Capacity and Robustness Tradeoffs in Bloom Filters for Distributed Applications

Marcelo Duffles Donato Moreira, Rafael Pinaud Laufer, *Member*, *IEEE*, Pedro Braconnot Velloso, *Member*, *IEEE*, and Otto Carlos M.B. Duarte

Abstract—The Bloom filter is a space-efficient data structure often employed in distributed applications to save bandwidth during data exchange. These savings, however, come at the cost of errors in the shared data, which are usually assumed low enough to not disrupt the application. We argue that this assumption does not hold in a more hostile environment, such as the Internet, where attackers can send a carefully crafted Bloom filter in order to break the application. In this paper, we propose the *concatenated Bloom filter* (CBF), a robust Bloom filter that prevents the attacker from interfering on the shared information, protecting the application data while still providing space efficiency. Instead of using a single large filter, the CBF concatenates small subfilters to improve both the filter robustness and capacity. We propose three CBF variants and provide analytical results that show the efficacy of the CBF for different scenarios. We also evaluate the performance of our filter in an IP traceback application and simulation results confirm the effectiveness of the proposed mechanism in the face of attackers.

Index Terms—Bloom filters, distributed applications, security, IP traceback

1 Introduction

Distributed applications commonly need to share information to locate system resources [1]. As an example, nodes in a peer-to-peer (P2P) network need to share their content with each other to create a global index indicating where each file is located [2], [3]. In a similar application, proxy servers exchange the content of their web cache with each other to create a distributed cache across all servers [4]. Webpage requests directed to a given proxy server can then be forwarded to another proxy server which already has the page in its cache. Several other applications (e.g., content synchronization) also have the same requirement of exchanging a set of elements between different parties [5]. For these applications, Bloom filters (BFs) can be used to achieve significant bandwidth savings [6]. The Bloom filter is basically a space-efficient data structure that represents a set S. The cost of this compact set representation is the possibility of an external element $s \notin S$ being recognized as a legitimate element of S; an event known as a *false positive*. Nonetheless, for the above-mentioned applications, the space and processing savings generated by Bloom filters largely outweigh this drawback.

Despite its efficiency, the use of Bloom filters in distributed applications is limited by security concerns. The major concern is the filter ability of representing the entire universe of elements into a few bits. This occurs whenever the false-positive rate of the filter increases to 100 percent. A malicious filter with such an error rate can be easily generated to compromise distributed applications, causing hazardous effects. In web cache sharing and P2P network applications, if a node announces a maliciously adjusted filter, the other nodes believe that the malicious node has the webpages or objects under search. All application requests are then forwarded to the malicious node, which can replace the requested objects at will.

In this paper, we address the problem of bounding the false positive rate of Bloom filters to avoid the filter saturation attack. Even though it is easy to conduct such an attack, little has been studied on efficient mechanisms for securing Bloom filters. A simple solution to this attack would be to remove the elements inserted by the attacker from the filter. Nevertheless, this is not possible, because one cannot distinguish an element inserted by the attacker from an element inserted by a legitimate node. Thus, an alternative idea is to automatically remove the older elements as newer ones are inserted.

Following these guidelines, we propose a Bloom filter that is robust to the filter saturation attack, since the error rate an attacker can generate is always limited. We call it the *concatenated Bloom filter* (CBF) and its key idea is to divide the original filter into smaller subfilters, with each subfilter admitting only a few elements. The CBF is the result of the concatenation of these subfilters. We introduce the concept of *filter capacity* and show that, compared to previous work, the CBF is able to achieve a higher capacity while still being robust to attacks. Under certain conditions, the CBF is able to achieve a false-negative rate of 0 percent, with a similar

- *M.D.D. Moreira is with Grupo de Teleinformática e Automação (GTA), COPPE, Universidade Federal do Rio de Janeiro (UFRJ), and Intelie Research Lab, Paramopama St. 201/103, Ribeira, Rio de Janeiro, RJ 21930-110, Brazil. E-mail: marcelo@gta.ufrj.br.*
- *R.P. Laufer is with Bell Labs, Alcatel-Lucent, Room R-231, 791 Holmdel Rd., Holmdel, NJ 07733. E-mail: rafael.laufer@alcatel-lucent.com.*
- *P.B. Velloso is with the Instituto de Computação, Universidade Federal Fluminense, Rua Marqus de Valença 25/802, Tijuca, Rio de Janeiro, RJ 20550-030, Brazil. E-mail: velloso@ic.uff.br.*
- *O.C.M.B. Duarte is with the Grupo de Teleinformática e Automação (GTA), COPPE, Universidade Federal do Rio de Janeiro (UFRJ), Av. Sernambetiba, 16200, Apt 201, Recreio dos Bandeitantes, Rio de Janeiro, RJ 22795-006, Brazil. E-mail: otto@gta.ufrj.br.*

Manuscript received 20 Apr. 2011; revised 27 Oct. 2011; accepted 25 Feb. 2012; published online 7 Mar. 2012.
Recommended for acceptance by X. Tang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-04-0235.
Digital Object Identifier no. 10.1109/TPDS.2012.87.

behavior to the standard Bloom filter, but with the advantage of limited false positives. We derive the analytical expressions for the CBF error rates and show the different tradeoffs of this data structure.

The remainder of this paper is organized as follows. Section 2 presents the background and motivation for our work. We then present the CBF in Section 3 with a theoretical analysis in Section 4. An application for the CBF in an IP traceback system is presented in Section 5. Section 6 describes the related work. Finally, we present the conclusions in Section 7.

2 Background and Motivation

The standard Bloom filter is a space-efficient data structure used to represent a set $S = \{s_1, s_2, \dots, s_n\}$ of n elements. The filter is constituted by an m-bit array and by k independent hash functions $h_1, h_2, \dots, h_k$, whose outputs vary uniformly over the discrete space $\{0, 1, \dots, m-1\}$. The filter is constructed as follows. First, all bits in the array are reset. Then, each element $s_i \in S$ is inserted into the filter by setting the bits corresponding to positions $h_1(s_i), h_2(s_i), \dots, h_k(s_i)$ to 1. At the end of the process, membership queries can be conducted. To determine whether a given element x belongs to S, we verify if the bits at positions $h_1(x), h_2(x), \dots, h_k(x)$ are all set to 1. If at least one bit is in 0, then $x \notin S$ for sure. Otherwise, if all bits are set to 1, then we assume that $x \in S$. There is a probability, however, that an external element $x \notin S$ is recognized as a legitimate element of S. Such an event, called a *false positive*, occurs when the bits $h_1(x), h_2(x), \dots, h_k(x)$ were all set to 1 by other previously inserted elements. The false-positive probability of an element $x \notin S$ is given by $f_p^{BF} = (1-p)^k \approx (1 - e^{-kn/m})^k$, which represents the probability of finding a bit set to 1 for all the k positions indicated by the hash functions [7].

2.1 Application Model

Our application model considers a distributed system where the Bloom filter compactly represents the information shared by several nodes. Each node i inserts a set S_i into the Bloom filter and sends it to other nodes, therefore, after being updated by n nodes, the Bloom filter represents the set $\bigcup_{i=1}^{n} S_i$. This model is general and covers the previously mentioned applications, where the Bloom filter represents either a set of P2P objects or the set of URLs in a web cache.

Now, assume that one of the nodes is an attacker. Since the Bloom filter and most of its variants do not protect the state of the filter, the attacker is free to adjust it with arbitrary information and this information is propagated with the Bloom filter. We define this event as the *state-adjusting attack*. This attack can disrupt the application by several ways, since the attacker is able to

- inject spoofed elements into the filter to benefit itself or cause harm to other specific nodes;
- pollute the shared information with a burst of random data to sabotage the application;
- saturate the Bloom filter with all bits in 1, yielding a false-positive rate of 100 percent.

This attack describes the vulnerability of Bloom filters and its variants in distributed applications. Our proposal is to bound the false positive rate of the filter to limit the efficiency of such an attack. Since the CBF has a higher probability of "forgetting" older information, it can significantly decrease the impact of state-adjusting attacks. Note that the problem is not related to the authentication, privacy, or integrity of the transmitted filter, but rather to the ability of the Bloom filter in mapping the entire universe of elements into a few bits.

2.2 Securing the Bloom Filter

The *generalized Bloom filter* (GBF) was proposed as the first alternative to secure Bloom filters by limiting the false-positive rate, and thus avoiding the state-adjusting attack [8]. The key idea behind the GBF is to not only set, but also reset bits of the filter during each insertion. The filter is an extension of a BF with k_0 additional independent hash functions $g_1, g_2, \dots, g_{k_0}$ to reset bits. To insert an element $s_i \in S$ into the GBF, the bits corresponding to positions $g_1(s_i), g_2(s_i), \dots, g_{k_0}(s_i)$ are set to 0 and the bits corresponding to positions $h_1(s_i), h_2(s_i), \dots, h_{k_1}(s_i)$ are set to 1. If $g_j(s_i) = h_l(s_i)$, for some j and l, the bit is always reset. To determine if a given element x belongs to the set S, we check if the bits corresponding to positions $g_1(x), g_2(x), \dots, g_{k_0}(x)$ are set to 0 and if the bits corresponding to positions $h_1(x), h_2(x), \dots, h_{k_1}(x)$ are set to 1. If at least one bit is inverted, then it is assumed that $x \notin S$. Unlike the BF, now there is a possibility of an element $x \in S$ not being recognized by the filter, which is called a *false negative*. In the case that no bit is inverted, it is assumed that $x \in S$, even though there is also a chance that x does not belong to S (i.e., a false positive). The GBF, then, has both false positives and false negatives. The introduction of false negatives, however, is helpful to limit the effect of false positives, and both the false-positive and false-negative rates are proven to be upper bounded, regardless of the initial state of the filter.

Even though these rates are bounded, lower error rates are only achieved at the expense of a longer bit array, and thus a higher storage cost (i.e., a lower capacity). False negatives occur when a subsequent element inverts the marked bits of a previously inserted element. The existence of false negatives allows the filter to be independent from the initial state of the bit array and to have a bounded false-positive rate [8]; however, a noxious effect of false negatives is the loss of information, interpreted as a phenomenon of "forgetting" old elements. In other words, the existence of false negatives limits the filter capacity. Therefore, we propose a Bloom filter variant that reduces the false-negative rate, improving the state of the art in filter capacity, while still maintaining the robustness to attacks (i.e., a bounded false-positive rate). For more details on the GBF insertion procedure and the influence of false positives and false negatives on the GBF capacity, please see Appendix A in the supplement material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.87.

3 The Concatenated Bloom Filter

In this section, we introduce the *concatenated Bloom filter*. As the standard Bloom filter, the CBF is a data structure capable of representing a set in a compact form. The CBF, however, has additional features that make it more suitable for

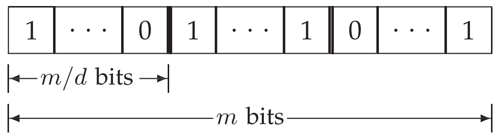


Fig. 1. The concatenated Bloom filter as the concatenation of d smaller subfilters, each with m/d bits.

distributed applications in which security issues are required. The key idea of the CBF is to concatenate small filters, called subfilters. Therefore, we have less insertions per subfilter and the false-negative probability can be arbitrarily reduced. In the limit, we can even have a null false-negative probability if just one element is inserted into each subfilter. We show that this idea greatly improves the filter capacity while maintaining the robustness to attacks.

Let m and d be positive integers such that m is a multiple of d. The CBF is defined as the concatenation of d smaller filters, called subfilters, each with m/d bits. The CBF is then composed of m bits, which are not restricted to any initial value. Fig. 1 illustrates this definition.

Let $S = \{s_1, s_2, \dots, s_n\}$ be the set of n elements to be inserted into the filter. To minimize the number of insertions per subfilter, inserted elements must be uniformly distributed among the d subfilters. Different approaches (e.g., a hash function) can be used for mapping input elements to the correct subfilters. For the scope of this work and for our application in Section 5, a simple circular operation is sufficient, i.e., element s_1 is inserted into the first subfilter; element s_2 is inserted into the second subfilter, and so on. When the last subfilter is reached, we go back to first subfilter, and hence element s_{d+1} is inserted into the first subfilter. This circular operation continues until all the elements are inserted.

We propose and analyze three variants for the CBF, namely CBF-1, CBF-2, and CBF-3. The insertion of an element $s_i \in S$ into the CBF is presented in Fig. 2. The procedure takes the element $s_i \in S$, the CBF m-bit array C, and a counter t as input arguments. The counter t is initialized to zero before all insertions, and indicates the subfilter where the current insertion should take place. Thus, we increase the counter t at the end of each insertion.

The CBF-1 variant uses a GBF for each subfilter. Therefore, the insertion procedure is similar to the one used in GBF, with the difference that the element is inserted into a smaller array of m/d bits. Bits which are neither set nor reset (i.e., untouched bits) are represented by the "X" label in the figure.

The CBF-2 variant is an adaptation of the insertion algorithm of the standard Bloom filter, but here zeros also represent information. The filter is initially reset, and then the bits corresponding to positions $h_1(s_i), \dots, h_k(s_i)$ are set to 1. The initial bit resetting erases the original content of the filter in order to increase the robustness to the attacker's information. The CBF-3 variant applies a hash function $H : S \rightarrow \{0, 1, \dots, 2^{m/d} - 1\}$ to s_i and replaces the original content of the subfilter with the hash function output. For detailed insertion algorithms, please see Appendix B.1, available in the online supplemental material. The parameters for each CBF variant can be found in Appendix B.2, available in the online supplemental material.

For the CBF-2 and CBF-3, the original content of the subfilter is erased to increase the robustness to attacks. On the other hand, if a subfilter is overwritten, it is very likely that the "signature" of the new element will be different than the "signature" of the previous element. Each subfilter then stores only one element and thus the CBF stores a total of d elements.

Different from these variants, the CBF-1 is general and admits an *unlimited* number of insertions, regardless of the value of d. It does not abruptly erase the original content of the subfilter, but rather smoothly deletes older elements with bit inversions as new elements are inserted into the filter.

The membership test of an element $x \notin S$ is simple. First, we need to know the subfilter where x may be located. In the context of our application in Section 5, the counter t is available at the end of the insertions and, therefore, we know where the last insertion occurred. Elements are also tested in the reverse order of insertion, starting with the last inserted element s_n, then element s_{n-1}, and so on, until the first element s_1 is tested. To allow a more general query sequence, a hash function (instead of a counter) can be used during insertions to select the subfilter of each element, and the same function must be used in the membership test to locate the correct subfilter. The counter approach, however, allows the filter to behave as a circular buffer and it is more suitable for certain applications.

The membership test changes according to the CBF variant used. For CBF-1, the procedure first checks if any of the bits at positions $g_1(x), \dots, g_{k_0}(x)$, which should be set to 0 if the element was inserted, are set to 1. If at least one bit is set to 1, then the procedure stops and returns false. Otherwise, it checks if the bits at positions $h_1(x), \dots, h_{k_1}(x)$

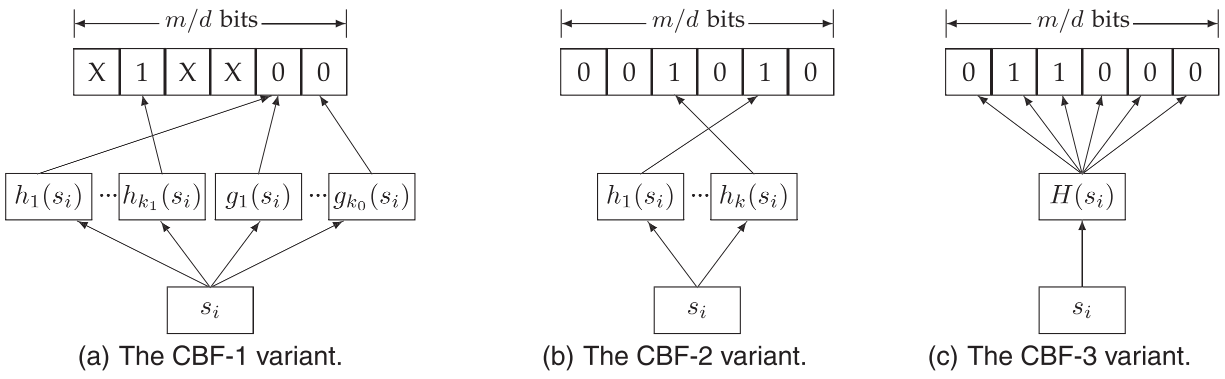


Fig. 2. The insertion of element s_i in the three variants of the CBF.

are set to 0 and if $h_j(x) \neq g_l(x)$, for $l = 1, \ldots, k_0$, because in a collision the functions g_l have precedence over the functions h_j. If the two conditions hold for at least one bit, the procedure stops and returns false. For CBF-2, the algorithm first allocates and resets an (m/d)-bit vector V. It then inserts x into V and compares V with the corresponding subfilter in C. The procedure returns true if both are equal. For CBF-3, the logic is pretty much the same, except that V now stores $H(x)$, the hash value of x. The procedure returns true only if the corresponding subfilter is equal to $H(x)$. At last, we decrease the counter t to update the subfilter index of the next membership test. The membership test algorithms can be found in Appendix B.3, available in the online supplemental material.

4 Analytical Results

In this section, we derive analytical expressions for the false-positive and false-negative rates of the proposed *concatenated Bloom filter*. Since the CBF is composed of small subfilters, we introduce in this work the analysis of small Bloom filters, considering the case where the number of hash functions is not negligible when compared to the filter size, a scenario not covered by previous work. We extend the analysis of the standard Bloom filter and the generalized Bloom filter to small filters so that they can be used as subfilters of the CBF. Finally, we introduce the concept of *filter capacity* and compare the evaluated filters with regards to this new metric. Most of our results consider the ratio m/n between the filter size and the number of elements, which provides a better idea of memory occupancy than if the parameters m and n were used individually.

4.1 False Positives

A false positive occurs when an external element $x \notin S$ is recognized by the filter as an element of S. We now derive the analytical expressions for the false-positive rates for each CBF variant introduced in Section 3.

4.1.1 The CBF-1 Variant

To calculate the false-positive rate of an m-bit CBF-1, we first compute the probabilities q_0 and q_1 of a bit being set to 0 or 1, respectively, during an insertion into a subfilter of m/d bits. Recalling that, in a collision, the functions g_j always have precedence over the functions h_l, the probability q_0 of a specific bit being reset is the probability that at least one of the k_0 hash functions reset the bit; that is, $q_0 = 1 - (1 - d/m)^{k_0}$. The probability q_1 of a specific bit being set by an insertion is the probability that at least one of the k_1 hash functions set the bit and none of the k_0 hash functions reset the bit. Therefore, $q_1 = [1 - (1 - d/m)^{k_1}](1 - d/m)^{k_0}$. Finally, the probability that a specific bit is neither set nor reset by the insertion, i.e., it remains untouched, is then $1 - q_0 - q_1 = (1 - d/m)^{k_0+k_1}$. The probabilities q_0 and q_1 can also be seen as the fraction of bits reset and the fraction of bits set by each insertion into the subfilter. For an (m/d)-bit subfilter, we then have $q_0(m/d)$ bits reset, $q_1(m/d)$ bits set, and $(1 - q_0 - q_1)m/d$ bits untouched by each insertion.

Assume that n is the total number of elements to be inserted into the filter and, thus a maximum of $\lceil n/d \rceil$ elements are inserted into each subfilter, where $\lceil x \rceil$ is the smallest integer not less than x. Let p be the probability that a specific bit in the subfilter is 0 after $\lceil n/d \rceil$ insertions. This probability can be calculated from the probability of the following $\lceil n/d \rceil + 1$ mutually exclusive events. The first event is when the bit is initially reset and it is not touched by any of the $\lceil n/d \rceil$ insertions. This happens with probability $p_0(1 - q_0 - q_1)^{\lceil n/d \rceil}$, where p_0 is the probability of the bit being in 0 in the initial state of the filter. The other $\lceil n/d \rceil$ events are when the bit is reset by the ith insertion and it is not touched anymore by the remaining $\lceil n/d \rceil - i$ elements. The probability of each of these events is $q_0(1 - q_0 - q_1)^{\lceil n/d \rceil - i}$. The probability p that a given bit is 0 after the $\lceil n/d \rceil$ insertions is then

$$\begin{aligned} p &= p_0(1 - q_0 - q_1)^{\lceil n/d \rceil} + \sum_{i=1}^{\lceil n/d \rceil} q_0(1 - q_0 - q_1)^{\lceil n/d \rceil - i} \\ &= p_0(1 - q_0 - q_1)^{\lceil n/d \rceil} + \frac{q_0}{q_0 + q_1}\left[1 - (1 - q_0 - q_1)^{\lceil n/d \rceil}\right]. \end{aligned} \tag{1}$$

This probability p can also be seen as the fraction of bits in 0 after $\lceil n/d \rceil$ insertions into the subfilter.

Recall that, on average, $q_0(m/d)$ bits are reset and $q_1(m/d)$ bits are set at each insertion. This means that for an element $x \notin S$ to be recognized by the filter, we must find all of its $q_0(m/d)$ bits in 0 and all of its $q_1(m/d)$ bits in 1. The false-positive probability of $x \notin S$ is then the probability of finding its $q_0(m/d)$ bits in 0 and its $q_1(m/d)$ bits in 1 on the specific subfilter of x, i.e.,

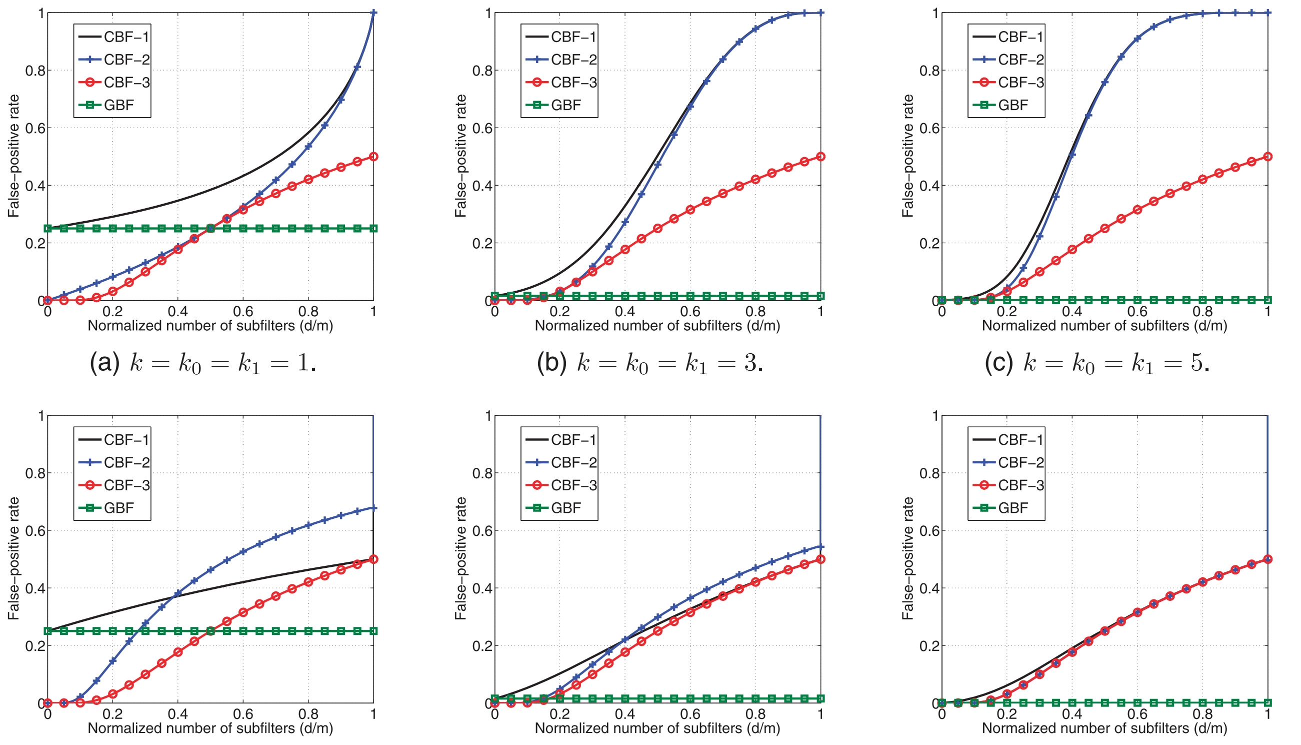
$$f_p^{\mathrm{CBF-1}} = p^{q_0 m/d}(1 - p)^{q_1 m/d} = \left[p^{q_0}(1 - p)^{q_1}\right]^{m/d}. \tag{2}$$

This generalizes the false-positive expression of the GBF for small filters, considering the case where the approximations $m \gg k_0$ and $m \gg k_1$ are not valid.

There is a tradeoff between the false-positive rate and p. Fig. 3, in Appendix B.4, available in the online supplemental material, depicts the false-positive rate of CBF-1 from (2) as a function of p. For a high p, we have most bits in 0, but very few in 1. For a low p, the reverse behavior is expected, where we have most bits in 1, but very few in 0. Since for a false-positive we need to find both bits in 0 and in 1, the false-positive rate is zero at these locations. We also observe that the false-positive rate of CBF-1 is limited for $m/d > 1$. This means that, regardless of the initial state of the filter, an attacker cannot generate more false positives than the upper bound of the filter. We can also see that this limit decreases for higher values of m/d. By equating to zero the derivative of $f_p^{\mathrm{CBF-1}}$ with respect to p, we find that the maximum false-positive rate occurs when $p = q_0/(q_0 + q_1)$. From (1), we know that p reaches this value when either the initial state of the filter is set to $p_0 = q_0/(q_0 + q_1)$, or when $\lceil n/d \rceil \to \infty$. Putting this back in (2), we find that the maximum false-positive rate $F_p^{\mathrm{CBF-1}}$ is

$$F_p^{\mathrm{CBF-1}} = \left[\left(\frac{q_0}{q_0 + q_1}\right)^{\frac{q_0}{q_0+q_1}} \left(\frac{q_1}{q_0 + q_1}\right)^{\frac{q_1}{q_0+q_1}}\right]^{(q_0+q_1)m/d}. \tag{3}$$

For a fixed value of $q_0 + q_1$, the expression within brackets in (3) is minimized when $q_0 = q_1$ and it is equal to $1/2$. Having $q_0 = q_1$ means that, during an insertion, the probability of setting or resetting a bit is the same. In this case, (3) is simplified to $F_p^{\mathrm{CBF-1}} = 0.5^{(q_0+q_1)m/d}$. The value of $q_0 + q_1$ in the



(a) $k = k_0 = k_1 = 1$. (b) $k = k_0 = k_1 = 3$. (c) $k = k_0 = k_1 = 5$.

(d) $k_1 = 1, k_0 = f(k_1, m/d), k = 0.2k^*$. (e) $k_1 = 3, k_0 = f(k_1, m/d), k = 0.5k^*$. (f) $k_1 = 5, k_0 = f(k_1, m/d), k = k^*$.

Fig. 3. The false-positive rate as a function of the normalized number of subfilters (d/m) and the number of hash functions, for $m = 1{,}024$ bits.

exponent is $1 - (1 - d/m)^{k_0+k_1}$ and it is maximized to 1 for large k_0 or k_1. Thus, if we choose relatively large values for k_0 and k_1, such that $q_0 = q_1$, the maximum false-positive rate $F_p^{\mathrm{CBF-1}}$ is minimized to

$$\min_{k_0,k_1} F_p^{\mathrm{CBF-1}} = 0.5^{m/d}. \tag{4}$$

From (3), we also know that $F_p^{\mathrm{CBF-1}}$ is maximized to

$$\max_{k_0,k_1} F_p^{\mathrm{CBF-1}} = 1 \tag{5}$$

when either $q_0 = 0$ or $q_1 = 0$, which only occurs for the standard Bloom filter ($k_0 = 0$), its dual ($k_1 = 0$), or for 1-bit subfilters, i.e., $m/d = 1$. Otherwise, $F_p^{\mathrm{CBF-1}} < 1$.

4.1.2 *The CBF-2 Variant*

For the CBF-2, a false positive occurs when an external element $x \notin S$ has the same “signature” as the last element inserted into the subfilter. Let p be the probability that a bit is 0 in a given subfilter of CBF-2. As the subfilter is reset before each insertion, p is the probability that none of the k hash functions set the bit. Therefore, $p = (1 - d/m)^k$. After each insertion, we then have pm/d bits of the subfilter in 0 and $(1 - p)m/d$ bits in 1. The false-positive rate $f_p^{\mathrm{CBF-2}}$ for an element $x \notin S$ is then the probability of finding all of its pm/d bits in 0 and all of its $(1 - p)m/d$ bits in 1, and thus

$$f_p^{\mathrm{CBF-2}} = p^{\,pm/d}(1 - p)^{(1-p)m/d} = \left[p^{\,p}(1 - p)^{(1-p)}\right]^{m/d}. \tag{6}$$

The expression within brackets in (6) is minimized when $p = 1/2$. In this case, we have

$$\min_k f_p^{\mathrm{CBF-2}} = 0.5^{m/d}, \tag{7}$$

and the optimal number of hash functions k^* that achieves this rate is

$$k^* = -\frac{\ln(2)}{\ln\left(1 - \dfrac{1}{m/d}\right)}. \tag{8}$$

The maximum false-positive rate is $\max_k f_p^{\mathrm{CBF-2}} = 1$, which occurs when $k = 0$ or $k \to \infty$, as we explain in Appendix B.5, available in the online supplemental material.

4.1.3 *The CBF-3 Variant*

For CBF-3, since each subfilter contains the hash value of the last element inserted, the false-positive rate is the probability of the element $x \notin S$ being tested having the same hash value of the last element inserted, i.e.,

$$f_p^{\mathrm{CBF-3}} = 0.5^{m/d}. \tag{9}$$

4.1.4 *Comparison*

We compare our proposal only to the GBF because, to the best of our knowledge, the GBF is the only data structure designed to address the same issues as the CBF. Other structures assume that the filter is always initialized to all zeros and that there are no threats within the network [4], [9], [10], [11], [12].

Fig. 3 shows the false-positive rates for CBF-1, CBF-2, CBF-3, and the GBF for an array of $m = 1{,}024$ bits, as we change the number of hash functions and the number of subfilters d. For each CBF variant, we fix m and continuously increase d. The x-axis is the normalized ratio d/m, such that the value $1/m$ on the x-axis represents a single m-bit filter and the value 1.0 represents m 1-bit subfilters. For the GBF, we assume that all m bits are allocated for the filter (i.e., d has no effect over the result) and it is presented here just for comparison. Both the GBF and the subfilters in the CBF-1 are initialized to the worst possible condition, namely $p_0 = q_0/(q_0 + q_1)$.

Figs. 3a, 3b, and 3c present the false-positive rate of each CBF variant, as we increase the number of subfilters d and the number of hash functions. The curve for CBF-3 is $0.5^{m/d}$, the lower bound for all CBF variants. In Fig. 3a, we fix $k = k_0 = k_1 = 1$ and, at this configuration, the GBF has a false-positive rate of 25 percent. The CBF-1 has the same error rate as the GBF when $d = 1$, since in this case the GBF and CBF-1 are the same. At these parameters, we have $q_0 \approx q_1$ for $k_0 = k_1$. As d increases, however, we leave this optimal configuration and q_0 becomes higher and higher than q_1, resulting in a higher fraction of bits in 0. When $d = m$, we have only one bit per subfilter with $q_0 = 1$ and $q_1 = 0$, leading to this bit being always in 0 and a false-positive rate of 100 percent. For CBF-2, we have a similar effect. As d increases, the fraction of bits in 1 also increases, reaching the 100 percent limit when $d = m$. For CBF-3, on the other hand, we have no dependency on the number of hash functions, only on the size of the subfilter m/d. The point where the curves of CBF-2 and CBF-3 touch is when $p = 1/2$ for CBF-2.

We see that CBF-1 has a higher false-positive rate than CBF-2, which, in turn, has a higher false-positive rate than CBF-3. In fact, this always holds for $k = k_0 = k_1$. The advantage of CBF-1 is the reduced false-negative rate and higher capacity, as shown in the next sections.

If we increase the number of hash functions, as in Figs. 3b and 3c, the false-positive rate of the GBF is significantly reduced to 1.58 and 0.10 percent, respectively. However, we can always find a value of d, such that CBF-2 and CBF-3 have a lower false-positive rate. In our case, for $k = 3$, we have $f_p \leq 1.58\%$ in CBF-2 and CBF-3 if $m/d \geq 7$ and $m/d \geq 6$, respectively. For $k = 5$ and $f_p \leq 0.10\%$, we must have $m/d \geq 11$ for CBF-2 and $m/d \geq 10$ for CBF-3.

Figs. 3b and 3c also show that the false-positive rate of CBF-1 and CBF-2 becomes closer to each other. This effect is seen in (3) as we increase k_0 and k_1. In this case, we have $q_0 + q_1 = 1 - (1 - d/m)^{k_0+k_1} \approx 1$ and $q_1 \approx 1 - q_0 = (1- d/m)^{k_0} = (1 - d/m)^k$, resulting in (6). From the figures, we also see that a larger $k = k_0 = k_1$ results in a lower false-positive rate for large subfilters (i.e., small d/m), but a higher false-positive rate for small subfilters (i.e., large d/m).

Figs. 3d, 3e, and 3f present the case where the number of hash functions is optimized for each CBF variant. For CBF-1, we fix k_1 and compute $k_0 = f(k_1, m/d)$ as a function of both k_1 and m/d, to guarantee that $q_0 = q_1$. For CBF-2, we use $k = \alpha k^*$, where $0 \leq \alpha \leq 1$ and k^* is defined in (8). For this choice of hash functions, the false-positive rates of CBF-1 and CBF-2 are bounded for $d < m$, suggesting that a careful selection of hash functions is necessary. If we maintain $q_0 = q_1$ and increase k_0 and k_1 for CBF-1 or increase α for CBF-2, we see in Figs. 3e and 3f that their false-positive rates get closer to the lower bound $0.5^{m/d}$, confirming the results in (4) and (7).

4.2 False Negatives

A false negative occurs when an element $s_i \in S$ is not recognized by the filter as an element of S. In this section, we derive the analytical expressions for the false-negative rate for each CBF variant.

4.2.1 The CBF-1 Variant

For CBF-1, a false negative occurs for an element $s_i \in S$ if at least one of its marked bits is inverted at the end of the insertions (i.e., a bit set by s_i is 0 or a bit reset is 1). Elements inserted first are more susceptible to bit inversions, because each succeeding element can invert their bits. Different from the false-positive rate, where the error rate is the same for every element $x \notin S$, the false-negative rate for an element $s_i \in S$ depends on the specific element. In fact, it depends on how many elements are inserted into the same subfilter after s_i. For a circular insertion (cf. Section 2.1), the number of succeeding insertions n_i into the same subfilter after s_i is $n_i = \lfloor (n - i)/d \rfloor$, where $\lfloor x \rfloor$ is the integer part of x.

Let $u_0(i)$ be the probability that a bit reset by s_i remains in 0 after the remaining n_i insertions into the same subfilter. This probability can be calculated from $n_i + 1$ mutually exclusive events. The first event is when none of the remaining elements touch the bit, which occurs with probability $(1 - q_0 - q_1)^{n_i}$. The others are when the bit is reset by the $(n_i - j)$th element, and not touched by the following j insertions, for $0 \leq j \leq n_i - 1$. The probability $u_0(s_i)$ is then

$$\begin{aligned} u_0(s_i) &= (1 - q_0 - q_1)^{n_i} + \sum_{j=0}^{n_i-1} q_0(1 - q_0 - q_1)^j \\ &= (1 - q_0 - q_1)^{n_i} + \frac{q_0}{q_0 + q_1}[1 - (1 - q_0 - q_1)^{n_i}]. \end{aligned} \tag{10}$$

Likewise, the probability $u_1(s_i)$ that a bit set by s_i remains in 1 after the remaining n_i insertions is

$$\begin{aligned} u_1(s_i) &= (1 - q_0 - q_1)^{n_i} + \sum_{j=0}^{n_i-1} q_1(1 - q_0 - q_1)^j \\ &= (1 - q_0 - q_1)^{n_i} + \frac{q_1}{q_0 + q_1}[1 - (1 - q_0 - q_1)^{n_i}]. \end{aligned} \tag{11}$$

Since, on average, an element resets $q_0(m/d)$ bits and sets $q_1(m/d)$ bits, the false-negative probability $f_n^{\mathrm{CBF-1}}(s_i)$ of element s_i is the probability that at least one of its bits is inverted at the end of the insertions, i.e.,

$$\begin{aligned} f_n^{\mathrm{CBF-1}}(s_i) &= 1 - u_0(s_i)^{q_0 m/d} u_1(s_i)^{q_1 m/d} \\ &= 1 - \left[u_0(s_i)^{q_0} u_1(s_i)^{q_1}\right]^{m/d}. \end{aligned} \tag{12}$$

From (10) and (11), we know that u_0 and u_1 increase as n_i decreases. This means that elements inserted first always have a higher false-negative rate than elements inserted later. As a result, $f_n^{\mathrm{CBF-1}}$ is a decreasing function of s_i. The maximum false-negative rate $F_n^{\mathrm{CBF-1}}$ over all elements s_i is then the false-negative rate of the first element s_1, and thus

$$F_n^{\mathrm{CBF-1}} = f_n^{\mathrm{CBF-1}}(s_1) = 1 - [u_0(s_1)^{q_0} u_1(s_1)^{q_1}]^{m/d}. \tag{13}$$

In Fig. 4a, we show the maximum false-negative rate $F_n^{\mathrm{CBF-1}}$ of CBF-1 as a function of n, the total number of insertions. We use $m = 1{,}024$ and show different curves for

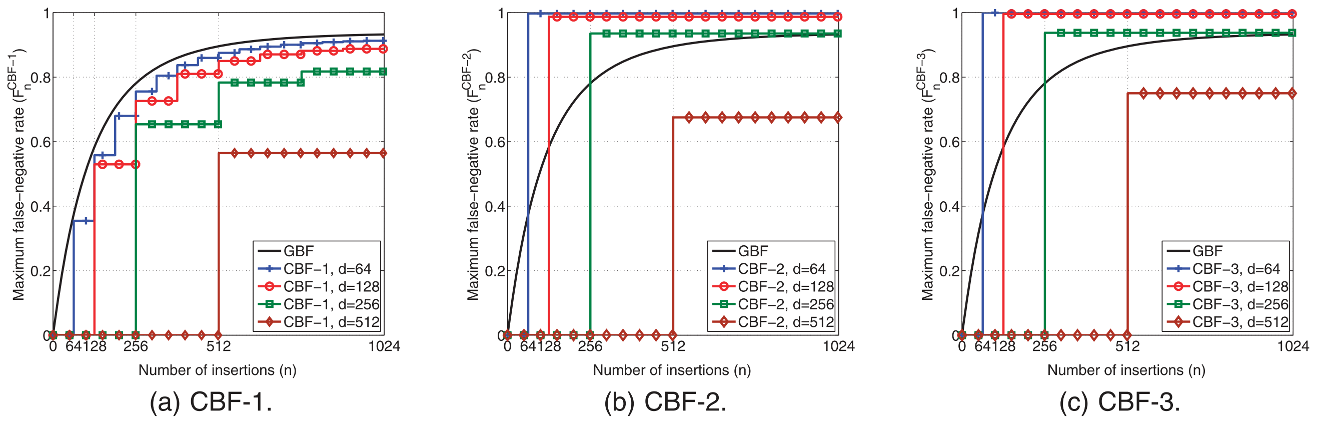


Fig. 4. The maximum false-negative rate as a function of n, for $k = k_0 = k_1 = 2$ and $m = 1{,}024$ bits.

d, the number of subfilters. Each step in the figure occurs when $\lfloor n/d \rfloor$ increases. We see that, when $n \leq d$, CBF-1 has no false negatives, since we have at most one element per subfilter. When $n > d$, CBF-1 presents false negatives, but the false-negative rate is always lower than the rate of GBF. Therefore, with regard to false negatives, it is always better to split the original GBF into d small subfilters. A larger d also is guaranteed to always reduce false negatives further.

The false-negative rate in (13) can be minimized to

$$\min_{k_0,k_1} F_n^{\mathrm{CBF}-2} = 0 \tag{14}$$

if $q_0 = 0$, $q_1 = 0$, or $n_1 = 0$. The first two conditions represent the standard Bloom filter ($k_0 = 0$) and its dual ($k_1 = 0$). An advantage of the standard Bloom filter is never erasing any inserted element; on the other hand, its false-positive rate is unbounded, which makes the filter vulnerable to state-adjusting attacks (cf. Section 2.1). A small rate of false negatives is therefore beneficial to make the filter more robust. The second condition, $q_1 = 0$, also holds for 1-bit subfilters (i.e., $m/d = 1$). The third condition, $n_1 = 0$, only holds if $n \leq d$. In this case, we have at most one element per subfilter and there is no succeeding element to invert their bits.

For a fixed value of $q_0 + q_1$, the expression within brackets in (13) is minimized when $q_0 = q_1$. In this case, we have $u_0(s_1) = u_1(s_1)$ and therefore

$$F_n^{\mathrm{CBF}-1} = 1 - \left[0.5 + 0.5(1 - q_0 - q_1)^{\lfloor (n-1)/d \rfloor}\right]^{(q_0+q_1)m/d}. \tag{15}$$

The value of $q_0 + q_1$ in the exponent is $1 - (1 - d/m)^{k_0+k_1}$ and it is maximized to 1 for large k_0 or k_1. In this case, $(1 - q_0 - q_1)^{\lfloor (n-1)/d \rfloor}$ is minimized to 0. Thus, if we choose relatively large values for k_0 and k_1, such that $q_0 = q_1$, then $F_n^{\mathrm{CBF}-1}$ is maximized to

$$\max_{k_0,k_1} F_n^{\mathrm{CBF}-1} = 1 - 0.5^{m/d}. \tag{16}$$

Note that the condition $q_0 = q_1$, which maximizes false negatives in (16), is the same condition that minimizes false positives in (4). Moreover, the conditions $q_0 = 0$ and $q_1 = 0$, which minimize false negatives in (14), are the same conditions that maximize false positives in (5). Therefore, it is not possible to jointly minimize both false positives and false negatives, unless $n \leq d$. If the number of insertions n is less than or equal to the number of subfilters d, then we can practically eliminate both false positives and false negatives. In this case, the CBF-1 behaves as CBF-3, with a false-negative rate of 0 percent and a false-positive rate of $0.5^{m/d}$ (cf. Section 4.2.3). The key advantage of CBF-1, however, is for $n > d$, since it has less false negatives and therefore a higher capacity than other variants, as we see next.

4.2.2 The CBF-2 Variant

For CBF-2, a false negative occurs if another element with a different "signature" is inserted into the same subfilter. If $n_i = 0$, this event cannot occur. For $n_i > 0$, if $p = (1 - d/m)^k$ is the fraction of bits in 0, this event occurs with probability $1 - [p^p(1-p)^{(1-p)}]^{m/d}$. The false-negative probability $f_n^{\mathrm{CBF}-2}(s_i)$ of an element s_i is then

$$f_n^{\mathrm{CBF}-2}(s_i) = \begin{cases} 0 & \text{if } n_i = 0 \\ 1 - \left[p^p(1-p)^{(1-p)}\right]^{m/d} & \text{if } n_i > 0. \end{cases} \tag{17}$$

As in CBF-1, the maximum false-negative rate $F_n^{\mathrm{CBF}-2}$ of CBF-2 can be defined as the false-negative rate of the first element s_1, as $F_n^{\mathrm{CBF}-2} = f_n^{\mathrm{CBF}-2}(s_1)$. Fig. 4b shows $F_n^{\mathrm{CBF}-2}$ as a function of n, the number of insertions, for different values of d. When $n \leq d$, CBF-2 has no false negatives, since we have at most one element per subfilter. However, when $n > d$, the false-negative rate abruptly increases to almost 100 percent, and it is much higher than the rate of GBF. This happens because the entire subfilter is replaced and the probability of the new "signature" being equal to the previous "signature" is very low, except for small subfilters (i.e., $m/d = 2$).

We know from (6) and (17) that $F_n^{\mathrm{CBF}-2} = 1 - F_p^{\mathrm{CBF}-2}$. Thus, $\min_k F_n^{\mathrm{CBF}-2} = 0$ occurs when $k = 0$ or $k \to \infty$, and

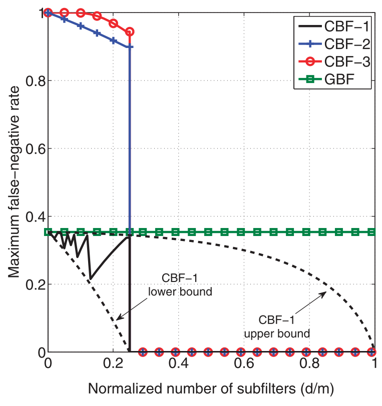
$$\max_k F_n^{\mathrm{CBF}-2} = 1 - 0.5^{m/d} \tag{18}$$

occurs for $k = k^*$, where k^* is defined in (8).

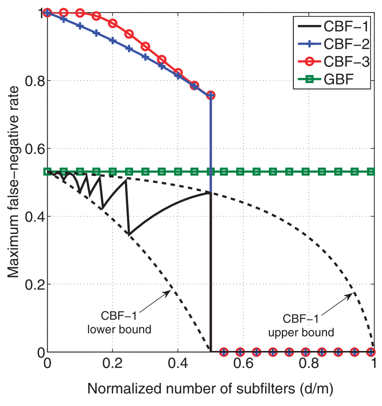
4.2.3 The CBF-3 Variant

Similar to CBF-2, a false negative occurs in CBF-3 only if the hash value of the new element is different than the hash value of a previously inserted element. The false-negative probability $f_n^{\mathrm{CBF}-3}(s_i)$ of an element s_i is then

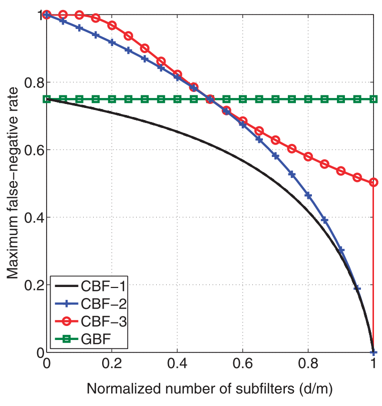
$$f_n^{\mathrm{CBF}-3}(s_i) = \begin{cases} 0 & \text{if } n_i = 0 \\ 1 - 0.5^{m/d} & \text{if } n_i > 0. \end{cases} \tag{19}$$



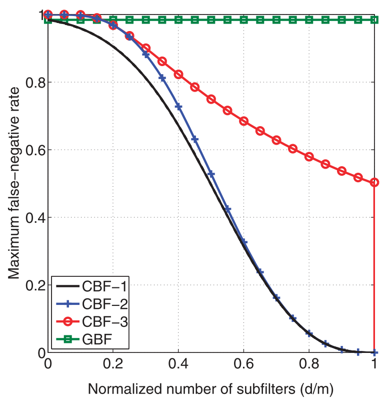
(a) $k = k_0 = k_1 = 1, n = 256$.



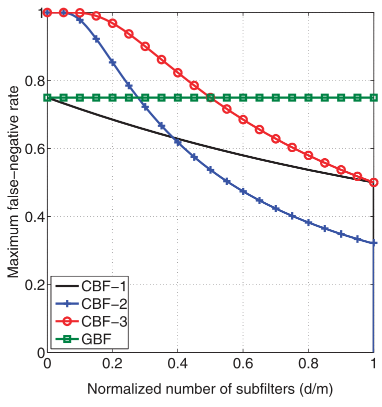
(b) $k = k_0 = k_1 = 1, n = 512$.



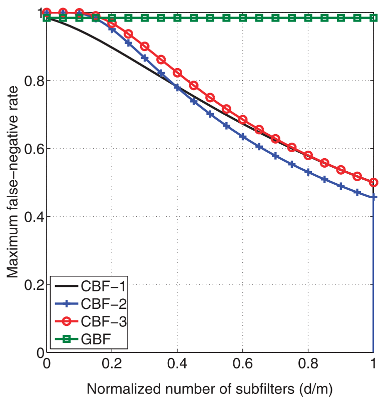
(c) $k = k_0 = k_1 = 1, n \rightarrow \infty$.



(d) $k = k_0 = k_1 = 3, n \rightarrow \infty$.



(e) $k_1 = 1, k_0 = f(k_1, m/d), k = 0.2k^*$.



(f) $k_1 = 3, k_0 = f(k_1, m/d), k = 0.5k^*$.

Fig. 5. The maximum false-negative rate as a function of the normalized number of subfilters (m/d), the number of insertions (n), and the number of hash functions, for $m = 1{,}024$.

As in the previous variants, the maximum false-negative rate of CBF-3 is defined as $F_n^{\text{CBF}-3} = f_n^{\text{CBF}-3}(s_1)$. Fig. 4c shows $F_n^{\text{CBF}-3}$ as a function of n for, different d values. Since CBF-3 also replaces all bits in the subfilter, its behavior is similar to CBF-2 in Fig. 4b, with slightly higher false-negative rates for CBF-3.

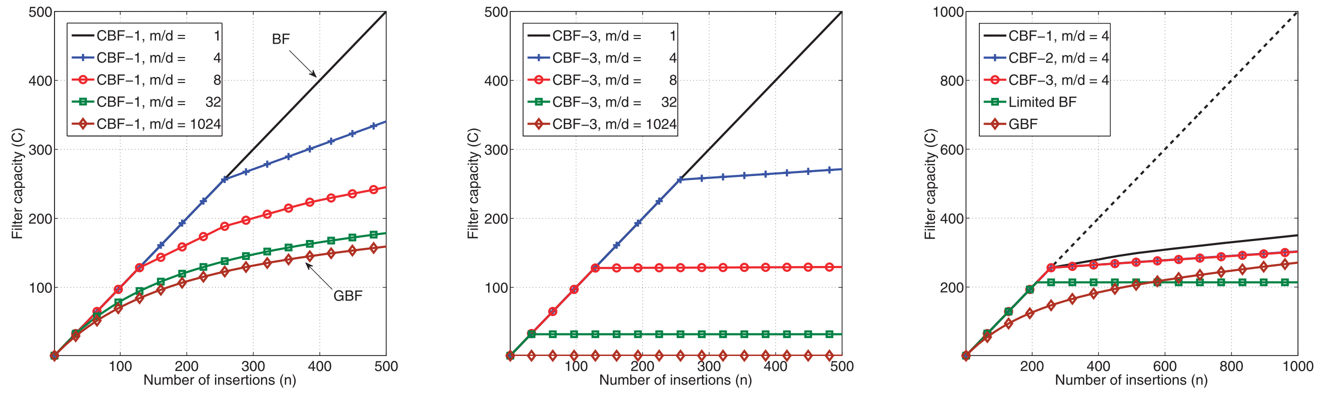
For both CBF-2 and CBF-3, false negatives are minimized when false positives are maximized. Thus, we cannot jointly minimize both error rates, unless $n \leq d$. Since false-negative rates are high for these variants, each subfilter only stores a single element in practice. The CBF-2 and CBF-3 are then ideal when $n \leq d$ since, in this condition, we have no false negatives and very low false positives (cf. Sections 4.1.2 and 4.1.3). When $n > d$, however, CBF-1 is the best choice because it remembers the last d elements with a probability of 100 percent and it also remembers the previous $n - d$ elements with a certain probability.

4.2.4 Comparison

Fig. 5 shows the maximum false-negative rate for the GBF and each CBF variant, for an array of $m = 1{,}024$ bits, as we change the number of insertions n, the number of subfilters d, and the number of hash functions. For each variant, we fix m and continuously increase d, with the x-axis representing the normalized ratio d/m. For the GBF, we assume that all m bits are allocated for the filter (i.e., d has no effect) and it is presented here just for comparison.

Figs. 5a and 5b depict the CBF behavior as we change d and n. For $d \geq n$, we have no false negatives because we have at most one element per subfilter. For $d < n$, we have a ripple for CBF-1. This is explained because increasing d has a dual effect. On one hand, with more subfilters (i.e., a higher d), we have less insertions per subfilter and less false negatives. On the other hand, the size m/d of each subfilter is smaller with a higher d, resulting in more bit inversions and false negatives. The parts of the curve where the false-negative rate increases are when the increase in d is still not enough to reduce the number of elements per subfilter, $\lfloor (n-1)/d \rfloor$. In this case, the decrease in the subfilter size m/d increases false negatives. When d increases such that $\lfloor (n-1)/d \rfloor$ is reduced, the false-negative rate steps down. The upper bound of CBF-1 is computed assuming $(n-1)/d$ insertions (i.e., no rounding); the lower bound is computed assuming $(n-d)/d$ insertions. Fig. 5c depicts the case where $n \rightarrow \infty$. In this case, both bounds are the same, since $(n-1)/d \approx (n-d)/d \approx n/d$ for large n.

Figs. 5c and 5d show the CBF behavior when we linearly increase $k = k_0 = k_1$ for the worst case $n \rightarrow \infty$. The curve of CBF-3 is $1 - 0.5^{m/d}$, the upper bound for all CBF variants. We can see that the false-negative rate of CBF-1 is always lower than the rate of GBF, and both are equal only when $d = 1$. Additionally, increasing the number of hash functions is harmful for a small d, but beneficial for large d for both CBF-1 and CBF-2. This occurs because, as d increases,



(a) CBF-1, with $k_0 = k_1 = 2$. (b) CBF-3. (c) BF variants, given $F_p \leq 10\%$.

Fig. 6. The capacity of Bloom filters as a function of the number of insertions (n), for $m = 1{,}024$.

$q_0 \rightarrow 1$ and $q_1 \rightarrow 0$ for CBF-1 and $k \rightarrow 0$ in CBF-2. Under these conditions, the false-negative rate is minimal for these variants. The point where the curves of CBF-2 and CBF-3 touch is when $p = 1/2$ for CBF-2.

Figs. 5e and 5f present the case where the number of hash functions is optimized to minimize false positives. For CBF-1, we fix k_1 and compute $k_0 = f(k_1, m/d)$ as a function of both k_1 and m/d to guarantee that $q_0 = q_1$. For CBF-2, we use $k = \alpha k^*$, where $0 \leq \alpha \leq 1$ and k^* is defined in (8). As we increase the number of hash functions under these conditions, the maximum false-negative rate gets closer to the upper bound $1 - 0.5^{m/d}$, confirming the results in (16) and (18). Finally, note that Figs. 5c and 5f are the complement of Figs. 3a and 3b and Figs. 3d and 3e, respectively, because when $n \rightarrow \infty$, we have $F_n = 1 - F_p$ for the GBF and all CBF variants.

4.3 Filter Capacity

We now introduce the concept of *filter capacity*. Given that inserted elements can be erased by subsequent elements, we define the filter capacity C as the average number of the inserted elements which are remembered at the end of insertions. Since each element is remembered with probability $1 - f_n(s_i)$, the filter capacity is

$$C = \sum_{i=1}^{n} 1 - f_n(s_i). \tag{20}$$

This definition is general and also handles the standard Bloom filter. In this case, we have no false negatives and the filter capacity is always n; that is, the filter stores every inserted element.

Fig. 6 depicts the filter capacity as a function of the number of insertions n, for $m = 1{,}024$. Fig. 6a shows the capacity of CBF-1, when we fix $k_0 = k_1 = 2$ and change the number of subfilters d. We see that the capacity always increases with d; this occurs because false negatives decrease with a larger d, as seen in Section 4.2. When $d = m$, we have one bit per subfilter and no false negatives, and thus the capacity grows linearly with n. Note that the curve for $m/d = 1$ and $m/d = 1{,}024$ are the same as the curves of the BF and the GBF, respectively. As a result, the CBF-1 allows the filter to behave closer to the BF or the GBF as we either increase or decrease d, respectively. Fig. 6b shows the capacity of CBF-3 as we change d. The capacity of CBF-2 is very similar and, thus, not shown. We can see that the filter capacity increases until $n = d$ and then it stabilizes. The CBF-2 and CBF-3, therefore, only remember the last d elements, whereas the CBF-1 also remembers the previous $n - d$ elements with a certain probability.

The results from Figs. 6a and 6b show that increasing the number of subfilters d always increases the capacity. However, a larger d also increases false positives, as seen in Section 4.1. The BF curve (i.e., CBF-1 for $m/d = 1$) in Fig. 6a, for instance, has a false-positive rate of 100 percent. As a result, a better definition for filter capacity C_f is the average number of elements remembered at the end of insertions, under the constraint that the false-positive rate of the filter is below or equal to f. This allows us to fairly compare the capacity of each filter.

Fig. 6c shows the filter capacity $C_{0.1}$ of the standard BF, the GBF, and each CBF variant, under the constraint that the false-positive rate is below or equal to 10 percent. For all CBF variants, the subfilter size $m/d = 4$ yields the highest capacity. Lower subfilter sizes (i.e., $m/d \leq 2$) are unable to satisfy the false-positive constraint, given that 10 percent is below the lower bound $0.5^{m/d} = 12.5\%$. The CBF-2 and CBF-3 present the same curve, which linearly increase until $n = d = 256$ and then roughly stabilizes. The CBF-1 has the highest capacity among all filters, since it is the filter with the lowest false-negative rate. For large n, both CBF-1 and GBF grow linearly with n at the same rate. When $n \rightarrow \infty$, we have $f_n(s_i) \rightarrow 1 - F_p$ for the GBF and all CBF variants. As a result, we know from (20) that the capacity of both filters increases linearly as $C_{0.1} = (F_p)n = (0.1)n$. The capacity also increases for CBF-2 and CBF-3, but at a much slower rate. In this case, $F_p^{\text{CBF}-2} \approx F_p^{\text{CBF}-3} = 0.5^{m/d} = 0.4\%$ and therefore the capacity grows with $C_{0.1} = (0.004)n$. The limited BF allows elements to be inserted until the false-positive threshold is reached [11]. We see that CBF and eventually the GBF have a higher capacity than the limited BF in this scenario. Note also that this BF is not robust to the initial state, since an attacker can prevent future insertions by initially saturating the filter to the false-positive threshold (cf. Section 2.1). Therefore, the limited BF is not suitable for distributed applications and we present it here just for comparison.

We conclude that CBF-2 and CBF-3 behave very similarly and should always be used when $n \leq d$. In this case, we have no false negatives and the false-positive rate is minimal at $0.5m/d$. Since CBF-3 only requires one hash function per insertion, we believe it is a better choice for this case. An application that fulfills this requirement is IP traceback, presented in Section 5. When $n > d$ or n is not known a priori, CBF-1 is the ideal choice, since for the same robustness level (i.e., a maximum false-positive rate) we have a higher capacity (i.e., a lower false-negative rate). In this case, however, the application must tolerate older elements being "forgotten" as new elements arrive. An example of such an application is web caching, where keeping old information is not very useful. Even though these guidelines are general, we emphasize that the tradeoff between false positives and false negatives is application specific. As a result, there is no optimal solution for all applications and one should first define the tolerated false-positive and false-negative rates before selecting the best CBF variant.

5 Application: IP Traceback

Motivated by common denial-of-service (DoS) attacks with spoofed source IP addresses, IP traceback has the goal of tracing network packets back to their true source [13]. After locating the attack origin, the attack traffic can then be filtered and legal actions can be taken. We consider a system composed of a packet marking and a path reconstruction procedures. In the packet marking procedure, each router marks the forwarded packets to notify its presence on the attack path [14]. Upon receiving a packet, the victim then has enough information to reconstruct the attack path.

Due to processing (e.g., fragmentation) and IP header space limitations, packets cannot store the IP address of each traversed router [15]. Instead, we propose to have a Bloom filter in each packet that stores the traversed routers. The standard Bloom filter cannot be used because it can be easily saturated by the attacker. We propose to use the *concatenated Bloom filter* for this purpose. Before forwarding a packet, each router first uses the time-to-live (TTL) field from the IP header (modulo d) to determine the subfilter where it should mark the packet. Then, the router inserts its IP address into the selected subfilter. Using the TTL field guarantees that insertions occur in a circular fashion across the path.

Upon receiving a packet it wishes to trace, the victim uses the TTL to determine the subfilter where the last marking occurred. It then checks for the presence of each neighbor router in this subfilter. If a router is recognized, both the CBF and the TTL counter are sent to this router over a control channel to continue the path reconstruction. The same procedure is repeated by each recognized router, until there is a router that does not recognize any neighbor in the CBF. At this point, the path reconstruction ends and the results are sent from this last router back to the victim.

To evaluate the performance of this traceback system, we implement the CBF as well as the packet marking and the path reconstruction procedures in a DoS attack simulator. To create the network topology, we used the topology generator *nem* [16], which is based on sampling the Internet map. The *nem* generator extracts a random subgraph of a network map, keeping its original properties, such as node degree, average distance, and network diameter. Our topology consists of 10,000 routers with an average of 3.15 neighbors per router and is sampled from a real Internet topology. The attacker and the victim are randomly chosen, and a path is set between the two using the shortest path. A DoS attack is simulated by marking the packets according to the selected path. After packet marking, the path reconstruction starts from the victim toward the attacker.

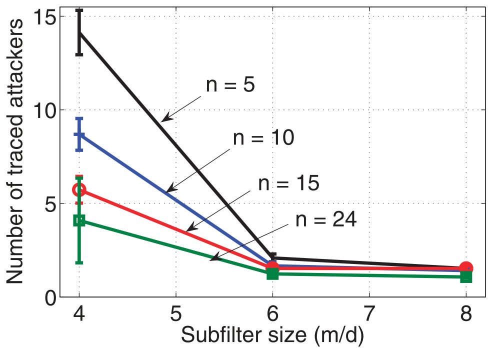


Fig. 7. The number of traced attackers as a function of the subfilter size (m/d).

We use CBF-3 in our simulation and initially assume to have only a single element per subfilter (i.e., $n \leq d$); thus, we have no false negatives. The proposed traceback system then always succeeds in tracing the attacker. However, false positives may affect the result. A false positive during the reconstruction means that routers not traversed by the packet may actually be included in the reconstructed graph. Thus, to measure the accuracy of the proposed system, we measured the number of traced attackers in each simulation round. In each round, a new pair of attacker and victim is chosen and the topology also changes. Ideally, it is expected that the number of traced attackers is 1, i.e., only the real attacker is traced. A large number of traced attackers then indicate a low accuracy. Fig. 7 shows the number of traced attackers as a function of the subfilter size (m/d), for different attack path lengths (n). The results are obtained with a confidence interval of 95 percent, represented by vertical error bars in the graph.

Fig. 7 shows the tradeoff between the IP header overhead, measured as the subfilter size (m/d), and system accuracy, for different path lengths (n). Since we assume $n \leq d$, the system always finds the real attacker. However, the accuracy decreases with smaller subfilters due to false positives. According to Internet statistics [17], path lengths have a mean of 15.3 hops and a standard deviation of 4.2. Assuming a Gaussian distribution, a path length is shorter than 25 hops with probability 98 percent. From the curve $n = 24$ in the figure, the proposed system can then trace 98 percent of Internet paths with an accuracy of 2.1, using only 6 bits per router. This represents a header space savings of 81.3 percent, compared to a list of 32-bit IPv4 addresses. This accuracy can be further improved with a higher overhead.

For $n > d$, we can choose among the different CBF variants. The CBF-2 and CBF-3 guarantee the discovery of the last d hops in the attack path with a low false-positive rate. The CBF-1, on the other hand, allows the discovery of even more distant hops. For instance, using CBF-1 with the same subfilter size $m/d = 6$ and a maximum false-positive rate of 10 percent, we remember the last d routers with probability 100 percent and the previous d routers with probability 33 percent, which helps to trace the source of the attack even further. The collected path information can then be used to filter the attack traffic closer to the source.

6 Related Work

There are several Bloom filter variants proposed for different applications [18], [19]. In this section, we present a few of these proposals more closely related to the CBF.

Fan et al. [4] use Bloom filters to compactly represent the cache content of web proxies. To allow elements to be deleted, the authors propose to replace each bit of the filter by a counter. The *counting Bloom filter* allows to delete an element by decrementing the corresponding counters. The authors show that using as few as 4 bits per counter is sufficient to maintain a negligible false-negative probability. Donnet et al. [9] also propose algorithms to delete information from the Bloom filter. The main idea consists of resetting the least touched bits, considering that this information is available.

Guo et al. [10] propose the *dynamic Bloom filter*, which is a set of counting Bloom filters that are dynamically allocated and managed according to the growth of the number of inserted elements. The goal is to avoid wasting space when the number of elements is low and avoid saturating the filter when the number of elements grows.

Hao et al. [11] limit the false-positive rate by using *restricted fill Bloom filters* (RFBF), which basically limits the fraction of bits in 1. The authors propose the *incremental Bloom filter* which is composed of multiple RFBFs. Every time an RFBF reaches its maximum number of elements, a new filter is allocated. The number of RFBFs needed and their size are defined according to the distribution of the number of elements n and the maximum false-positive rate.

Deng and Rafiei [12] consider the problem of representing streaming data, which has the particularity that recent elements are more important than older ones. They propose the *stable Bloom filter*, which works as a buffer that probabilistically deletes some old elements during each element insertion. Before inserting a new element, a few random bits are reset. This construction achieves a stable state with the increase of the number of insertions, avoiding the saturation of the filter.

More closely related to this work, Laufer et al. [8] propose the *generalized Bloom filter*, whose novelty is the introduction of hash functions that reset bits. This approach makes the filter less dependent on its initial state, since the false-positive probability becomes limited, regardless of the initial state.

Although space efficient, some of these proposals are not designed for distributed applications and thus are vulnerable to state-adjusting attacks [4], [9], [10], [11]. In this case, an attacker can initialize the filter to its worst condition to either saturate the filter [4], [9] or avoid future insertions[10], [11]. The *stable* and *generalized* variants protect the represented information, since the filter is independent from the initial state. However, both filters suffer from a low storage capacity. The CBF allows us to increase the filter capacity, while still keeping the filter robust to its initial condition.

7 Conclusions

In this paper, we presented the *concatenated Bloom filter*, a novel data structure designed to improve the storage capacity of the filter while still maintaining a high robustness (i.e., bounded false positives). The key idea is to concatenate several small subfilters, each one representing only a few elements. We derive the analytical expressions for the false-positive and false-negative rates of the filter, and introduce the concept of *filter capacity*. We show that this structure allows us to significantly reduce false negatives and, thus, increase the filter capacity without increasing false positives. For a fixed maximum false-positive rate, we show that the CBF has a higher capacity when compared to the *generalized Bloom filter*. As a result, due to its higher space efficiency for the same robustness level, the CBF is better suited for set representation in distributed applications.

We also evaluate the performance of the CBF in an IP traceback application. We show that the system is able to find the attacker using only 6 bits for each router in the path from the attacker to the victim.

Acknowledgments

This work was supported by CNPq, CAPES, FINEP, FUNTTEL, and FAPERJ.

References

[1] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Supporting Scalable and Adaptive Metadata Management in Ultralarge-Scale File Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 22, no. 4, pp. 580-593, Apr. 2011.

[2] A.C. Viana, M.D. Amorim, Y. Viniotis, S. Fdida, and J.F. de Rezende, "Twins: A Dual Addressing Space Representation for Self-Organizing Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 17, no. 12, pp. 1468-1481, Dec. 2006.

[3] I.M. Moraes and O.C.M.B. Duarte, "A Lifetime-Based Peer Selection Mechanism for Peer-to-Peer Video-on-Demand Systems," *Proc. IEEE Int'l Conf. Comm. (ICC '10),* May 2010.

[4] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking,* vol. 8, no. 3, pp. 281-293, June 2000.

[5] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Math.,* vol. 1, no. 4, pp. 485-509, 2003.

[6] S. Tarkoma, C.E. Rothenberg, and E. Lagerspetz, "Theory and Practice of Bloom Filters for Distributed Systems," *IEEE Comm. Surveys and Tutorials,* vol. 14, no. 1, pp. 131-155, first quarter 2012.

[7] K. Christensen, A. Roginsky, and M. Jimeno, "A New Analysis of the False Positive Rate of a Bloom Filter," *Information Processing Letters,* vol. 110, no. 21, pp. 944-949, 2010.

[8] 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, pp. 1804-1819, 2011.

[9] B. Donnet, B. Baynat, and T. Friedman, "Retouched Bloom Filters: Allowing Networked Applications to Trade off Selected False Positives against False Negative," *Proc. ACM CoNEXT Conf. (CoNEXT '06),* Dec. 2006.

[10] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The Dynamic Bloom Filters," *IEEE Trans. Knowledge and Data Eng.,* vol. 22, no. 1, pp. 120-133, Jan. 2010.

[11] F. Hao, M. Kodialam, and T.V. Lakshman, "Incremental Bloom Filters," *Proc. IEEE INFOCOM '08,* Apr. 2008.

[12] F. Deng and D. Rafiei, "Approximately Detecting Duplicates for Streaming Data Using Stable Bloom Filters," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '06),* 2006.

[13] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network Support for IP Traceback," *IEEE/ACM Trans. Networking,* vol. 9, no. 3, pp. 226-237, June 2001.

[14] Y. Xiang, W. Zhou, and M. Guo, "Flexible Deterministic Packet Marking: An IP Traceback System to Find the Real Source of Attacks," *IEEE Trans. Parallel and Distributed Systems,* vol. 20, no. 4, pp. 567-580, Apr. 2009.

[15] R.P. Laufer, P.B. Velloso, D. de O. Cunha, I.M. Moraes, M.D.D. Bicudo, M.D.D. Moreira, and O.C.M.B. Duarte, "Towards Stateless Single-Packet IP Traceback," *Proc. 32nd IEEE Conf. Local Computer Networks (LCN '07),* 2007.

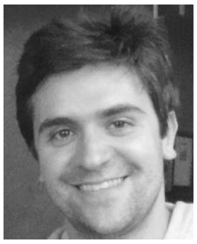
[16] D. Magoni and J.-J. Pansiot, "Internet Topology Modeler Based on Map Sampling," *Proc. IEEE Seventh Int'l Symp. Computers and Comm. (ISCC '02),* p. 1021, July 2002.

[17] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, K.C. Claffy, and A. Vahdat, "The Internet AS-Level Topology: Three Data Sources and One Definitive Metric," *ACM SIGCOMM Computer Comm. Rev.,* vol. 36, no. 1, pp. 17-26, 2006.

[18] B. Xiao and Y. Hua, "Using Parallel Bloom Filters for Multi-attribute Representation on Network Services," *IEEE Trans. Parallel and Distributed Systems,* vol. 21, no. 1, pp. 20-32, Jan. 2010.

[19] Y. Qiao, T. Li, and S. Chen, "One Memory Access Bloom Filters and Their Generalization," *Proc. IEEE INFOCOM '11,* Apr. 2011.

Marcelo Duffles Donato Moreira is an electronics and computer engineer from Federal University of Rio de Janeiro (UFRJ) with the MSc degree in computer networks from COPPE/UFRJ. Currently, he is a research engineer at INTELIE Research Lab, Brazil. His major research interests are in distributed systems, computer networks, and information security.



Rafael Pinaud Laufer received the BSc and MSc degrees in electrical and computer engineering from Universidade Federal do Rio de Janeiro (UFRJ), Brazil, in 2003 and 2005, respectively. He is currently working toward the PhD degree in computer science from the University of California, Los Angeles (UCLA). During his graduate studies, he worked at Bell Labs, Cisco, Technicolor, and EPFL. He received the Marconi Society's Young Scholar Award in 2008. He is a member of the IEEE.

Pedro Braconnot Velloso received the BSc and MSc degrees in electrical engineering from the Universidade Federal do Rio de Janeiro, Brazil, in 2001 and 2003, respectively. He received the PhD degree from the Université Pierre et Marie Curie (Paris 6) in 2008. He spent one year as a postdoc researcher at the Laboratoire d'Informatique de Paris 6 in 2008/2009. He worked for Bell Labs France as a research engineer in 2009/2011. Currently, he is an associate professor in the Department of Computer Science, Universidade Federal Fluminense (UFF), Brazil. His interests are in distributed systems, wireless communications, and network security. He is a member of the IEEE.

Otto Carlos M.B. Duarte received the electronic engineer and MSc degrees from the Federal University of Rio de Janeiro (UFRJ), in 1976 and 1981, respectively, and the DrIng degree from ENST/Paris, Paris, France, in 1985. Since 1978, he has been a professor with UFRJ. From January 1992 to June 1993, he was with MASI Laboratory, University Paris 6, Paris. In 1995, he spent three with the International Computer Science Institute (ICSI), University of California, Berkeley. In 1999 and 2001, he was an invited professor with the University Paris 6. His major research interests are in multicast, QoS guarantees, security, and mobile communications.