5	3.09e-4	8.65e-1	2.93e5	3e-6	1.00	3.32e5
10	33280	8.24e-3	0.00	1.38e5	1.35e-4	8.44e-1	2.83e5	8.87e-3	0.00	1.49e5

		Selective Bloom Filter (Only Insertion)			Selective Bloom Filter (Only Query)			Selective Bloom Filter (Insertion & Query)		
bpe	m	fpr	fnr	cost	fpr	fnr	cost	fpr	fnr	cost
4	13312	4.94e-2	3.64e-1	9.49e5	2.20e-3	4.62e-1	1.90e5	3.08e-3	3.80e-1	1.78e5
6	19968	2.40e-2	2.24e-1	4.78e5	1.60e-3	3.85e-1	1.55e5	3.00e-3	2.31e-1	1.27e5
8	26624	9.28e-3	1.51e-1	2.06e5	2.29e-3	2.31e-1	1.15e5	2.31e-3	1.54e-1	9.00e4
10	33280	5.39e-3	7.64e-2	1.16e5	1.99e-3	1.54e-1	8.45e4	2.69e-3	7.69e-2	7.08e4

(b) $\alpha = 5$

		Bloom Filter			Generalized Bloom Filter			Retouched Bloom Filter		
bpe	m	fpr	fnr	cost	fpr	fnr	cost	fpr	fnr	cost
4	13312	1.47e-1	0.00	2.46e6	2.53e-2	6.43e-1	4.34e5	0.00	1.00	1.66e4
6	19968	5.66e-2	0.00	9.50e5	5.06e-3	7.34e-1	9.70e4	0.00	1.00	1.66e4
8	26624	2.17e-2	0.00	3.64e5	3.04e-4	8.65e-1	1.95e4	0.00	1.00	1.66e4
10	33280	8.23e-3	0.00	1.38e5	1.33e-4	8.43e-1	1.63e4	0.00	1.00	1.66e4

		Selective Bloom Filter (Only Insertion)			Selective Bloom Filter (Only Query)			Selective Bloom Filter (Insertion & Query)		
bpe	m	fpr	fnr	cost	fpr	fnr	cost	fpr	fnr	cost
4	13312	2.47e-2	5.24e-1	4.23e5	1.16e-4	7.69e-1	1.47e4	1.87e-4	5.38e-1	1.21e4
6	19968	7.90e-3	4.56e-1	1.40e5	9.1e-5	6.92e-1	1.31e4	2.44e-4	4.61e-1	1.18e4
8	26624	2.22e-3	3.83e-1	4.37e4	6.8e-5	6.15e-1	1.14e4	1.39e-4	3.84e-1	8.73e3
10	33280	1.21e-3	3.07e-1	2.53e4	1.22e-4	4.62e-1	9.72e3	1.52e-4	3.08e-1	7.67e3

TABLE II
COMPARISON OF FALSE POSITIVE RATE (FPR), FALSE NEGATIVE RATE (FNR) AND THE TOTAL COST FOR BLOOM FILTER, GENERALIZED BLOOM FILTER, RETOUCHED BLOOM FILTER AND THE SUGGESTED SELECTIVE BLOOM FILTER WITH THREE VARIANTS. IN THE FIRST, THE *a priori* SET-MEMBERSHIP PROBABILITY IS KNOWN ONLY DURING THE INSERTION OF THE ELEMENTS, WHILE IN THE SECOND VARIANT IT IS KNOWN ONLY IN THE QUERY PROCESS AND IN THE THIRD ONE IT IS KNOWN IN BOTH OF THEM. THE TOTAL NUMBER OF INSERTED ELEMENTS IS $n = 256 \cdot 13 = 3328$ WITH *a priori* SET-MEMBERSHIP PROBABILITIES OF $2^{-3}, 2^{-4}, ..., 2^{-15}$ AND $|U| = 16775168$.

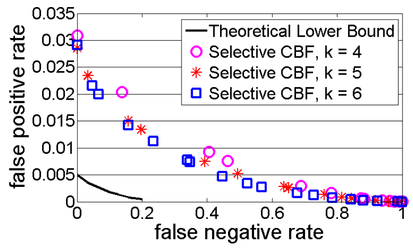
probabilities for the most common values are compared with the theory. The dependency in the *a priori* set-membership probability is demonstrated again. For instance, if the product is 8, the observed probabilities are $0.90594, 0.68504$ and 0.34679 for the *a priori* probabilities $2^{-3}, 2^{-5}, 2^{-7}$. Likewise, to obtain a membership probability of at least 0.8, the minimal required products are $5, 23$ and 91, respectively.

Last, we compared the achieved false positive rate, false negative rate and total cost in a CBF (with the regular query policy) and while using our suggested query policy of the Selective Counting Bloom Filter, as presented in Section V. There is no change in the insertion of elements policy of the CBF, allowing the insertion of all elements, even with low *a priori* probability. The results are presented in Table III. With the increase (in factor of 4) in memory, the performance of the CBF is the same as of the Bloom filter. This is because its regular query policy cannot contribute to reduce the false positive rate.

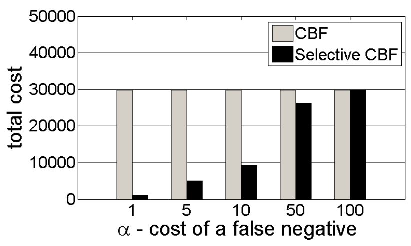
Our suggested policy for the decision helps to reduce the total cost in at most 99.42%. By comparing the query policy of the Selective CBF, to the results of the suggested query policy in the Selective Bloom Filter (presented in Table III), we can see an additional improvement of up to 11.43%. This reduction in the total cost is due to the more accurate calculation of the membership probability based on the information of the exact values of the counters. Such information is not available in the Selective Bloom Filter.

		Counting Bloom Filter			Selective Counting Bloom Filter (Only Query)		
bpe	m	fpr	fnr	cost	fpr	fnr	cost
16	13312	1.47e-1	0.00	2.46e6	8.95e-5	7.65e-1	1.42e4
24	19968	5.61e-2	0.00	9.40e5	9.59e-5	6.57e-1	1.25e4
32	26624	2.16e-2	0.00	3.61e5	9.42e-5	5.35e-1	1.05e4
40	33280	8.20e-3	0.00	1.37e5	1.01e-4	4.16e-1	8.61e3

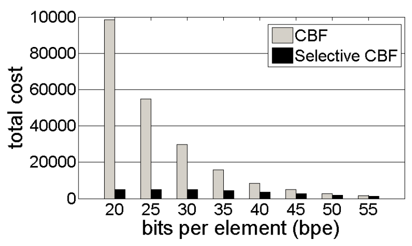
TABLE III
SIMULATION RESULTS FOR COUNTING BLOOM FILTERS WITH $\alpha = 5$. SIMULATION PARAMETERS ARE SAME AS IN TABLE II.



(a) Tradeoff of false positive rate vs. false negative rate for 30 bits per element.



(b) Total cost of N queries in the CBF and the Selective CBF. The cost of a false positive is 1, and the cost of a false negative is α. Since the CBF does not allow any false negatives, its cost is constant and does not depend on α. For smaller values of α, the improvement of the Selective CBF is more significant.



(c) Total cost of N queries in the CBF and the Selective CBF as a function of the number of bits per element (bpe). The cost of a false positive is 1, and the cost of a false negative is $\alpha = 5$. For smaller values of bpe, the false positive rate of the CBF is larger. The probability that its positive indication is correct is lower, and the improvement of the Selective CBF is more significant.

Fig. 6. Trace-Driven Simulations

C. Trace-Driven Simulations

We now want to explore the tradeoff of the false positive rate and the false negative rate in the Selective CBF. To do so, we conduct experiments using real-life traces recorded on a single direction of an OC192 backbone link [16]. We used a 64-bit mix hash function [17] to implement the requested hash functions. The hash functions are calculated of the tuple (Source IP, Destination IP, Source Port, Destination Port, Protocol).

The Selective CBF represents here, using 30 bits per element and 4 bits per counter, a set of $n = 2^{10}$ different tuples that we encounter in a short period of 3614 μs. Our queries are based on $N = 2^{20}$ tuples (that includes the first n) that were encountered later on during a longer time interval. This yields an *a priori* set-membership probability of $n/N = 2^{10}/2^{20} = 2^{-10}$.

Figure 6(a) illustrates this tradeoff. First, the solid line presents the tradeoff obtained by Theorem 5. We remind that this theorem is general and talks about the length in bits of a general representing binary string. However, in order to have a fair comparison with the Selective CBF and to compare apples to apples (i.e. counters to counters), in the calculation of this curve, we assume that each string entry has a width of 4 bits, the typical number of counter size in CBFs. Using binary search, we found for each value of the false negative rate δ, a maximal value of the false positive rate ϵ, that still holds the memory constraint according to the theorem. For instance, if $\delta = 0.001$ (i.e. at most $\lfloor \delta \cdot n \rfloor = 1$ out of n are not accepted by the representing string), we found a maximal false positive rate of $\epsilon = 0.004981$ (and at most $\lfloor \epsilon(N-n) \rfloor = 5217$ elements outside the set are accepted by the string). If $\delta = 0.0316$, then ϵ drops to 0.00360.

The three dashed lines draw the tradeoff achieved using the Selective CBF with the $k = 4, 5$ and 6 hash functions. Three points are located on the y-axis. They present, of course, the typical false positive rate of the CBF where no false negatives are allowed. The rates are $0.03086, 0.02850, 0.02908$, respectively and the minimum is achieved for $k = 5 \approx 30/4 \cdot \log(2)$. Thus, if W_{FN} is large enough and $\alpha \to \infty$, the optimal number of hash functions is $k = 5$.

We earlier showed that the membership probability is an increasing function of the product of the k counters. In each of these three lines, each point illustrates a different threshold of the counters product such that a negative answer is returned only if the product is smaller than the threshold. As explained in Section V, in order to minimize the expected cost, each value of α can be translated to a probability threshold of $\frac{1}{\alpha+1}$ by Theorem 4 and later on also to a product threshold. For instance, for $\alpha = 2.4$, the probability threshold is $\frac{1}{\alpha+1} = \frac{10}{34}$. For $k = 5$, the product threshold is 6 and the obtained false positive and false negative rates are 0.00746 and 0.39258, respectively. However, lower error rates of 0.00739 and 0.34766 can be obtained for $k = 6$ with the product threshold 10. Thus, for such α, for instance, the optimal number of hash functions is not the typical number of hash functions in a CBF with the same parameters.

Figure 6(b) compares the total cost of queries of the CBF and the Selective CBF in this simulation as a function of α. We again assume 30 bits per element and $k = 5$ hash functions. Since the CBF does not allow any false negatives, its total cost is constant and equals the number of obtained false positives $\epsilon(N - n) = 29856$. For small values of α, such that 1 and 5, the total costs of the Selective CBF are only 1024 and 4977, respectively. This is a relative reduction of 96.57% and 83.32%. The improvement might still be not negligible, even for larger values of α. For instance, for $\alpha = 50$ the total cost is reduced by 12.09% to 26247.

Figure 6(c) compares the total cost of queries of the CBF and the Selective CBF in the simulation above, as a function of the number of bits per element (bpe). For each value of bpe, the optimal number of hash functions of the CBF are used (in both schemes) and the results are presented for $\alpha = 5$. If less bits per element are used, the false positive rate of the CBF is larger. The probability that its positive indication is correct is lower, and the improvement of the Selective CBF is more significant. Likewise, the tradeoff in the Selective CBF is improved using more bit per element, and thus also its total cost. In all cases, the Selective CBF achieves lower total cost than the CBF. For instance, if bpe=20, the cost of the CBF is reduced from 98561 by 94.80% to 5122. If 50 bits per element are used, the costs are 2690 and 1876, respectively. In this case, since the false positive rate of the CBF is smaller, the relative improvement drops to 30.26%.

VIII. Conclusion

In this paper, we introduced the *Bloom paradox* and showed that in some cases, it is better to return a negative answer to a query of an element, even if the Bloom filter or the CBF indicate its membership. We developed lower bounds on the *a priori* set-membership probability of an element required for the relevancy of the Bloom filter in its queries. We also showed that the exact values of the CBF counters can be used to calculate the set-membership probability. Last, we showed that our schemes lead to a significant improvement in the query cost.

IX. Acknowledgment

This work was partly supported by the European Research Council Starting Grant no. 210389.

References

[1] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, 1970.
[2] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, 2000.
[3] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "Beyond Bloom filters: from approximate membership checks to approximate state machines," in *SIGCOMM*, 2006.
[4] ——, "An improved construction for counting Bloom filters," in *ESA*, 2006.
[5] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," in *SIGMETRICS*, 2008.
[6] H. Song, F. Hao, M. S. Kodialam, and T. V. Lakshman, "IPv6 lookups using distributed and load balanced Bloom filters for 100gbps core router line cards," in *IEEE Infocom*, 2009.
[7] O. Rottenstreich and I. Keslassy, "The variable-increment Counting Bloom Filter," Comnet, Technion, Israel, Tech. Rep. TR10-05, 2011. [Online]. Available: http://webee.technion.ac.il/~isaac/papers.html
[8] B. Donnet, B. Baynat, and T. Friedman, "Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives," in *CoNEXT*, 2006.
[9] R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte, "A generalized bloom filter to secure distributed network applications," *Computer Networks*, vol. 55, no. 8, 2011.
[10] W. Thompson and E. Shumann, "Interpretation of statistical evidence in criminal trials: The prosecutor's fallacy and the defense attorney's fallacy," *Law and Human Behavior*, vol. 1, no. 3, 1987.
[11] H. L. Vacher, "Quantitative literacy - drug testing, cancer screening, and the identification of igneous rocks," *Journal of Geoscience Education*, 2003.
[12] P. Beauchemin, G. Brassard, C. Crepeau, and C. Goutier, "Two observations on probabilistic primality testing," in *Crypto'*, 1986.
[13] L. Carter, R. W. Floyd, J. Gill, G. Markowsky, and M. N. Wegman, "Exact and approximate membership testers," in *STOC*, 1978.
[14] S. Lovett and E. Porat, "A lower bound for dynamic approximate membership data structures," in *FOCS*, 2010.
[15] R. Pagh and F. F. Rodler, "Lossy dictionaries," in *ESA*, 2001.
[16] C. Shannon, E. Aben, K. claffy, and D. E. Andersen, "CAIDA Anonymized 2008 Internet Trace equinix-chicago 2008-03-19 19:00-20:00 UTC (DITL) (collection)," http://imdc.datcat.org/collection/.
[17] T. Wang, "Integer hash function," http://www.concentric.net/~Ttwang/tech/inthash.htm.