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

Ori Rottenstreich and Isaac Keslassy

***Abstract*—In this paper, we uncover the *Bloom paradox* in Bloom filters: sometimes, the Bloom filter is harmful and should not be queried.**

We first analyze conditions under which the Bloom paradox occurs in a Bloom filter, and demonstrate that it depends on the *a priori* probability that a given element belongs to the represented set. We show that the Bloom paradox also applies to Counting Bloom Filters (CBFs), and depends on the product of the hashed counters of each element. In addition, we further suggest improved architectures that deal with the Bloom paradox in Bloom filters, CBFs and their variants. We further present an application of the presented theory in cache sharing among Web proxies. Last, using simulations, we verify our theoretical results, and show that our improved schemes can lead to a large improvement in the performance of Bloom filters and CBFs.

***Index Terms*—The Bloom Filter Paradox, Bloom Filter, Counting Bloom Filter, *A Priori* Membership Probability.**

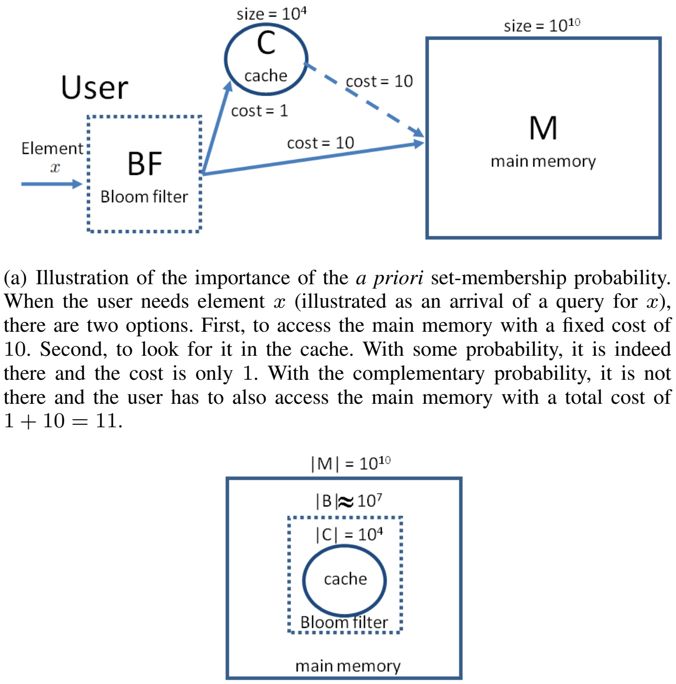
I. Introduction

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]–[6]. 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.



(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.

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).

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

O. Rottenstreich and I. Keslassy are with the Department of Electrical Engineering, Technion, Haifa 32000, Israel (e-mails: or@tx.technion.ac.il, isaac@ee.technion.ac.il).

he pays just above 1 on average, since he will most often pay 1 (with probability $1 - 10^{-3}$), and rarely $1 + 10 = 11$ (with probability 10^{-3}). If instead he directly accesses the main memory, he 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, in this case the Bloom filter is harmful and 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 $\frac{|B \setminus C|}{|B|} \approx \frac{10^7 - 10^4}{10^7} = 1 - 10^{-3}$.

Of course, in the general case, different assumptions may weaken or even cancel the Bloom paradox, especially when caches have significantly non-uniform *a priori* probabilities.

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. We also discuss the effect of the number of hash functions on the CBF performance.

Next, in Section VI, we generalize our analysis to other variants of the Bloom filter and in particular the $B_h - CBF$ scheme [5].

Later, in Section VII, we consider a distributed-cache application with several proxies that use CBFs to represent their cache contents. We suggest an optimal order of the queries that should be sent to the proxies in this network.

Last, in Section VIII, 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

The Bloom filter data structure (and its variants) has a long list of applications in the networking area [7]. This includes, for instance, cache digests [2], packet classification [8], routing [9], deep packet inspection [10], security [11] and state representation [12]. Bloom Filters and CBFs can be found in such well-known products as Facebook's distributed storage system Cassandra [13], Google's web browser Chrome [14] and the network storage system Venti [15].

Different design schemes have been suggested to improve the false positive rate of CBFs with a limited memory size. For instance, the MultiLayer Hashed CBF performs a hierarchical compression [16]. A related approach is presented in [17], [18]. Memory-efficient schemes based on fingerprints instead of on counters were suggested in [3], [12]. The $B_h - CBF$ [5] is a recent efficient CBF variant based on variable increments. In all these variants, false negatives are prohibited and only false positives are allowed.

In [19], Donnet *et 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.

Laufer *et al.* presented in [20] a similar idea called the Generalized Bloom Filter (GBF) in which at each insertion, several bits are set and others 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 during 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. On the other, it increases the false negative rate due to a higher probability of bit overwriting. Data structures with false positives as well as false negatives have also been discussed in [21].

In some Bloom-filter applications, false negatives are not acceptable in any circumstances and avoiding them is mandatory. Such applications can tolerate false positives and not false negatives. In these applications the Bloom paradox does

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}	$W_N = 0$

not exist. Consider for instance, a variant of the Web proxy application from [2] in which Bloom filters are used to summarize the content of each cache. If a large memory with all elements does not exist and each data element can be found in at most one of the caches, false negatives can eliminate the possibility to find a queried element. Another application is in Bloom-filter-based forwarding [22], [23]. Bloom filters are used to encode a delivery tree as a set of forwarding-hop identifiers, e.g. links or pairs of adjacent links in the tree. While false positives can just cause a short increase in the delay, false negatives may prevent the packet from reaching its destination.

The issue of wrongly considering the *a priori* probabilities is a known problem in diverse fields. For instance, the *Prosecutor's Fallacy* [24] 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 [25], and is also related to *Probabilistic Primality Testing* [26]. 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 $x \in S$, i.e. the probability before we query the Bloom filter. We further denote by $\Pr(x \in S | \mathit{BF} = 1)$ 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 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.

In the suggested analysis, we assume for the sake of simplicity uniformly-distributed and independent hash functions. We also assume that the number of hash functions in the Bloom filter and the Counting Bloom filter is the optimal number $k \approx \ln(2) \cdot (m/n)$. Accordingly, the probability of a bit to be set after the insertion of the n elements is 0.5. These assumptions often appear in the literature, e.g. in [3], [27], [28]. Of course, in practice k has to be integer and thus the probability of 0.5 is approximated. Likewise, the requirement for independency between the hash functions can be slightly relaxed while keeping the same asymptotic error [29]. We also assume that each hash function has a range of m/k entries that are disjoint from the entries for the other hash functions. This common implementation is described in [7] and is shown to have the same asymptotic performance.

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 for an element x if and only if its *a priori* membership probability satisfies

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

Proof: We compare the expected cost of positive and negative answers in case the Bloom filter indicates a membership and show that for a low *a priori* membership probability, the cost of a positive answer can be larger. 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,

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

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

$$\begin{aligned}\Pr(x \notin S | \mathit{BF} = 1) &= 1 - \Pr(x \in S | \mathit{BF} = 1) \\ &= \frac{\Pr(\mathit{BF} = 1) - \Pr(x \in S)}{\Pr(\mathit{BF} = 1)}.\end{aligned}$$

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,

$$\begin{aligned} E_1(x) &= \Pr\left(x \notin S | BF = 1\right) \cdot W_{FP} \\ &= \frac{\Pr(BF = 1) - \Pr(x \in S)}{\Pr(BF = 1)} \cdot W_{FP}, \end{aligned}$$

and

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

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 side of the last condition can be rewritten as $\Big((1/2)^{\ln(2) \cdot (m/n)} \cdot \Pr(x \notin S) + 1 \cdot \Pr(x \in S)\Big)$, 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 required result. ■

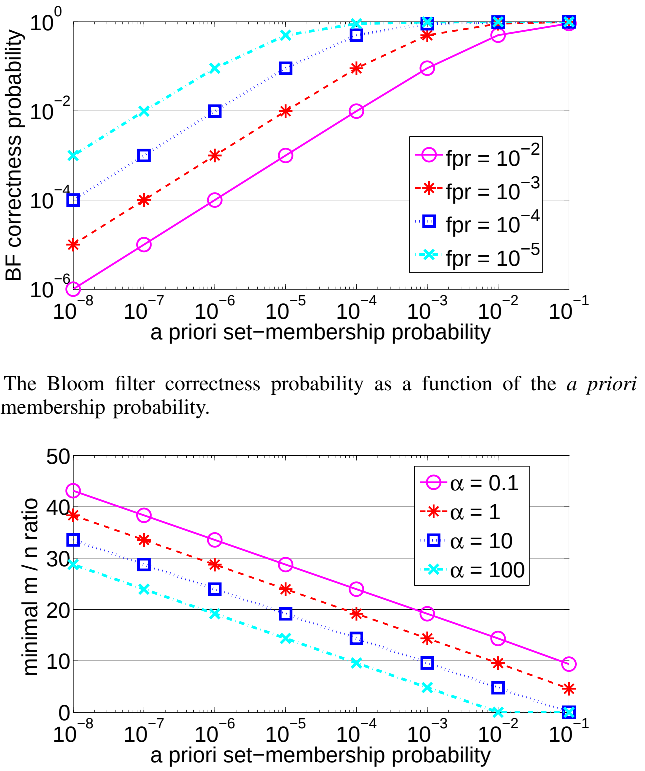
We refer to the inequality from Theorem 1 as the condition for the Bloom paradox.

B. Analysis of the Bloom Paradox

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

Figure 2(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, $\Pr\left(x \in S | BF = 1\right)$, 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. $\Pr\left(x \in S | BF = 1\right) = \frac{\Pr(x \in S)}{\Pr(BF=1)} = \frac{10^{-6}}{10^{-3} \cdot (1 - 10^{-6}) + 1 \cdot 10^{-6}} \approx 10^{-6}/10^{-3} = 10^{-3}$.

Figure 2(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 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.



(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. 2. 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.

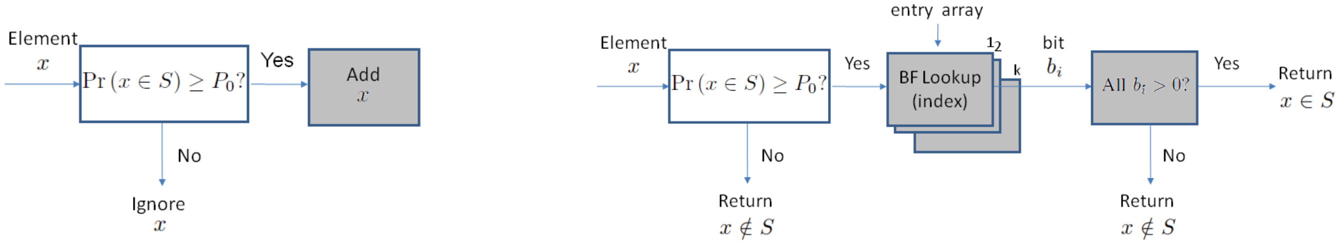
C. Bloom Filter Improvements Against the Bloom Paradox

Based on the observation in Theorem 1, we suggest the two following improvements to Bloom filters, as illustrated in Figure 3:

Selective Bloom Filter Insertion—If the *a priori* probability of an element x satisfies the condition for 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 condition for 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



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

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 harmful 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 VIII.

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. Of course, the accuracy of the estimation depends on the sampling rate. The estimation accuracy as a function of the rate is discussed for instance in [30].

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.

Besides sampling, analytical models have been proposed to predict the locality and the hit rate of a cache while considering its cache size and replacement policy [31]–[33]. For instance, [34] described a model based on the histogram of the time differences between two accesses to the same memory element.

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 simple result that surprised 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.

For an element $x \in U$, we denote by $\Pr(x \in S|CBF)$ its membership probability in S, based on its CBF counter values. That is, on the values of the k counters with indices $h_j(x)$ for $j \in \{1, \ldots, k\}$ pointed by the set of k hash functions $\{h_1, \ldots, h_k\}$. Let $C = (C_1, \ldots, C_k)$ denote the k counter indices of x, i.e. $C_j = h_j(x)$ for $j \in \{1, \ldots, k\}$, and let $c = (c_1, \ldots, c_k)$ denote the values of these counters.

Theorem 2: The CBF-based membership probability is

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

Proof: We again use Bayes' theorem to calculate the conditioned membership probability while now considering the exact values of the k counters. 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

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

Likewise,

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

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

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

$$=\frac{\Pr\left(C=c|X=1\right)\Pr\left(X=1\right)}{\Pr\left(C=c|X=1\right)\Pr\left(X=1\right)+\Pr\left(C=c|X=0\right)\Pr\left(X=0\right)}$$
$$=\frac{\prod_{j=1}^{k}\frac{(n-1)!}{(c_j-1)!(n-c_j)!}\Pr\left(X=1\right)}{\prod_{j=1}^{k}\frac{(n-1)!}{(c_j-1)!(n-c_j)!}\Pr\left(X=1\right)+\prod_{j=1}^{k}\left(\frac{n!}{c_j!(n-c_j)!}\cdot\frac{k}{m}\right)\Pr\left(X=0\right)}$$

$$=\frac{\Pr\left(X=1\right)}{\Pr\left(X=1\right)+\prod_{j=1}^{k}\left(\frac{n}{c_j}\cdot\frac{k}{m}\right)\cdot\Pr\left(X=0\right)}$$
$$=\frac{m^k\cdot(\prod_{j=1}^{k}c_j)\cdot\Pr(x\in S)}{m^k\cdot(\prod_{j=1}^{k}c_j)\cdot\Pr(x\in S)+(n\cdot k)^k\cdot(1-\Pr(x\in S))}.$$

■

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

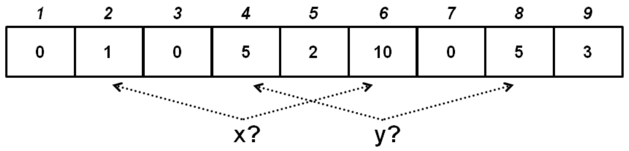
Corollary 3: For an element $x\in U$, the CBF-based membership probability $\Pr\left(x\in S|CBF\right)$ is an increasing function of the *product* of the k counters pointed by $h_i(x)$ for $i\in\{1,\ldots,k\}$.

This result surprised us because of the simple dependency of $\Pr\left(x\in S|CBF\right)$ only in the product of the k counters and not in a more complicated function of them.

Example 1: Figure 4 illustrates queries of two elements x and y in a CBF with parameters of m, n and k. The values of the counters pointed by x are $1,10$. Likewise, they are $5,5$ for y. Then, by Theorem 2 their CBF-based membership probabilities are $\Pr\left(x\in S|CBF\right)=\frac{m^k\cdot(1\cdot10)\cdot\Pr(x\in S)}{m^k\cdot(1\cdot10)\cdot\Pr(x\in S)+(n\cdot k)^k\cdot(1-\Pr(x\in S))}$ and $\Pr\left(y\in S|CBF\right)=\frac{m^k\cdot(5\cdot5)\cdot\Pr(y\in S)}{m^k\cdot(5\cdot5)\cdot\Pr(y\in S)+(n\cdot k)^k\cdot(1-\Pr(y\in S))}$, respectively. In particular, if they have the same *a priori* membership probabilities (i.e. $\Pr(x\in S)=\Pr(y\in S)$) then $\Pr\left(y\in S|CBF\right)>\Pr\left(x\in S|CBF\right)$ since $5\cdot5=25>10=1\cdot10$.

B. Optimal Decision Policy for a Minimal Cost

We now suggest an optimal decision policy for the query of an element $x\in U$ in a CBF. This policy relies on its CBF-based membership probability, which was expressed in Theorem 2 as a function of the product of its counter values.



1	2	3	4	5	6	7	8	9
0	1	0	5	2	10	0	5	3

Fig. 4. Illustration of CBF queries of two elements x and y. If x and y have the same *a priori* membership probabilities then their CBF-based membership probabilities satisfies $\Pr\left(y\in S|CBF\right)>\Pr\left(x\in S|CBF\right)$ since $5\cdot5=25>10=1\cdot10$.

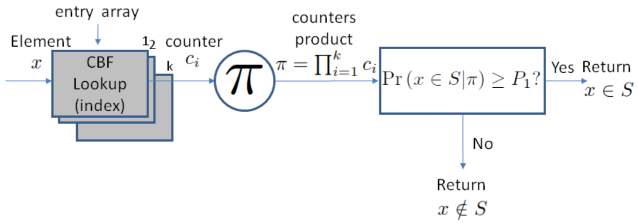


Fig. 5. Logical view of the Selective Counting Bloom Filter implementation. Components that also appear in a regular CBF are presented in gray. Membership probability is calculated based on the counters product. A negative answer is returned for elements with low calculated membership probability.

Theorem 4: An optimal decision policy for the CBF is to be positive on the membership iff

$$\Pr\left(x\in S|CBF\right)\geq\frac{1}{\alpha+1}.$$

Proof: We compare the expected cost of positive and negative decisions given the conditioned CBF-based membership probability, and explain that a positive decision should be made only when the probability is beyond a given threshold. 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. $\Pr\left(x\in S|CBF\right)$) and W_{FP} if $x\notin S$ (w.p $1-\Pr\left(x\in S|CBF\right)$). Thus, we have that

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

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

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

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

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

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

$$\Pr\left(x\in S|CBF\right)\geq\frac{1}{\alpha+1}.$$

■

Figure 5 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 3(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. 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 these counters, while m and n are considered as constants, such that

$$\begin{aligned} P(x,k,\pi) &= \Pr\left(x \in S | k = k, \Big(\prod_{j=1}^{k} c_j\Big) = \pi \right) \\ &= \frac{m^k \cdot \pi \cdot \Pr(x \in S)}{m^k \cdot \pi \cdot \Pr(x \in S) + (n \cdot k)^k \cdot (1 - \Pr(x \in S))}. \end{aligned}$$

Likewise, we generalize the function of the expected cost for both of the decisions. We define $E_0(x,k,\pi)$ and $E_1(x,k,\pi)$ as the expected cost of negative and positive decisions for an element x when the number of hash functions is k and the product of the counters is π. Formally,

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

and

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

We also define for $x \in U$ and k hash functions, the value $\Pi(x,k)$ as the minimal value of π that yields a probability $P(x,k,\pi)$ not smaller than the threshold $1/(\alpha+1)$, i.e.

$$\Pi(x,k) = \min\left\{ \pi \in N | P(x,k,\pi) \geq \frac{1}{\alpha+1} \right\}.$$

Last, we define $P(k,\pi)$ as the probability that the product of k counters, in a CBF is exactly π, where each counter is randomly picked in each sub-array of the CBF. 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 a query of an element $x \in U$ is:

$$\begin{aligned} E(x,k) = &\sum_{\pi=0}^{\Pi(x,k)-1} P(k,\pi) \cdot E_0(x,k,\pi) \\ &+ \sum_{\pi=\Pi(x,k)}^{\infty} P(k,\pi) \cdot E_1(x,k,\pi). \end{aligned}$$

Given the probability that a query is of a specific element $x \in U$ denoted by $P_Q(x)$, the expected cost when k hash functions are used is

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

Thus, the optimal number of hash functions is the value of k that minimizes the last expression. In simulations, we discuss this further.

VI. The Bloom Paradox in Additional Variants of the Bloom Filter

In this section, we examine the existence and the consequences of the Bloom paradox in an additional variant of the Bloom Filter and the Counting Bloom Filter. We consider the $B_h - CBF$ scheme [5], a recently suggested improved Counting Bloom Filter technique based on variable increments. The scheme makes use of B_h *sequences*. Intuitively, a B_h sequence is a set of integers with the property that for any $h' \leq h$, all the sums of h' elements from the set are distinct. Therefore, given a sum of up to h elements, we can calculate, for each element of the B_h *sequence*, its multiplicity in the unique multiset of addends from which the sum is comprised.

The $B_h - CBF$ scheme includes a pair of counters in each hash entry: one with fixed increments, and another one with variable increments that are selected from the B_h sequence. For $j \in [1,k]$, let c_j and a_j be the values of the fixed-increment and variable-increment counters in the entry $h_j(x)$, respectively. Further, we denote the B_h sequence by $D = \{v_1, v_2, ..., v_\ell\}$ such that $|D| = \ell$. The $B_h - CBF$ uses two sets of k hash functions. The first set $H = \{h_1, \ldots, h_k\}$ uses k hash functions with range $\{1, \ldots, m\}$ and points to the set of entries, as in the CBF. The second set $G = \{g_1, \ldots, g_k\}$ uses k functions with range $\{1, \ldots, \ell\}$, i.e. it points to the set D and determines the variable increment of an element in each of its k updated counters.

For an element $x \in U$, we would like to calculate $\Pr(x \in S|B_h - CBF)$, the membership probability of x based on the k corresponding hash entries of the $B_h - CBF$.

Theorem 5: For $j \in \{1, \ldots, k\}$, let y_j denote the multiplicity of $v_{g_j(x)}$ in the addends of the sum a_j. Further, let $I(j)$ be an indicator function representing whether $c_j \leq h$. Then, $\Pr(x \in S|B_h - CBF) =$

$$\frac{m^k \cdot \delta \cdot \Pr(x \in S)}{m^k \cdot \delta \cdot \Pr(x \in S) + (n \cdot k)^k \cdot (1 - \Pr(x \in S))},$$

for

$$\delta = \left(\prod_{j=1, I(j)=0}^{k} c_j \right) \cdot \left(\prod_{j=1, I(j)=1}^{k} \ell \cdot y_j \right).$$

Proof: We again use Bayes' theorem to calculate the conditioned membership probability in $B_h - CBF$ scheme while now considering the k pairs of counters. In each hash entry $h_j(x)$, we consider two possible options according to the value of c_j, the number of elements hashed into this hash entry.

If $c_j \leq h$ (denoted by $I(j) = 1$), based on the properties of the B_h sequence, we can calculate the multiplicity of $v_{g_i(x)}$, the corresponding element of D, in the sum a_j. We denote the random variable representing this value by Y_j.

If $c_j > h$ (and $I(j) = 0$), we cannot necessarily calculate Y_j, and the membership probability is based on the value of the c_j which represents the value of the random variable denoted by C_j.

As in the CBF, also for the $B_h - CBF$ we have that

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

and

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

Likewise, since in the $B_h - CBF$, a hash entry is accessed w.p. k/m and in each entry the variable increment is uniformly selected w.p. $1/\ell$, we also have that

$$\Pr\left(Y_j = y_j | X = 1\right) = \binom{n-1}{y_j - 1}\Big(\frac{k}{m \cdot \ell}\Big)^{y_j - 1}\Big(1 - \frac{k}{m \cdot \ell}\Big)^{n - y_j},$$

and

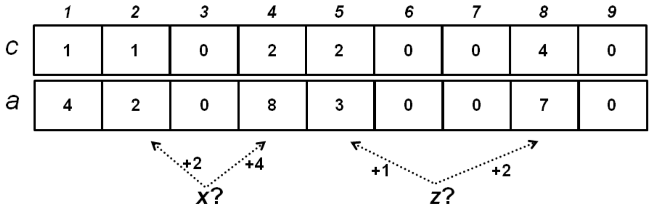
$$\Pr\left(Y_j = y_j | X = 0\right) = \binom{n}{y_j}\Big(\frac{k}{m \cdot \ell}\Big)^{y_j}\Big(1 - \frac{k}{m \cdot \ell}\Big)^{n - y_j}.$$

Let $Q = (Q_1, \ldots, Q_k)$ be an ordered set of n variables such that, for $j \in \{1, \ldots, k\}$, $Q_j = Y_j$ if $I(j) = 1$ and $Q_j = C_j$ if $I(j) = 0$. Let $q = (q_1, \ldots, q_k)$ denote their values. Then,

$$\begin{aligned}
&\Pr\left(x \in S | B_h - CBF\right) = \Pr\left(X = 1 | Q = q\right) \\
&= \frac{\Pr\left(Q = q | X = 1\right)\Pr\left(X = 1\right)}{\Pr\left(Q = q | X = 1\right)\Pr\left(X = 1\right) + \Pr\left(Q = q | X = 0\right)\Pr\left(X = 0\right)} \\
&= \frac{\beta \cdot \Pr\left(X = 1\right)}{\beta \cdot \Pr\left(X = 1\right) + \gamma \cdot \Pr\left(X = 0\right)},
\end{aligned}$$

for

$$\begin{aligned}
\beta = \Pr\left(Q = q | X = 1\right) &= \Big(\prod_{j=1, I(j)=0}^{k} \Pr\left(C_j = c_j | X = 1\right)\Big) \cdot \\
&\Big(\prod_{j=1, I(j)=1}^{k} \Pr\left(Y_j = y_j | X = 1\right)\Big) \\
= \Big(\prod_{j=1, I(j)=0}^{k} &\binom{n-1}{c_j - 1}\Big(\frac{k}{m}\Big)^{c_j - 1}\Big(1 - \frac{k}{m}\Big)^{n - c_j}\Big) \cdot \\
\Big(\prod_{j=1, I(j)=1}^{k} &\binom{n-1}{y_j - 1}\Big(\frac{k}{m \cdot \ell}\Big)^{y_j - 1}\Big(1 - \frac{k}{m \cdot \ell}\Big)^{n - y_j}\Big),
\end{aligned}$$



	1	2	3	4	5	6	7	8	9
c	1	1	0	2	2	0	0	4	0
a	4	2	0	8	3	0	0	7	0

Fig. 6. Illustration of two queries in the $B_h - CBF$ with the $B_{h=2}$ sequence $D = \{1, 2, 4\}$. The j^{th} selected hash entry includes a pair of counters: c_j with fixed increments, and a_j with variable increments. The k variable increments of each element are presented. For each element, we consider the exact values of both counters in $k = 2$ hash entries to calculate the membership probability based on the $B_h - CBF$.

and

$$\begin{aligned}
\gamma = \Pr\left(Q = q | X = 0\right) &= \Big(\prod_{j=1, I(j)=0}^{k} \Pr\left(C_j = c_j | X = 0\right)\Big) \cdot \\
&\Big(\prod_{j=1, I(j)=1}^{k} \Pr\left(Y_j = y_j | X = 0\right)\Big) \\
&= \Big(\prod_{j=1, I(j)=0}^{k} \binom{n}{c_j}\Big(\frac{k}{m}\Big)^{c_j}\Big(1 - \frac{k}{m}\Big)^{n - c_j}\Big) \cdot \\
&\Big(\prod_{j=1, I(j)=1}^{k} \binom{n}{y_j}\Big(\frac{k}{m \cdot \ell}\Big)^{y_j}\Big(1 - \frac{k}{m \cdot \ell}\Big)^{n - y_j}\Big).
\end{aligned}$$

Finally, we have that

$$\begin{aligned}
&\Pr\left(x \in S | B_h - CBF\right) = \frac{\beta \cdot \Pr\left(X = 1\right)}{\beta \cdot \Pr\left(X = 1\right) + \gamma \cdot \Pr\left(X = 0\right)} \\
&= \frac{\Pr\left(X = 1\right)}{\Pr\left(X = 1\right) + \prod_{j=1, I(j)=0}^{k} \frac{n \cdot k}{m \cdot c_j} \cdot \prod_{j=1, I(j)=1}^{k} \frac{n \cdot k}{m \cdot \ell \cdot y_j} \cdot \Pr\left(X = 0\right)} \\
&= \frac{m^k \cdot \delta \cdot \Pr\left(X = 1\right)}{m^k \cdot \delta \cdot \Pr\left(X = 1\right) + (n \cdot k)^k \cdot \Pr\left(X = 0\right)}
\end{aligned}$$

for δ as defined above. ■

We would like now to demonstrate the similarity of the formulas of $\Pr\left(x \in S | CBF\right)$ and $\Pr\left(x \in S | B_h - CBF\right)$ as presented in Theorem 2 and Theorem 5, respectively. Simply, $\Pr\left(x \in S | CBF\right)$ can be obtained from $\Pr\left(x \in S | B_h - CBF\right)$ by setting $I(j) = 0$ for $j \in \{1, \ldots, k\}$, which yields that $\delta = \prod_{j=1}^{k} c_j$. The intuition for that is as follows. In the CBF, the value of δ in the formula of $\Pr\left(x \in S | CBF\right)$ equals the product of the number of inserted elements to each of the k hash entries used by the element x. In the $B_h - CBF$ we can obtain, for some of the hash entries used by x, not only the value of c_j which represents the number of inserted elements into the j^{th} hash entry, but also y_j, the number of inserted elements with the same variable increment as the element x. For such counters, the value of the counter c_j is replaced in the formula of δ by a more informative value which is $\ell \cdot y_j$, where ℓ is the number of possible variable increments. Of course, we should try to avoid the influence of elements inserted into this hash entry with other variable increments that are guaranteed not to be the element x itself. Unfortunately, if $c_j > h$, we cannot do that and δ depends on the value of c_j itself.

As in the CBF, for each value of α we can consider the membership probability $\Pr(x \in S|B_h - CBF)$ based on the $B_h - CBF$, in order to decide on the optimal answer to a membership query. Again, in some cases it might be better to return a negative answer even if $\Pr(x \in S|B_h - CBF) > 0$ and the indication of the $B_h - CBF$ is positive.

Example 2: Figure 6 illustrates queries of two elements x and z in a $B_h - CBF$ with the set of variable increments $D = \{1, 2, 4\}$. We can see all the 6 sums 2 elements of D are distinct: $1 + 1 = 2, 1 + 2 = 3, 1 + 4 = 5, 2 + 2 = 4, 2 + 4 = 6, 4 + 4 = 8$. Therefore, D is a $B_{h=2}$ sequence. For the element x, since $c_1 = 1, c_2 = 2 \leq h$ we can determine the multiplicities $y_1 = 1, y_2 = 2$ of the variable increments $v_{g_1(x)} = 2, v_{g_2(x)} = 4$ in the sums $a_1 = 2, a_2 = 8$, respectively. By Theorem 5, we have that $\Pr(x \in S|B_h - CBF) = \frac{m^k \cdot \delta \cdot \Pr(x \in S)}{m^k \cdot \delta \cdot \Pr(x \in S) + (n \cdot k)^k \cdot (1 - \Pr(x \in S))}$ for $\delta = (\ell \cdot y_1) \cdot (\ell \cdot y_2) = (3 \cdot 1) \cdot (3 \cdot 2) = 18$. Likewise, for the element z, $c_1 = 2, a_1 = 3, v_{g_1(z)} = 1$ and necessarily $y_1 = 1$. However, since $c_2 = 4 > h$, we cannot determine the multiplicity of $v_{g_2(z)} = 2$ in the sum $a_2 = 7 = 1+1+1+4 = 1+2+2+2$. Thus, $I(2) = 0$ and the $B_h - CBF$-based membership probability takes into account c_2 and not y_2. Therefore, $\Pr(z \in S|B_h - CBF) = \frac{m^k \cdot \delta \cdot \Pr(z \in S)}{m^k \cdot \delta \cdot \Pr(z \in S) + (n \cdot k)^k \cdot (1 - \Pr(z \in S))}$ for $\delta = (\ell \cdot y_1) \cdot (c_2) = (3 \cdot 1) \cdot 4 = 12$.

VII. Distributed-Cache Application

Eliminating the Bloom paradox and the ability to calculate the membership probability based on the exact values of the CBF counters can improve the performance of many applications. In this section, we suggest an example for such an application.

A. Model

Our model is based on the model of cache sharing among Web proxies as described in the seminal paper presenting the CBFs [2]. In the *Summary Cache* sharing protocol suggested in this paper, each proxy keeps an array of CBFs that summarize the cache content of each of the other proxies. If a proxy encounters a local cache miss, it examines the summaries of the contents of the other caches. It then sends a query message only to the proxies with a positive membership indication for the existence of the required data element in their caches, based on the corresponding CBFs. Since CBFs have no false negatives, only this subset of the proxies might contain this specific data element. Since the content of each of the caches dynamically changes and deletions have to be supported, we cannot simply use Bloom filters in this case, and need to keep counters. In their model, the performance is measured by the total amount of network traffic as a function of the memory size dedicated for the summaries.

Based on the theory we presented, it is possible to consider the exact values of the CBF counters in order to calculate the membership probability in each of the possible proxies. Thus, we can further distinguish between the proxies with a positive indication. For instance, we can prefer to probe first the proxies with higher chances to have the element in their caches.

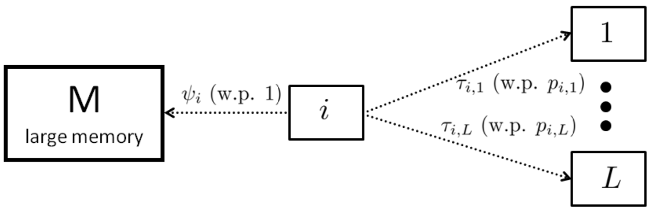


Fig. 7. Illustration of the cache sharing model. A query of an element x can be sent from proxy i to $L - 1$ other proxies. A query sent to proxy $j \in (\{1, \ldots, L\} \setminus \{i\})$ has a traffic cost of $\tau_{i,j}$ and is successful w.p. $p_{i,j} = \Pr(x \in S_j|CBF_{i,j})$. Alternatively, the element x can be found with certainty in a single larger memory with a relatively expensive traffic cost ψ_i.

As illustrated in Figure 7, we generalize the model as follows. Let L be the number of proxies and let S_i represent the content of cache i for $i \in \{1, \ldots, L\}$. The set of all possible data elements is denoted by U. We assume that different proxies may have different traffic costs for a query they send to each of the other proxies. Let $\tau_{i,j} > 0$ denote the traffic cost of a query sent from proxy i to proxy j for $i, j \in \{1, \ldots, L\}$.

We also assume that the *a priori* probability $\Pr(x \in S_i)$ for the membership of each data element $x \in U$ in each of the caches is available. The probabilities may be uniform or non-uniform over U and may vary in each of the caches. For the required data element $x \in U$, we denote by $p_{i,j}$ the membership probability based on the CBF held by proxy i for summarizing the cache content of proxy j. These probabilities are assumed independent. By denoting this CBF by $CBF_{i,j}$, we have $p_{i,j} = \Pr(x \in S_j|CBF_{i,j})$.

Last, we assume that all of the elements can be found in a special proxy (denoted by the index $L + 1$) with a large memory containing all the elements in U. A query sent to this special proxy from each other proxy i has a relatively expensive traffic cost ψ_i. We also refer to the other L proxies as regular proxies.

B. Optimal Query Policy

Given a request for a data element in one of the proxies, we would like to determine the order of the queries it should send in order to minimize the total expected cost of queries until the element is found. Of course, we can clearly see, as suggested also in [2], that since all the queries costs are positive there is no reason to send a query to a proxy when the CBF indicates that it does not contain the required element. In addition, we observe that in this model, any two queries should not be sent in parallel. By sending these queries one after another, the expected cost is reduced since if the first is successful, then the second query can be avoided.

When proxy i encounters a local cache miss for the query of an element $x \in U$, it should consider the success probability of probing a proxy j that satisfies $p_{i,j} = \Pr(x \in S_j|CBF_{i,j}) > 0$. It should also take into account the cost of a single query to each of these proxies $\tau_{i,j}$ and the traffic cost to the special proxy with all the data elements ψ_i.

We consider two possible scenarios and present the optimal order of queries that should be sent in each. These two scenarios can be described as a special case of a very general problem described in [35].

General case: In the first scenario, we deal with general traffic costs and just assume that each of the costs $\tau_{i,j}$ are positive without any additional constraints.

The next theorem presents the optimal order of queries for minimizing the expected cost. Intuitively, we want to access first proxies with high success probability. We also prefer proxies with low query cost. The theorem shows that we should compare the ratio of these two parameters for each of the proxies and query them in a non-increasing order of the ratios. After encountering some failures while accessing the proxies with high ratio, we are left only with proxies that have a relatively low ratio, i.e. proxies with low success probability or high traffic cost. At some point, we prefer to access the special proxy despite its high cost, and obtain the data element with certainty rather than probing the proxies left.

As mentioned earlier, the theorem follows from a more general claim presented in [35] in a different context. We present the shorter and easier proof of this special case.

Theorem 6: An optimal sequence of queries performed by proxy i that minimizes the expected total cost is given by accessing the proxies (the regular proxies or the single special proxy) in a non-increasing order of the ratio of their success probability and the query cost, i.e. the ratio $\frac{p_{i,j}}{\tau_{i,j}}$ for a proxy $j \in (\{1,\ldots,L\} \setminus \{i\})$ or $\frac{1}{\psi_i}$ for the special proxy.
Indeed, after accessing a proxy with a success probability of 1 (and in particular the special proxy), the order of the next queries can be arbitrary.

Proof: The proof is by contradiction. We show that for any sequence with different order, we can switch the order of two queries and reduce the expected cost. For the sake of simplicity, we also use the notations $p_{i,L+1}$ and $\tau_{i,L+1}$ for the success probability (which equals 1) and the query traffic cost (denoted earlier also by ψ_i) of the special proxy, respectively.

Clearly, after accessing a proxy with a success probability of 1, the data element is available, any additional queries are always useless. Let $\sigma = (\sigma_1, \ldots, \sigma_L)$ be an optimal order such that $\sigma_j \in (\{1,\ldots,(L+1)\}\setminus\{i\})$ and for each $1 \leq j < t \leq L$, we have $\sigma_j \neq \sigma_t$. We denote by σ_ℓ the first proxy in this order that has a success probability of 1. Then, $p_{i,\sigma_\ell} = 1$ and $p_{i,\sigma_j} < 1$ for $j \in \{1,\ldots,(\ell-1)\}$. The special proxy is an example for such a proxy and thus there is always at least one proxy with this property. Let $\mathbb{E}(\sigma)$ be the expected traffic cost when the order σ is used.

If σ does not satisfy the presented condition, there are two adjacent indices $1 \leq k, k+1 \leq \ell$ such that $\frac{p_{i,\sigma_k}}{\tau_{i,\sigma_k}} < \frac{p_{i,\sigma_{k+1}}}{\tau_{i,\sigma_{k+1}}}$. We then consider a second order $\sigma' = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_{k+1}, \sigma_k, \sigma_{k+2}, \ldots, \sigma_L)$ obtained by flipping the order of the k^{th} and $(k+1)^{\text{th}}$ queries in σ. We show that $\mathbb{E}(\sigma') < \mathbb{E}(\sigma)$, a contradiction to the optimality of σ.

Clearly, since σ and σ' differ only in the two indices k and $k+1$, such a change can influence the total traffic cost only when the first $k-1$ queries fail and at least one of these two queries succeeds. Assuming that the first $k-1$ queries fail (it happens w.p. $p_{fail} = \left(\prod_{j=1}^{k-1}(1-p_{i,\sigma_j})\right)$), and both of these two considered queries succeed (w.p. $p_{i,\sigma_k} \cdot p_{i,\sigma_{k+1}}$), the change in the traffic cost is $\tau_{i,\sigma_{k+1}} - \tau_{i,\sigma_k}$ with the switching from σ to σ'. This is because the k^{th} and last query is now sent to proxy σ_{k+1} instead of to σ_k. Likewise, when only the query sent to σ_k succeeds, the cost is increased by $\tau_{i,\sigma_{k+1}}$ with the order change. Last, if only σ_{k+1} has the data element, the cost is decreased by τ_{i,σ_k}. To summarize, we have that $\mathbb{E}(\sigma') - \mathbb{E}(\sigma) = p_{fail} \cdot \Big(p_{i,\sigma_k} \cdot p_{i,\sigma_{k+1}} \cdot (\tau_{i,\sigma_{k+1}} - \tau_{i,\sigma_k}) + p_{i,\sigma_k} \cdot (1-p_{i,\sigma_{k+1}}) \cdot \tau_{i,\sigma_{k+1}} - (1-p_{i,\sigma_k}) \cdot p_{i,\sigma_{k+1}} \cdot \tau_{i,\sigma_k}\Big) = p_{fail} \cdot \Big(p_{i,\sigma_k} \cdot \tau_{i,\sigma_{k+1}} - p_{i,\sigma_{k+1}} \cdot \tau_{i,\sigma_k}\Big) < 0$. A contradiction to the optimality of $\mathbb{E}(\sigma)$. ■

Simple case: We now consider the simple case in which each proxy i has the same traffic cost τ_i to each of the regular proxies such that $\tau_i = \tau_{i,j}$ for $j \in (\{1,\ldots,L\} \setminus \{i\})$.

The next theorem presents the optimal order of queries for a minimal expected cost. Now, since the query costs are fixed, we access the proxies in a decreasing order of their success probability and again prefer to access the special proxy after some failures rather than examining the other left proxies with low success probability.

Theorem 7: An optimal sequence of queries performed by proxy i that minimizes the expected total cost is given by accessing the proxies in a non-increasing order of their success probability. When the left proxies have a success probability smaller than $\frac{\tau_i}{\psi_i}$, the next and last query should be sent to the special proxy.

Proof: The proof directly follows from Theorem 6 based on the following observations. Clearly, if the query cost to each of the other proxies is fixed then a non-increasing order of the success probability is also a non-increasing order of the ratio of the success probability and the query cost. Likewise, if $(\forall j \in (\{1,\ldots,L\} \setminus \{i\}))$, $\tau_i = \tau_{i,j}$, then $p_{i,j} < \frac{\tau_i}{\psi_i}$ exactly when $\frac{p_{i,j}}{\tau_{i,j}} < \frac{1}{\psi_i}$ and the next query should be sent to the special proxy. ■

It is interesting to see that the last result can be useful even in some cases where the values of the *a priori* membership probabilities are not available. With the assumption of fixed query costs, if we further assume that the *a priori* membership probability of an element is identical among the different proxies, we can prefer one regular proxy on another. Based on Theorem 2, the order can be determined based only on the values of the product of the k counters in the corresponding CBFs, without knowing the exact values of the success probabilities in each proxy.

VIII. 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) [20], Retouched Bloom Filter (RBF) [19] 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 is selected homogeneously among sets of sizes $2^{11}, 2^{12}, \ldots, 2^{23}$. 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 illustrate two possible scenarios for the ratio of the two error 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. Likewise, the number of functions used to select bits to reset, $k_0 \in [1, k_1 - 1]$, 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 [19], 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 BF, GBF and RBF, even when $\alpha = 100$ and accordingly the cost of a false negative is very high. For instance, when the probabilities are used in the insertion as well as in the query, with a 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 BF, GBF and RBF respectively, i.e. a relative reduction of 92.76%, 72.04%, 46.46%.

If $\alpha = 5$, the cost of a false negative is relatively small. As a result, optimizing the tradeoff of fpr vs. fnr results in an (fpr,fnr) pair of (1.87e-4, 5.38e-1) instead of (3.08e-3, 3.80e-1) for $\alpha = 100$. That is, as expected, the fpr is smaller and the fnr is larger when the relative cost of fnr is smaller. If $\alpha = 5$, the cost is 1.21e4 instead of 2.46e6 in BF. This is a significant improvement by more than 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 used 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.

B. Counting Bloom Filter Simulations

In this section we conduct experiments on CBFs. We first examine the CBF-based membership probability in comparison with 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 bpe values of $16, 24, 32$ and 40.

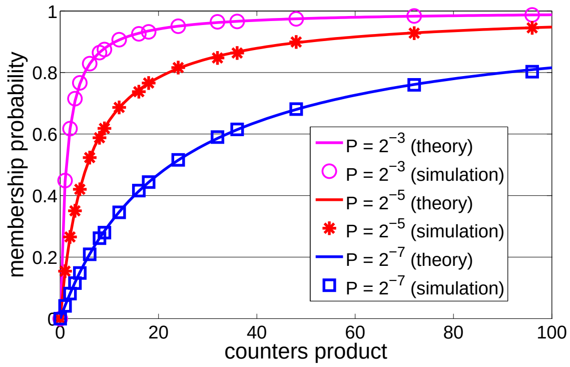


Fig. 8. CBF-based membership probability for elements with *a priori* set-membership probability $P = \Pr(x \in S)$. The probability is based on *k=6 counter values* and compared with Theorem 2.

		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 THE SAME AS IN TABLE II.

Figure 8 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 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 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 (by a factor of 4) in memory, the performance of the CBF is the same as that 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 by 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 II), we can

(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 used only during the insertion of the elements, while in the second variant it is used only in the query process and in the third one it is used 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 $|U| = 16775168$.

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 on the exact values of the counters. Such information is not available in the Selective Bloom Filter.

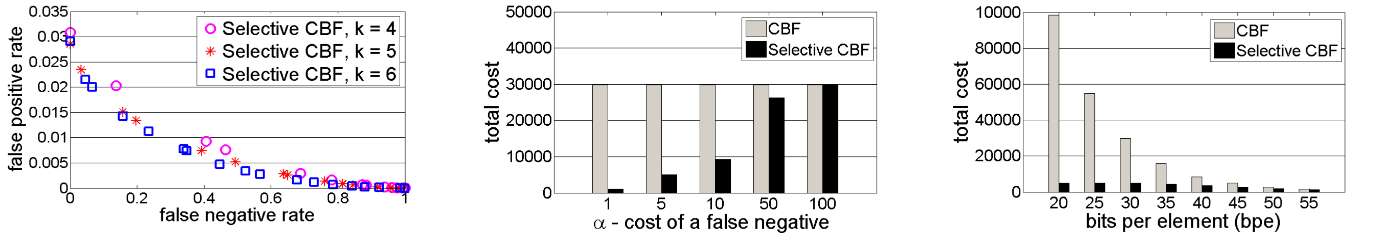
C. Trace-Driven Simulations

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 [36]. We used a 64-bit mix hash function [37] to implement the requested hash functions. The hash functions are calculated based on the 5-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 9(a) illustrates this tradeoff. 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



(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. 9. Trace-Driven Simulations

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.

Figure 9(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 $\epsilon(N - n) = 29856$ (where ϵ is the false positive rate). For small values of α, such that $\alpha = 1$ and $\alpha = 5$, 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 may still not be negligible, even for larger values of α. For instance, for $\alpha = 50$ the total cost is reduced by 12.09% to 26247. In practice, the α may reflect the latency difference between an SRAM memory access and a DRAM or an eDRAM (extended DRAM) memory access. If a DRAM access is 12.5 times slower than an SRAM access [38], then $\alpha = 12.5 - 1 = 11.5$, and therefore the cost is reduced by a factor of 3. Likewise, if an eDRAM access is 3 times slower than an SRAM access [39], then $\alpha = 3 - 1 = 2$, and the total cost is approximately reduced by an order of magnitude.

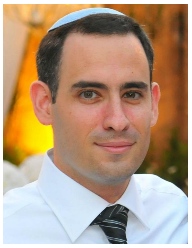
Figure 9(c) compares the total cost of queries of the CBF and the Selective CBF in the simulation above, as a function of the number of bits per element (bpe). For each value of bpe, the optimal number of hash functions of the CBF is used (in both schemes) and the results are presented for $\alpha = 5$. 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 bits per element, and thus also its total cost. In all cases, the Selective CBF achieves a 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 bpe=50, 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%.

IX. 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 that is required for the relevancy of the Bloom filter in its queries. We also showed that the exact values of the CBF counters can be easily used to calculate the set-membership probability. Last, we showed that our schemes significantly improve the average query cost.

X. Acknowledgment

We would like to thank Avinatan Hassidim for his helpful suggestions. This work was partly supported by the Google Europe Fellowship in Computer Networking, by the European Research Council Starting Grant No. 210389, by the Intel ICRI-CI Center, by the Israel Ministry of Science and Technology, by the Technion Funds for Security Research, and by the B. and G. Greenberg Research Fund (Ottawa).



Ori Rottenstreich received the B.S. in computer engineering (summa cum laude) and Ph.D. degree from the electrical engineering department of the Technion, Haifa, Israel in 2008 and 2014, respectively. His current research interests include exploring novel coding techniques for networking applications. He is a recipient of the Google Europe Fellowship in Computer Networking, the Andrew Viterbi graduate fellowship, the Jacobs-Qualcomm fellowship, the Intel graduate fellowship and the Gutwirth Memorial fellowship. He also received the Best Paper Runner Up Award at the IEEE Infocom 2013 conference.

Isaac Keslassy (M'02, SM'11) received his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 2000 and 2004, respectively.

He is currently an associate professor in the Electrical Engineering department of the Technion, Israel. His recent research interests include the design and analysis of high-performance routers and multi-core architectures. He is the recipient of the European Research Council Starting Grant, the Alon Fellowship, the Mani Teaching Award and the Yanai Teaching Award.

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. on Networking*, vol. 8, no. 3, 2000.

[3] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting Bloom filters," in *ESA*, 2006.

[4] 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.

[5] O. Rottenstreich, Y. Kanizo, and I. Keslassy, "The variable-increment counting Bloom filter," *IEEE/ACM Trans. on Networking*, 2014.

[6] Y. Kanizo, D. Hay, and I. Keslassy, "Access-efficient balanced Bloom filters," *Computer Communications*, vol. 36, no. 4, pp. 373–385, 2013.

[7] A. Z. Broder and M. Mitzenmacher, "Survey: Network applications of Bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, 2003.

[8] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using Bloom filters," *IEEE/ACM Trans. on Networking*, vol. 14, no. 2, pp. 397–409, 2006.

[9] 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.

[10] S. Dharmapurikar and J. W. Lockwood, "Fast and scalable pattern matching for network intrusion detection systems," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1781–1792, 2006.

[11] M. Antikainen, T. Aura, and M. Sarela, "Denial-of-service attacks in Bloom-filter-based forwarding," *IEEE/ACM Trans. on Networking*, 2014.

[12] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "Beyond Bloom filters: from approximate membership checks to approximate state machines," in *ACM SIGCOMM*, 2006.

[13] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM Operating Systems Review*, vol. 44, no. 2, 2010.

[14] "Google Chrome safe browsing." [Online]. Available: http://src.chromium.org/viewvc/chrome/trunk/src/chrome/browser/safe_browsing/

[15] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *FAST*, 2002.

[16] D. Ficara, A. D. Pietro, S. Giordano, G. Procissi, and F. Vitucci, "Enhancing counting Bloom filters through Huffman-coded multilayer structures," *IEEE/ACM Trans. on Networking*, vol. 18, no. 6, 2010.

[17] L. Li, B. Wang, and J. Lan, "A variable length counting Bloom filter," in *2nd Int'l Conf. on Computer Engineering and Technology*, 2010.

[18] E. Porat, "An optimal Bloom filter replacement based on matrix solving," in *CSR*, 2009.

[19] B. Donnet, B. Baynat, and T. Friedman, "Retouched Bloom filters: allowing networked applications to trade off selected false positives against false negatives," in *ACM CoNEXT*, 2006.

[20] 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.

[21] S. Lovett and E. Porat, "A lower bound for dynamic approximate membership data structures," in *IEEE FOCS*, 2010.

[22] C. E. Rothenberg, P. Jokela, P. Nikander, M. Sarela, and J. Ylitalo, "Self-routing denial-of-service resistant capabilities using in-packet Bloom filters," in *IEEE European Conference on Computer Network Defense*, 2009.

[23] M. Yu, A. Fabrikant, and J. Rexford, "Buffalo: Bloom filter forwarding architecture for large organizations," in *ACM CoNEXT*, 2009.

[24] 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.

[25] H. L. Vacher, "Quantitative literacy - drug testing, cancer screening, and the identification of igneous rocks," *Journal of Geoscience Education*, 2003.

[26] P. Beauchemin, G. Brassard, C. Crepeau, and C. Goutier, "Two observations on probabilistic primality testing," in *Crypto'*, 1986.

[27] M. Mitzenmacher, "Compressed Bloom filters," *IEEE/ACM Trans. on Networking*, vol. 10, no. 5, pp. 604–612, 2002.

[28] F. Putze, P. Sanders, and J. Singler, "Cache-,hash- and space-efficient Bloom filters," in *Workshop on Experimental Algorithms*, 2007.

[29] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better Bloom filter," in *ESA*, 2006.

[30] E. Berg and E. Hagersten, "Statcache: a probabilistic approach to efficient and accurate data locality analysis," in *ISPASS*, 2004.

[31] S. Ghosh, M. Martonosi, and S. Malik, "Cache miss equations: a compiler framework for analyzing and tuning memory behavior," *ACM Trans. Program. Lang. Syst.*, vol. 21, no. 4, pp. 703–746, 1999.

[32] X. Shen, J. Shaw, B. Meeker, and C. Ding, "Locality approximation using time," in *POPL*, 2007.

[33] Y. Zhong, S. G. Dropsho, X. Shen, A. Studer, and C. Ding, "Miss rate prediction across program inputs and cache configurations," *IEEE Trans. Computers*, vol. 56, no. 3, pp. 328–343, 2007.

[34] C. Cascaval and D. A. Padua, "Estimating cache misses and locality using stack distances," in *ICS*, 2003.

[35] M. L. Weitzman, "Optimal search for the best alternative," *Econometrica*, vol. 47, no. 3, pp. 641–654, 1979.

[36] 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/.

[37] T. Wang, "Integer hash function," http://www.concentric.net/∼Ttwang/tech/inthash.htm.

[38] S. Iyer, R. R. Kompella, and N. McKeown, "Designing packet buffers for router linecards," *IEEE/ACM Trans. on Networking*, vol. 16, no. 3, pp. 705–717, 2008.

[39] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Design exploration of hybrid caches with disparate memory technologies," *TACO*, vol. 7, no. 3, p. 15, 2010.