Computer Networks 55 (2011) 1804–1819

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A Generalized Bloom Filter to Secure Distributed Network Applications ☆

Rafael P. Laufer [a,*], Pedro B. Velloso [b], Otto Carlos M.B. Duarte [c]

[a] University of California, Los Angeles, United States
[b] Bell Labs, Alcatel-Lucent, France
[c] COPPE/Poli, Universidade Federal do Rio de Janeiro, Brazil

ARTICLE INFO

Article history:
Received 27 September 2010
Accepted 23 December 2010
Available online 1 February 2011

Keywords:
Bloom filters
Data structures
Network security

ABSTRACT

Distributed applications use Bloom filters to transmit large sets in a compact form. However, attackers can easily disrupt these applications by using or advertising saturated filters. In this paper we introduce the Generalized Bloom Filter (GBF), a space-efficient data structure to securely represent a set in distributed applications, such as IP traceback, web caching, and peer-to-peer networks. Different from the standard Bloom filter, the GBF has an upper bound on the false-positive probability, limiting the effect of these attacks. The key idea of the GBF is to not only set, but also reset bits of the filter at each insertion. This procedure limits the false positives at the expense of introducing false negatives in membership queries. We derive expressions for the false-positive and false-negative rates and show that they are both upper-bounded in the GBF. We conduct simulations that validate the derived expressions and explore the tradeoffs of this data structure.

1. Introduction

The Bloom filter is a space-efficient data structure used to represent a set [1]. The filter is a bit array over which membership queries are conducted to distinguish non-members of the given set. Hash coding techniques are used to save space and allow efficient member lookup. The tradeoff cost associated with hashing is the introduction of false positives in the membership queries, which occurs when an external element is recognized as an authentic member of the set, even though it is not.

The Bloom filter works as follows. First, a bit array is allocated and initialized to all zeros. Each element of the given set is then digested through a number of independent and random hash functions. The hashing results are used to index into the bit array, and each indexed bit is set to one. After the insertions, membership queries can be conducted by digesting the element in question and checking whether the indicated bits are set. If at least one bit is zero, then the element does not belong to the given set for sure. Otherwise, the element is assumed to be an actual member of the set. There is a probability, however, that this assumption is wrong and that the element is, in fact, a false positive. Such an event occurs when all indicated bits of a non-member element are previously set by other legitimate elements. As more elements are inserted into the filter, the false-positive probability increases and the filter becomes more saturated (i.e., the fraction of bits set to one increases until we cannot distinguish which elements belong to the filter). Despite this drawback, Bloom filters are quite efficient if the probability of a false positive is kept low.

Bloom filters have been used in many different areas, including spell checkers, database applications, and networking [2,3]. Numerous proposed networking applications use Bloom filters to either keep a local compact log or to efficiently share information over the network [4]. We believe the standard Bloom filter fits well for local functions, such as logging, storage, and filtering [5–9].

☆ An earlier version of this paper appeared in the *Proceedings of the 12th International Conference on Telecommunications*, May 3–6, 2005, Capetown, South Africa. This work was supported by CNPq, CAPES, FAPERJ, FINEP, FUNTTEL, RNP, and UOL.

* Corresponding author.
E-mail address: rlaufer@cs.ucla.edu (R.P. Laufer).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.12.025

Our work, on the other hand, is motivated by applications that must transmit Bloom filters over the network, such as the following examples.

- In cache sharing, web proxy servers broadcast Bloom filters representing their cache contents to other neighbor proxies [5]. Proxy servers then check each other's Bloom filters before yielding a page fault. If the required page is in the cache of a neighbor server, the page is requested from this neighbor instead of from the actual web server.
- In peer-to-peer (P2P) applications and data-centric routing [10–12], each node keeps track of the objects reachable through its neighbors by using Bloom filters. For proper routing, each node periodically broadcasts a Bloom filter representing the objects it can reach.
- In IP traceback [13,14], the main goal is to identify the true source of anonymous denial-of-service (DoS) attacks. In this particular application, packets carry a built-in Bloom filter to store the IP addresses of traversed routers in a compact form. Using the filter of a received attack packet, the victim initiates membership tests with its neighbor routers to identify the one that forwarded the packet. A recursive procedure is performed on each identified router to trace the packet path back to its true source.

Security issues, however, restrict the deployment of standard Bloom filters in such distributed applications. For instance, an attacker could generate offending filters with all bits set to one, a simple technique we define as an *all-one attack*. An all-one filter makes it impossible to distinguish inserted elements from false positives during membership queries. The effect of this action in the above-mentioned applications is disruptive. If an attacker broadcasts a saturated filter in the web cache sharing and P2P applications, other nodes believe that the attacker has all the required web pages or objects. Requests are then forwarded to the attacking node, who can replace the respective web pages or objects at will. In the IP traceback application, the victim receives a saturated filter representing the traversed path of the packet. During the path traceback procedure, every router is recognized as a member of the filter and the path reconstruction is completely ineffective.

From a security viewpoint, the standard Bloom filter presents an inherent flaw. Depending on the number of bits set in the filter array, 100% false-positive rates are possible. An attacker can easily take advantage of this vulnerability to disrupt distributed applications in which the Bloom filter is transmitted over the network. Note that the problem is not related to the authentication, privacy, or integrity of the transmitted filter, but rather to the ability of the data structure to allow the mapping of the entire universe of elements into a few bits.

In this paper we propose a novel tamper-resistant data structure called Generalized Bloom Filter (GBF), which is not vulnerable to all-one attacks, because it has a limited false-positive rate and cannot be saturated. The basic idea of the GBF is to not only set, but also reset bits of the filter during each insertion. For this purpose it employs both hash functions that set and hash functions that reset bits. We show that the GBF has an upper bound on the false-positive probability regardless of the number of bits set in the filter. This bound is exclusively determined by the number of hash functions used; consequently, the action of an attacker in creating false positives is very limited. On the other hand, false negatives, which do not exist in standard Bloom filters, are now introduced with this generalization. A false negative occurs when an inserted element is not recognized by membership queries. The effect of false negatives, however, is also shown to be upper-bounded and considerably reduced with larger bit arrays.

The advantage of the GBF is that it restricts the number of elements we can insert into the filter. In fact, the GBF works as a probabilistic buffer, where older elements are more likely to be "forgotten" or overwritten by recent elements. This key idea results in interesting properties, such as a higher robustness to all-one attacks, as well as upper-bounded false positives and false negatives. Prior knowledge on the element group size is also not required, since only recent elements are kept in the GBF. We explore these tradeoffs throughout the paper and provide an in-depth understanding of all aspects of the GBF.

The remainder of the paper is organized as follows. A brief overview of Bloom filters is described in Section 2. The proposed generalization is then outlined in Section 3, along with a theoretical analysis. In Section 4 we detail our simulation environment for the Generalized Bloom Filter and compare analytical and simulation results; the simulation results corroborate our proposed analysis. Section 5 presents the tradeoffs between false positives and false negatives. In Section 6 we review the related work and discuss a few applications which could take advantage of the GBF. Finally, conclusions are presented in Section 7.

2. The Bloom filter

In this section an overview of the Bloom filter is described, following the analysis of Fan et al. [5], Margoliash [15], and Mitzenmacher [16].

The Bloom filter [1] is a space-efficient data structure used to represent a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements. It is constituted by an array of m bits and by k independent hash functions $h_1, h_2, \ldots, h_k$ whose outputs are uniformly distributed over the discrete range $\{0, 1, \ldots, m-1\}$. The average number of bits used to represent a single element is thus m/n. The filter is constructed as follows. First, all bits in the array are reset. Then, for each element $s_i \in S$, the bits corresponding to the positions $h_1(s_i), h_2(s_i), \ldots, h_k(s_i)$ are set. The same bit can be set several times without restrictions. Fig. 1 depicts how an element is inserted into a Bloom filter. After inserting the elements, membership queries can be easily conducted. To determine if an element x belongs to S, we check whether the bits of the array corresponding to the positions $h_1(x), h_2(x), \ldots, h_k(x)$ are 1. If at least one bit is 0, then $x \notin S$ for sure. Otherwise, $x \in S$ with high probability. Actually, an element $x \notin S$ may be recognized as an element of the set, creating a false positive. Such an anomaly occurs when the bits $h_1(x), h_2(x), \ldots, h_k(x)$ are all previously set by other previously inserted elements.

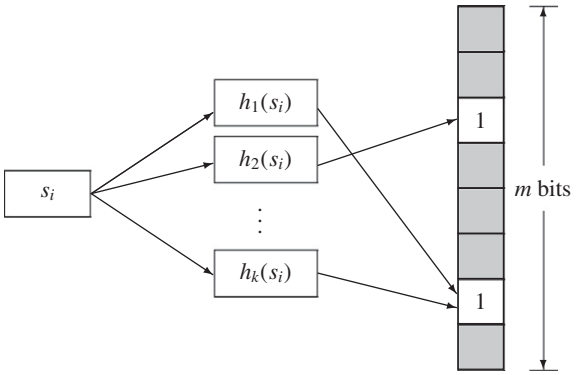


Fig. 1. Insertion of an element into a Bloom Filter. The s_i element is used as the input for the hash functions and the bits indexed by the $h_1, h_2, \ldots, h_k$ functions are set to 1.

The probability of a false positive for an element $x \notin S$ is calculated as follows. Given that perfectly independent and random hash functions are used, the probability p that a specific bit is still zero after inserting n elements is

$$p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}. \quad (1)$$

Since the same computation can be applied to every bit in the array, on average, the fraction of bits in 0 after all the n insertions is $p \approx e^{-kn/m}$. Hence, the fraction of bits in 1 after the n insertions is $(1 - p)$. The probability of a false positive f is the probability that we find a bit in 1 for each of the k indicated positions of a non-member; that is,

$$f = (1 - p)^k \approx (1 - e^{-kn/m})^k. \quad (2)$$

Two interesting comments can be made considering the false-positive probability f in Eq. (2). First, increasing the relation m/n always reduces the false-positive probability. By using more bits per element, the fraction $(1 - e^{-kn/m})$ of bits in 1 is reduced, and thus the false-positive probability decreases. Additionally, there is an important tradeoff between the number of hash functions k and the false-positive probability. Increasing k leads to a higher fraction of bits in 1 and, accordingly, a higher false-positive rate. On the other hand, the probability of finding every indicated bit in 1 decreases with higher values of k. This tradeoff is depicted in Fig. 2, where an optimal value that minimizes f is clearly observed for each m/n ratio.

The optimal number of hash functions is found by differentiating f with respect to k; that is

$$\frac{\partial f}{\partial k} = \left[\ln(1 - e^{-kn/m}) + \frac{kn}{m}\frac{e^{-kn/m}}{1 - e^{-kn/m}}\right](1 - e^{-kn/m})^k. \quad (3)$$

Setting Eq. (3) to zero, noting that $(1 - e^{-kn/m})^k$ is never null, and cross-multiplying gives $(1 - p)^{(1-p)} = p^p$. Assuming k is finite and positive, the only solution is $p = 1/2$. Therefore, the minimum false-positive probability is achieved when approximately half of the bits is in 1 and the other half is in 0. In this case, $k = (m/n)\ln 2$ and the false-positive probability f is $(0.5)^k = (0.6185)^{m/n}$.

One problem with the Bloom filter is that its false-positive probability strongly depends on the number of bits in 1.

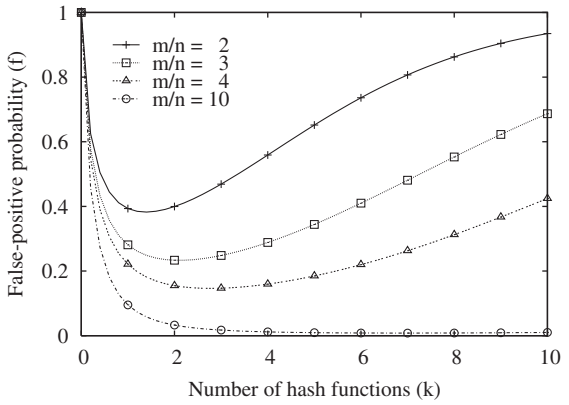


Fig. 2. False-positive probability of a Bloom filter as a function of the number of hash functions.

Fig. 3 shows the false-positive probability of a Bloom filter as a function of the fraction of bits in 1. We see that, for any chosen number of hash functions, the false-positive probability always increases as more bits are set. When all bits of the filter are set, the false-positive probability is clearly 100%. In this condition, every non-member tested against the filter is accepted as a genuine member of the set. The situation is even worse because the number of bits in 1 increases as more elements are inserted into the filter. In distributed applications that transmit the Bloom filter over the network [5,10–13], these inherent properties are harmful and can be easily exploited using an all-one attack. Therefore, in the next section we propose a generalization to Bloom filters that has an upper-bounded false-positive rate independent of the number of bits in 1. In order to distinguish the Bloom filter from our generalized version, the Bloom filter is hereafter referred to as standard Bloom filter or standard filter.

3. The Generalized Bloom Filter (GBF)

Like the standard filter, the Generalized Bloom Filter is also a data structure used to represent a set $S = \{s_1, s_2, \ldots, s_n\}$

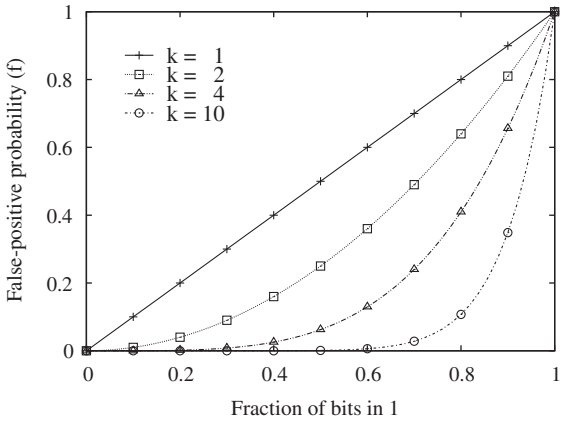


Fig. 3. False-positive probability of a Bloom filter as a function of the fraction of bits in 1.

of n elements in a compact form. It is constituted by an array of m bits and by $k_0 + k_1$ independent hash functions $g_1, g_2, \ldots, g_{k_0}$, $h_1, h_2, \ldots, h_{k_1}$ whose outputs are uniformly distributed over the discrete range $\{0, 1, \ldots, m-1\}$. The GBF is built in a similar way to the standard filter. Nevertheless, the initial value of the bits of the array is not restricted to 0 anymore. In the GBF, these bits can be initialized to any value. For each element $s_i \in S$, the bits corresponding to the positions $g_1(s_i), g_2(s_i), \ldots, g_{k_0}(s_i)$ are reset, and the bits corresponding to the positions $h_1(s_i), h_2(s_i), \ldots, h_{k_1}(s_i)$ are set. In the case of a collision between a function g_i and a function h_j for the same element, we arbitrate that the resulting bit in the filter is always reset. The same bit can be set to 0 or 1 several times without restrictions. Fig. 4 depicts how an element is inserted into a GBF. After inserting the elements, membership queries can be easily conducted. To determine if an element x belongs to S, we check whether the bits of the array corresponding to the positions $g_1(x), g_2(x), \ldots, g_{k_0}(x)$ are 0, and if the bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ are 1. If at least one bit is inverted, then $x \notin S$ with high probability. In the GBF, it is possible that an element $x \in S$ may not be recognized as an element of the set, creating a false negative. Such an anomaly happens when at least one of the bits $g_1(x), g_2(x), \ldots, g_{k_0}(x)$ is set, or one of the bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ is reset by another element inserted afterwards. On the other hand, if no bit is inverted, then $x \in S$ also with high probability. In fact, an element $x \notin S$ may be recognized as an element of the set, creating a false positive. A false positive occurs when the bits $g_1(x), g_2(x), \ldots, g_{k_0}(x)$ are all in 0, and the bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ are all in 1 due to other inserted elements or due to the initial condition of the bit array.

3.1. False positives

To calculate the false-positive probability of the Generalized Bloom Filter (GBF), we first calculate the probability of a bit being set or reset by each element insertion. Given that in a collision the functions g_i always take precedence over the functions h_j, the probability q_0 that a specific bit is reset by an element insertion is the probability that at least one of the k_0 hash functions reset the bit. Then, q_0 is expressed by

$$q_0 = \left[1 - \left(1 - \frac{1}{m}\right)^{k_0}\right] \approx (1 - e^{-k_0/m}). \tag{4}$$

The probability q_1 that a specific bit is set by an element 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. Thus, q_1 is given by

$$q_1 = \left[1 - \left(1 - \frac{1}{m}\right)^{k_1}\right]\left(1 - \frac{1}{m}\right)^{k_0} \approx (1 - e^{-k_1/m})e^{-k_0/m}. \tag{5}$$

Finally, the probability that a specific bit is neither set nor reset during an insertion is given by

$$(1 - q_0 - q_1) = \left(1 - \frac{1}{m}\right)^{k_0+k_1} \approx e^{-(k_0+k_1)/m}. \tag{6}$$

Since the same calculation applies for every bit in the array, on average, a fraction of q_0 bits is reset, a fraction of q_1 bits is set, and a fraction of $(1 - q_0 - q_1)$ bits is neither set nor reset (i.e., remains untouched) at each element insertion. For an m-bit array, we then have on average $b_0 = m \cdot q_0$ bits reset, $b_1 = m \cdot q_1$ bits set, and $(m - b_0 - b_1)$ bits neither set nor reset by each insertion.

From these probabilities, the distribution of bits over the bit array is determined. The probability p that a specific bit is 0 after n insertions is calculated from the probabilities of $n + 1$ mutually exclusive events. The first event is when the bit is initially 0 and it is not touched by the n inserted elements. If p_0 represents the probability that a specific bit is initially 0, the probability of such an event is $p_0(1 - q_0 - q_1)^n$. The next n events are those where the bit is reset by the $(n - i)$th element, and it is not touched by the following i elements, for $0 \leqslant i \leqslant n - 1$. The probability of each one of these events is $q_0(1 - q_0 - q_1)^i$. Accordingly, we have

$$\begin{aligned} p &= p_0(1 - q_0 - q_1)^n + \sum_{i=0}^{n-1} q_0(1 - q_0 - q_1)^i \\ &= p_0(1 - q_0 - q_1)^n + q_0\left[\frac{1 - (1 - q_0 - q_1)^n}{1 - (1 - q_0 - q_1)}\right] \\ &= p_0(1 - q_0 - q_1)^n + \frac{q_0}{q_0 + q_1}[1 - (1 - q_0 - q_1)^n]. \end{aligned} \tag{7}$$

Since the same computation applies to every bit in the array, on average, a fraction of p bits is in 0, and a fraction of $(1 - p)$ bits is in 1 after n insertions. If we increase the number of inserted elements (i.e., $n \to \infty$), we can observe in Eq. (7) that the fraction p of bits in 0 converges to $q_0/(q_0 + q_1)$ and does not change with future insertions; that is, the filter reaches a *steady state*. This is the key property that keeps the filter from saturating and bounds the false-positive probability. We talk more about this in future sections.

From the bit distribution, the probability of a false positive is calculated as follows. Since on average $b_0 = m \cdot q_0$ bits are reset and $b_1 = m \cdot q_1$ bits are set by each element insertion, a false positive occurs if the b_0 selected bits are

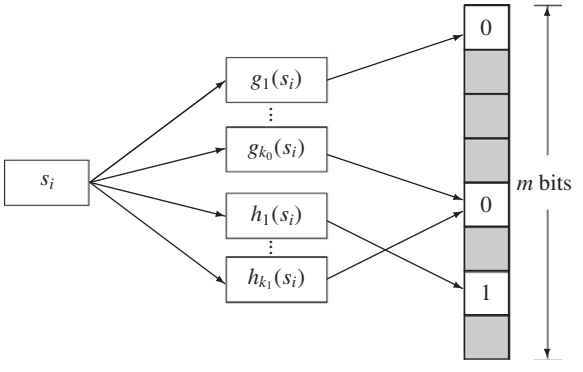


Fig. 4. Insertion of an element into a Generalized Bloom Filter. The s_i element is used as the input for the hash functions. The bits indexed by the $g_1, g_2, \ldots, g_{k_0}$ functions are reset and the bits indexed by the $h_1, h_2, \ldots, h_{k_1}$ functions are set. The g_i functions have preference over the h_j functions.

0 and the b_1 selected bits are 1. Hence, the probability of a false positive f_p for the GBF is calculated as

$$f_p = p^{b_0}(1-p)^{b_1}. \tag{8}$$

3.2. False negatives

False positives can happen only when *non-inserted elements* are tested against the filter, and the probability of a false positive is the same for every checked element. On the other hand, false negatives only occur for *inserted elements*, and each element has a different probability of being a false negative. An element is said to be a false negative if at least one of its marked bits is inverted (i.e., a bit in 1 is reset or a bit in 0 is set) by a succeeding element, and that bit remains inverted until the end of the insertions.[1] As a consequence, the false-negative probability depends on the insertion order and elements inserted first have a higher chance of having their bits inverted by subsequent elements.

The false-negative probability is calculated from the probability that a specific bit of the $(n-i)$th element is not inverted by the subsequent i inserted elements, for $0 \leqslant i \leqslant n-1$. The probability $p_{00}(n-i)$ that a bit reset by the $(n-i)$th element remains in 0 by the end of the following i insertions is calculated from the probabilities of $i+1$ mutually exclusive events. The first event is when the bit is neither set nor reset by all of the subsequent i insertions; it happens with probability $(1-q_0-q_1)^i$. The other i events are those where the bit is reset by the $(n-j)$th element, and it is not touched anymore by the following j insertions, for $0 \leqslant j \leqslant i-1$. Therefore, $p_{00}(n-i)$ is

$$\begin{aligned} p_{00}(n-i) &= (1-q_0-q_1)^i + \sum_{j=0}^{i-1} q_0(1-q_0-q_1)^j \\ &= (1-q_0-q_1)^i + q_0\left[\frac{1-(1-q_0-q_1)^i}{1-(1-q_0-q_1)}\right] \\ &= (1-q_0-q_1)^i + \frac{q_0}{q_0+q_1}\left[1-(1-q_0-q_1)^i\right]. \end{aligned} \tag{9}$$

Similarly, the probability $p_{11}(n-i)$ that a bit set by the $(n-i)$th element remains in 1 by the end of the following i insertions is given by

$$\begin{aligned} p_{11}(n-i) &= (1-q_0-q_1)^i + \sum_{j=0}^{i-1} q_1(1-q_0-q_1)^j \\ &= (1-q_0-q_1)^i + q_1\left[\frac{1-(1-q_0-q_1)^i}{1-(1-q_0-q_1)}\right] \\ &= (1-q_0-q_1)^i + \frac{q_1}{q_0+q_1}[1-(1-q_0-q_1)^i]. \end{aligned} \tag{10}$$

When increasing the number of insertions after the element (i.e., $i \to \infty$), we observe from Eqs. (9) and (10) that there is always a probability that a bit set to either 0 or 1 remains the same, even after a high number of insertions. This is an interesting property that also allows us to bound the false negatives.

From Eqs. (9) and (10), the false-negative probability of the inserted elements is determined. The false-negative probability $f_n(n-i)$ of the $(n-i)$th element is calculated by taking the complement of the probability that none of its bits are inverted. Thus, since on average $b_0 = m \cdot q_0$ bits are reset and $b_1 = m \cdot q_1$ bits are set by each element insertion, this probability is

$$f_n(n-i) = 1 - p_{00}(n-i)^{b_0} p_{11}(n-i)^{b_1}. \tag{11}$$

The average false-negative probability f_n is then defined as

$$f_n = \frac{1}{n}\sum_{i=0}^{n-1}[1 - p_{00}(n-i)^{b_0} p_{11}(n-i)^{b_1}]. \tag{12}$$

3.3. The Bloom filter as a particular case

As expected, the Bloom filter is a particular case of the Generalized Bloom Filter (GBF). For instance, the false-positive probability f_p in Eq. (8) is reduced to the probability of the standard filter when its parameters are used, that is, the number of hash functions to set is $k_1 = k$, the number of hash functions to reset is $k_0 = 0$, and the probability that a bit is initially reset is $p_0 = 1$. In this case, the fraction of 0 bits is $p = e^{-kn/m}$, the average number of bits reset per insertion is $b_0 = 0$, and the average number of bits set per insertion is $b_1 = m(1-e^{-k/m})$. Noticing that $m \gg k$ and using an expansion expression for b_1, we get to the simplification usually made for the standard filter

$$\begin{aligned} b_1 &= m(1-e^{-k/m}) \\ &= \left[1-\left(1-\frac{k}{m}+\frac{k^2}{2m^2}-\frac{k^3}{6m^3}+\cdots\right)\right] \approx k. \end{aligned} \tag{13}$$

Therefore, the probability of a false positive f_p is reduced to the probability of the standard Bloom filter in Eq. (2).

Likewise, the false-negative probability in Eq. (11) is zero for the standard Bloom filter. In this case, $k_0 = 0$ and thus $b_0 = 0$ and $p_{11}(n-i) = 1$, for $0 \leqslant i \leqslant n-1$. Therefore, the probability of a false negative is always zero. Additionally, for the last inserted element, the nth element, the false-negative probability is always zero since no other element can invert any of its bits. In this case, we have $i = 0$ and $p_{00}(n) = p_{11}(n) = 1$; therefore, the probability of a false negative is also zero.

4. Evaluation

In order to analyze the behavior of the Generalized Bloom Filter (GBF), we implemented a simulator using C++. The simulator is based on a class called GBF that contains the methods for inserting and checking elements in a bit array. For each simulation round, we select new hash functions, set the bit array to the desired initial condition, and insert the elements into the filter. After the insertions, membership queries of external elements and inserted elements are performed to respectively measure the false-positive

[1] Another definition is to consider an element a false negative only if more than b bits are inverted. Our definition is more conservative, and we only consider $b = 0$ in this work.

and false-negative rates. To show the advantages of the GBF over the standard filter, we also present an analytical comparison between the two versions of the filter in this section. The analysis includes three different metrics: false positives, false negatives, and interference of the initial condition.

For the independent and random hash functions assumed in Sections 2 and 3, our simulations used a universal class of hash functions [17], defined as follows. Let the elements of the set be integers from the universe $U = \{1,2,\ldots,z-1\}$, and let the output range of the hash functions be defined as $\{0,1,\ldots,m-1\}$. The class H of hash transformations $h_{c,d}$ which map an element $x \in U$ into the respective range is defined as

$$H = \{h_{c,d}(\cdot) | 0 < c < z, 0 \leqslant d < z\}, \tag{14}$$

where

$$h_{c,d}(x) = [(cx + d) \bmod z] \bmod m. \tag{15}$$

The numbers c and d are integers within the interval defined by Eq. (14). For each hash function, c and d can be arbitrarily chosen. The integer z is defined as a large prime.

For each point in the following graphs, we ran 1000 simulation rounds. For each round, we define new hash functions $h_{c,d}(\cdot)$ by selecting different parameters c and d. The integer z is defined as the prime 2,100,000,011, so we can test elements from the universe $U = \{1,2,\ldots,2100000010\}$. We split this element universe into two halves. From the first half, we randomly select n elements and insert them into the GBF, where n is defined according to the experiment. After the insertions, we randomly select 10,000 elements from the second half and check whether the GBF recognizes these elements as false positives. To estimate the false negatives, we store the inserted elements in memory and check whether the GBF recognizes them after all insertions. Each inserted element which is not recognized by the GBF is reported as a false negative.

Simulation results are very close to the analytical results, which validates the analytical expressions derived in the previous section. In our simulations, for a 95% confidence level, the largest confidence interval obtained was 0.003, which show that, besides matching the analytical results, the simulation results also have a small variance. Hence, the confidence intervals are not presented in the following graphs. In this section, all graphs present simulation results as discrete points and analytical results as continuous curves.

4.1. Upper-bounding the false-positive probability

In membership queries, a false positive implies recognizing an element that does not belong to S as a legitimate element. As the false-positive probability increases, more and more non-members are incorrectly identified as belonging to the set. Therefore, a low false-positive probability is desired.

First, we note that Eq. (8) can be simplified if we assume that $m \gg k_0$ and $m \gg k_1$. This assumption is reasonable since usually the size of the filter is much larger than the number of hash functions used. In this case, using the expansion in Eq. (13), we can rewrite $b_0 \approx k_0$ and $b_1 \approx k_1$. Accordingly, the simplified probability of a false positive using these assumptions is

$$f_p \approx p^{k_0}(1-p)^{k_1}. \tag{16}$$

Fig. 5 shows the false-positive probability of a GBF f_p as a function of $(1-p)$, according to Eq. (16) and our simulation results. The probability $(1-p)$ can also be seen as the fraction of bits in 1 after inserting n elements. In Fig. 5(a), we see that the false-positive probability of the standard Bloom filter increases with $(1-p)$, which is in accordance with Eq. (2). A clear tradeoff between the distribution of bits and the false-positive probability is noticed in the curves representing the GBF. This tradeoff is explained by observing that, on average, k_0 bits reset and k_1 bits set are required to cause a false positive. If, however, $(1-p)$ is low, it is easy to find a bit in 0, but hard to find a bit in 1. On the other hand, if $(1-p)$ is high, it is easy to find a bit in 1 and difficult to find a bit in 0. In Fig. 5(b), we see a similar behavior. We see that when $k_1 = 0$, we have the analog of the standard Bloom filter, with only functions that reset bits. For this filter, the false-positive probability increases with the fraction of bits in 0. The other curves represent the analogs of the GBFs of Fig. 5(a).

Fig. 6 depicts the case where we have the same number of hash functions that set and reset bits. We see from this figure that increasing both k_0 and k_1 always decreases the false-positive probability. This behavior is expected since, when more hash functions are used, more bits need to be found either in 0 or in 1 to yield a false positive. In addition, we see that the maximum false-positive probability decreases exponentially with k. As we explain next, the maximum false-positive probability decays with rate $1/2^{k_0+k_1}$, for the case where $k_0 = k_1$. It is interesting to note that, while $1/2^k$ is the minimum false-positive rate for the standard Bloom filter, $1/2^{k_0+k_1}$ is the maximum false-positive rate for the GBF, when $k_0 = k_1$.

By differentiating Eq. (16) with respect to p, we get

$$\frac{\partial f_p}{\partial p} = k_0 p^{k_0-1}(1-p)^{k_1} - k_1 p^{k_0}(1-p)^{k_1-1}. \tag{17}$$

Assuming that p is neither 0 nor 1, it is easy to verify that the maximum false-positive probability of a GBF is reached when

$$p = \frac{k_0}{k_0 + k_1}. \tag{18}$$

The upper bound on the false-positive probability F_p is determined by substituting Eq. (18) back into Eq. (16), and it is given by the expression

$$F_p = \left(\frac{k_0}{k_0+k_1}\right)^{k_0}\left(\frac{k_1}{k_0+k_1}\right)^{k_1}. \tag{19}$$

As seen in Section 2, the false-positive probability of the standard Bloom filter always increases with the number of inserted elements. This behavior is expected since, as more elements are inserted, more bits of the filter are set. Eventually, the filter becomes saturated with ones, leading to a false-positive probability of 100%. We can confirm this in Eq. (19) if we let $k_0 = 0$. In this case, the maximum false-positive probability of the standard Bloom filter is 100%

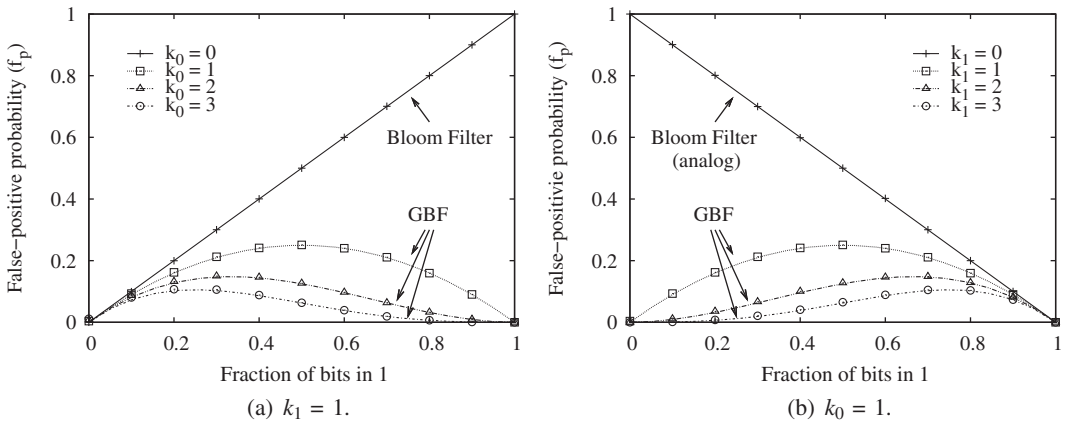


Fig. 5. False-positive probability of a GBF as a function of the fraction of bits set for (a) $k_1 = 1$ and (b) $k_0 = 1$.

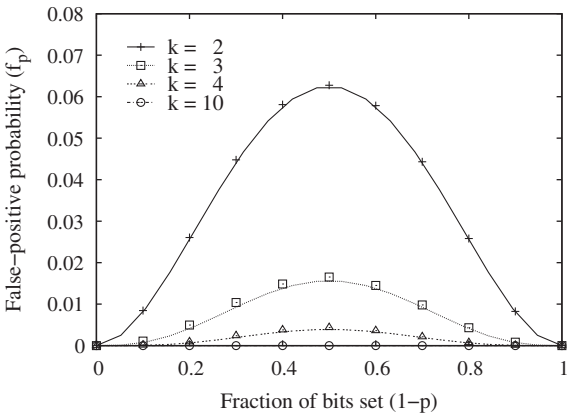


Fig. 6. False-positive probability of a GBF as a function of the fraction of bits set, for $k = k_0 = k_1$.

for any value of k_1. The key idea of the Generalized Bloom Filter (GBF) is to use both hash functions that set and hash functions that reset bits in the filter. As a consequence, saturation does not occur anymore, and the false-positive probability is now upper-bounded. Furthermore, this upper bound is exclusively determined by the number of hash functions employed, k_0 and k_1. As an example, when using as few hash functions as $k_0 = k_1 = 2$, we have an upper bound of only 6.3% on the false-positive probability. If we increase the number of hash functions from 2 to $k_0 = k_1 = 3$ and $k_0 = k_1 = 4$, this bound drops to 1.6% and 0.4%, respectively. This bound can be further reduced if a larger number of hash functions is used.

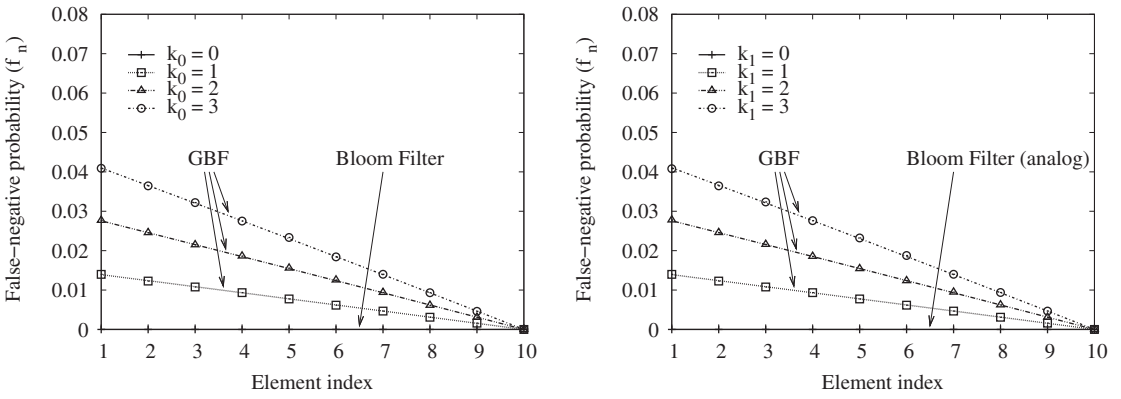
Two interesting properties of the GBF should be emphasized for clarity:

- The bound F_p in Eq. (19) only depends on the number of hash functions used, k_0 and k_1. It does not depend on the filter size m or the number of elements n to be inserted, which means that the GBF allows an unlimited number of insertions without compromising F_p. It can also be shown that F_p decreases as either k_0 or k_1 increases.
- The value of F_p in Eq. (19) is an upper bound on the false-positive probability of the GBF. Therefore, it should not be confused with the *average* false-positive probability f_p in Eq. (8), which by definition is always lower than or equal to F_p. On the other hand, the upper bound on the false-positive probability of the standard Bloom filter is always 100%, regardless of the filter parameters, and it cannot be reduced. Tuning the parameters of the standard Bloom filter only affects the average false-positive probability f_p.

4.2. Upper-bounding the false-negative probability

As shown in Section 4.1, when using hash functions that reset bits in the filter, an upper bound is imposed on the false-positive probability. The tradeoff cost is the introduction of false negatives in membership queries. A false negative implies not detecting an actually inserted element.

In Fig. 7(a) we show the false-negative probability for each element $(n - i)$, for $0 \leqslant i \leqslant n - 1$, according to Eq. (11) and our simulation results, using $k_1 = 1$, $n = 10$, and $m/n = 128$. Element 1 represents the first element inserted into the filter and element 10 is the last. We notice that the false-negative probability always equals zero for the standard Bloom filter, as expected. Since the original Bloom filter does not use hash functions to reset bits, it is impossible to have false negatives. For the GBF, however, we see that elements inserted first have higher false-negative probabilities. This behavior occurs because an element inserted earlier has a higher number of insertions ahead and, as a consequence, the probability of an inversion of its bit markings is higher. Another important observation is that the false-negative probability increases with k_0. It happens because the more functions we use, the higher the probability of a bit inversion by a subsequent element. The result is approximately the same if k_0 is fixed and k_1 increases instead, as shown in Fig. 7(b). The figures look very similar because the false-negative probability is approximately symmetric regarding k_0 and k_1, according to Eqs. (9)–(11). This result becomes more evident in the simplified versions of these equations — Eqs. (22)–(24) — explained later in this section.



(a) $k_1 = 1, n = 10, m/n = 128$. (b) $k_0 = 1, n = 10, m/n = 128$.

Fig. 7. False-negative probability of each element inserted into a GBF, for (a) $k_1 = 1$, $n = 10$, $m/n = 128$ and (b) $k_0 = 1$, $n = 10$, $m/n = 128$.

Fig. 8 depicts the behavior of the GBF when we dramatically increase the number of hash functions $k = k_0 = k_1$. Clearly, the false-negative probability of the first elements increases significantly. It happens because, with more hash functions, it is more likely that the bit markings of the first elements will be inverted by the last elements. From this and the previous figures, we can think of the GBF as a memory buffer where the first inserted elements are more likely to be "forgotten" or overwritten by the last ones. This effect depends not only on the number of hash functions used, k_0 and k_1, but also on the relative size of the filter m/n. For instance, a larger filter implies a larger output range for the hash functions, and hence the probability of one element overwriting the bits of previous elements decreases.

From Figs. 6 and 8, we see that the number of hash functions used in the GBF has a dual effect. On one hand, increasing the number of hash functions reduces the false-positive probability, as shown in Section 4.1. On the other hand, it increases the false-negative probability. We return to this dual effect in Section 5. In the following paragraphs, we derive an upper bound on the false-negative probability that depends only on k_0, k_1, and m/n.

According to Fig. 7, the false-negative probability of an element is lower if we have less elements inserted after it. In fact, this behavior is always true, and it is analytically proven if we observe Eq. (9) closely, as in

$$p_{00}(n-i) = (1-q_0-q_1)^i + \frac{q_0}{q_0+q_1}[1-(1-q_0-q_1)^i] = \frac{q_0}{q_0+q_1} + \frac{q_1}{q_0+q_1}(1-q_0-q_1)^i. \quad (20)$$

For two elements x and y with x inserted before y, we have $p_{00}(x) \leqslant p_{00}(y)$. A similar result could be derived to show that $p_{11}(x) \leqslant p_{11}(y)$. From Eq. (11), it is easy to check then that $f_n(x) \geqslant f_n(y)$ is always true. The $f_n(x) = f_n(y)$ equality only holds for the standard Bloom filter, when the false-negative probability is zero, regardless of the insertion order.

We now derive an upper bound on the false-negative probability of a GBF. Let $f_n(0)$ be the false-negative probability of an hypothetical element inserted prior to the n elements of a given set. Following our previous result, the inequality $f_n(0) \geqslant f_n(1) \geqslant f_n(2) \geqslant \cdots \geqslant f_n(n)$ always holds. Accordingly, we define the upper bound as $f_n(0)$. An advantage in doing so is that we represent the upper bound on the false-negative probability as a function of only the number of hash functions, k_0 and k_1, and the relative size of the filter, m/n, as we show now.

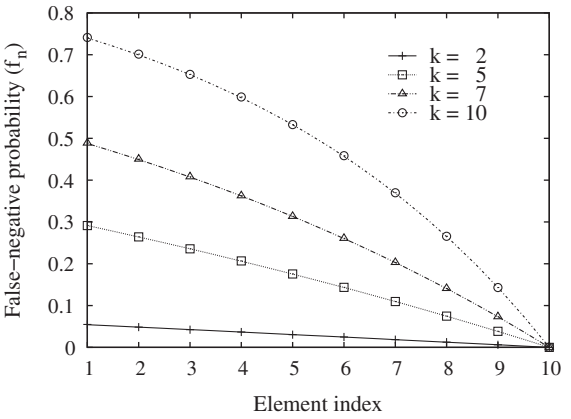


Fig. 8. The effect of increasing the number of hash functions on the false-negative probability of a GBF for each inserted element, for $k = k_0 = k_1$, $n = 10$, and $m/n = 128$.

Assuming $m \gg k_0$ and $m \gg k_1$, we first observe that the following approximation is reasonable

$$\frac{q_0}{q_0+q_1} \approx \frac{1-e^{-k_0/m}}{1-e^{-(k_0+k_1)/m}} = \frac{1-\left(1-\frac{k_0}{m}+\cdots\right)}{1-\left(1-\frac{k_0+k_1}{m}+\cdots\right)} \approx \frac{k_0}{k_0+k_1}. \quad (21)$$

As a result, we rewrite Eqs. (9) and (10) for the hypothetical element 0 as

$$p_{00}(0) \approx (1-q_0-q_1)^n + \frac{k_0}{k_0+k_1}[1-(1-q_0-q_1)^n] \approx e^{-(k_0+k_1)n/m} + \frac{k_0}{k_0+k_1}(1-e^{-(k_0+k_1)n/m}), \quad (22)$$

$$p_{11}(0) \approx (1-q_0-q_1)^n + \frac{k_1}{k_0+k_1}[1-(1-q_0-q_1)^n]$$
$$\approx e^{-(k_0+k_1)n/m} + \frac{k_1}{k_0+k_1}(1-e^{-(k_0+k_1)n/m}). \quad (23)$$

Substituting Eq. (22) and (23) back into Eq. (11), and observing that $b_0 \approx k_0$ and $b_1 \approx k_1$, the upper bound on the false-negative probability F_n is

$$F_n = f_n(0) \approx 1 - p_{00}(0)^{k_0} p_{11}(0)^{k_1}. \quad (24)$$

As in the false-positive case, the upper bound F_n should not be confused with the element false-negative probability $f_n(n-i)$ in Eq. (11) or the average false-negative probability f_n in Eq. (12), which by definition are always lower than F_n.

According to Eqs. (22)–(24), we can see that F_n does not depend on the initial condition p_0, and it is uniquely defined by k_0, k_1, and the m/n ratio. Therefore, the system designer may first arbitrate the upper bound on the GBF false-positive probability F_p by defining both k_0 and k_1. The desired upper bound on the false-negative probability F_n is then achieved by adjusting the m/n ratio. Lower false-negative probabilities, however, are achieved at the cost of using more bits per element.

Fig. 9 depicts the upper bound on the GBF false-negative probability as a function of the m/n ratio, according to Eq. (22)–(24) and simulation results. From the figure we see that increasing m/n leads to a lower false-negative probability. It occurs because by increasing the number of bits per element we increase the output range of the hash functions, thereby decreasing the probability of a bit overwriting. A very similar result is presented when k_0 is fixed and k_1 changes (not shown).

According to the results mentioned so far, the GBF represents a set with limited false positives and limited false negatives, regardless of the state of the bit array. For instance, when using $k_0 = k_1 = 2$ and $m/n = 128$ bits per element, the upper bound on the false-positive probability F_p is 6.3% and the upper bound on the false-negative probability F_n is 6.0%. The average false-negative f_n, however, is only 3.0%. The average false-positive rate f_p depends on the initial condition of the filter; for instance, assuming that $p_0 = 0\%$ and $p_0 = 25\%$, we have $f_p = 0.02\%$ and $f_p = 3.7\%$, respectively. If lower rates are desired, we could use $k_0 = 2$, $k_1 = 3$, and $m/n = 256$ bits per element. In this case, we have $F_p = 3.5\%$ and $F_n = 4.6\%$. The average false-negative rate f_n is only 2.3%, and the average false-positive rate f_p is only 0.01% and 2.7%, assuming $p_0 = 0\%$ and $p_0 = 25\%$, respectively. The value of these parameters can be further increased as long as we are willing to reduce the false-positive and false-negative probabilities. In Section 5, we present a few tables showing these tradeoffs for different parameters.

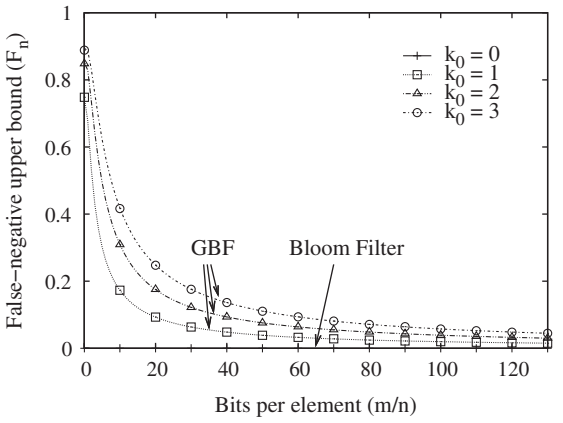


Fig. 9. Upper bound on the false-negative probability of a GBF as a function of the number of bits per element m/n, for $k_1 = 1$.

4.3. Interference of the initial condition

The false-positive probability is the only one that is affected by the initial condition. However, in this section we show that its interference is limited, and that the maximum false-positive rate one can achieve setting the initial condition in a GBF is the upper bound F_p derived in Eq. (19). False negatives are not affected by the initial condition of the filter.

Fig. 10 shows how the false-positive probability of a GBF is affected by the initial fraction of bits set, $(1-p_0)$, according to Eq. (16) and simulation results. We observe in Fig. 10(a) that the initial condition of the filter may be adjusted to maximize the false-positive probability. By fixing k_0 and changing k_1, an analogous result is achieved, as seen in Fig. 10(b). By differentiating Eq. (16) with respect to p_0, we have

$$\frac{\partial f_p}{\partial p_0} = k_0 p^{k_0-1}\frac{\partial p}{\partial p_0}(1-p)^{k_1} - k_1 p^{k_0}(1-p)^{k_1-1}\frac{\partial p}{\partial p_0}. \quad (25)$$

The maximum false-positive probability is reached when the fraction of 0 bits is $p = k_0/(k_0+k_1)$. We first rewrite Eq. (7) using the approximation in Eq. (21) as

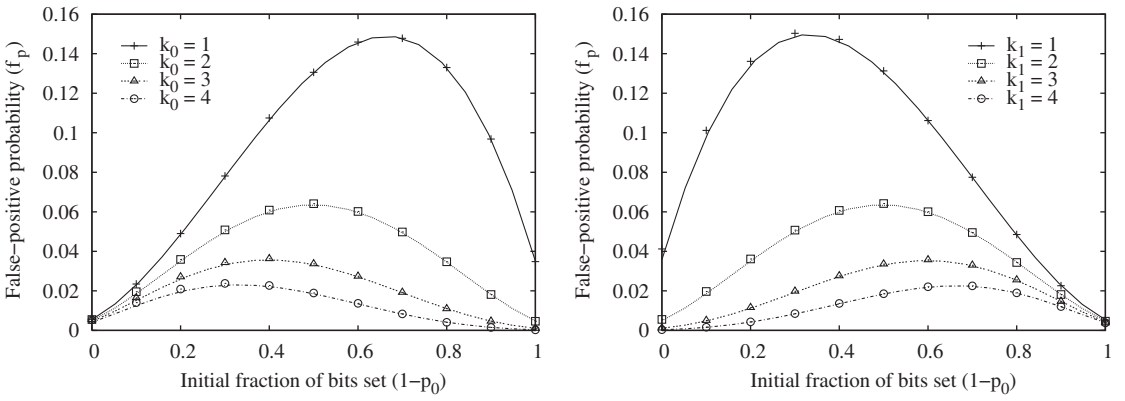
$$p = p_0(1-q_0-q_1)^n + \frac{k_0}{k_0+k_1}[1-(1-q_0-q_1)^n]. \quad (26)$$

Considering only p_0 as the variable, in order to have p in Eq. (26) satisfying $p = k_0/(k_0+k_1)$, we must have the initial condition

$$p_0 = \frac{k_0}{k_0+k_1}. \quad (27)$$

Interestingly enough, the value of p_0 that maximizes the false-positive probability is the same as that in Eq. (18). By substituting Eq. (27) back into Eq. (16), we obtain the maximum false-positive probability achieved by initial-condition tuning, which is precisely the upper bound F_p.

The false-positive probability is much less dependent on the initial condition of the filter when a GBF is used instead of a standard Bloom filter. Having all bits in 1 (i.e., $p_0 = 0\%$) is not the worst initial condition for the GBF, as it is for the standard Bloom filter. In fact, all bits in 1 or in 0 are the initial conditions that give the best results for the false-positive probability. The worst initial condition for the GBF is when the initial condition is set as in Eq. (27).



(a) $k_1 = 2$, $m/n = 25$. (b) $k_0 = 2$, $m/n = 25$.

Fig. 10. False-positive probability of a GBF as a function of the initial fraction of bits set, for (a) $k_1 = 2$, $m/n = 25$ and (b) $k_0 = 2$, $m/n = 25$.

According to Eq. (26), if the GBF is initialized as stated in Eq. (27), p remains constant independent of n. In addition, the fraction p also remains constant when too many elements are inserted into the filter (i.e., $n \to \infty$). In both cases, the constant value of p is the same as described in Eq. (18), and it leads to the upper bound F_p. When a GBF reaches the fraction of bits denoted in Eq. (18), we say it reaches a *steady state*. Once in steady state, no matter how many elements are inserted into the GBF, the fractions of bits set and reset remain constant, and the false-positive probability is F_p. Fig. 11 corroborates this result. In the figure, we observe the two ways for the GBF to reach the steady state. One can either set the initial condition of the filter according to Eq. (27) or insert too many elements into the filter. In both cases, the false-positive probability achieves the upper bound F_p.

In summary, the GBF allows an infinite number of insertions without increasing false positives, which are always bounded by F_p at steady state. On the other hand, not every inserted element is "remembered" by the GBF; some of them are overwritten by newer elements and become false negatives. However, the probability of one of the inserted elements being a false negative is also bounded by F_n.

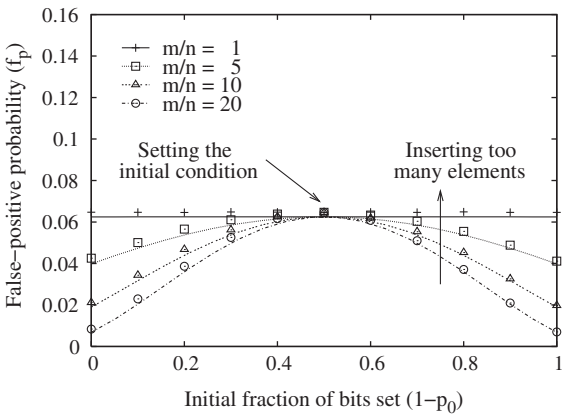


Fig. 11. False-positive probability of a GBF as a function of the initial fraction of bits set, for different values of m/n and $k_0 = k_1 = 2$.

5. Tuning false positives and false negatives

Depending on the application (e.g., IP traceback [13,14]), it may be important to reduce the probability of false negatives at the expense of a higher false-positive rate. For other applications (e.g., P2P and data-centric routing [10–12]), the opposite can be true. As a result, it is important to choose the appropriate number of hash functions that provide the ideal tradeoff between false positives and false negatives.

According to our previous observations, increasing the number of hash functions leads to a lower false-positive probability. On the other hand, it also increases the false-negative probability. For simplicity, we assume $k = k_0 = k_1$. Therefore, as we increase k, we should see F_p decrease and F_n increase. Fig. 12(a) depicts F_p and F_n as functions of the number of hash functions, for different values of m/n. As in our previous graphs, it includes simulation results as discrete points and analytical results as continuous curves. We see that, for each m/n value, there is a fixed number of hash functions that balances the upper bounds on the false-positive and false-negative probabilities.

The above result can be further generalized if we remove the constraint $k_0 = k_1$. Fig. 12(b) depicts F_p and F_n, for $m/n = 256$, as a function of k_0 and k_1. The two surfaces represent the upper bounds on the false-positive and false-negative probabilities for each (k_0, k_1) pair. As expected, we see that false positives decrease and the false negatives increase as we increase either k_0 or k_1. The intersection of both surfaces is a curve constituted by the points for which $F_n = F_p$. Depending on the application, however, it might be worthy to use a different condition for tuning.

In order to give a better intuition on how we can trade off the different parameters of the GBF, we show in the following tables a comparison between the standard Bloom filter and the GBF with regard to the average false-positive probability (f_p), average false-negative probability (f_n), upper bound on the false-positive probability (F_p), and upper bound on the false-negative probability (F_n) using analytical results. The value of k_0 should be ignored for the results of the standard Bloom filter.

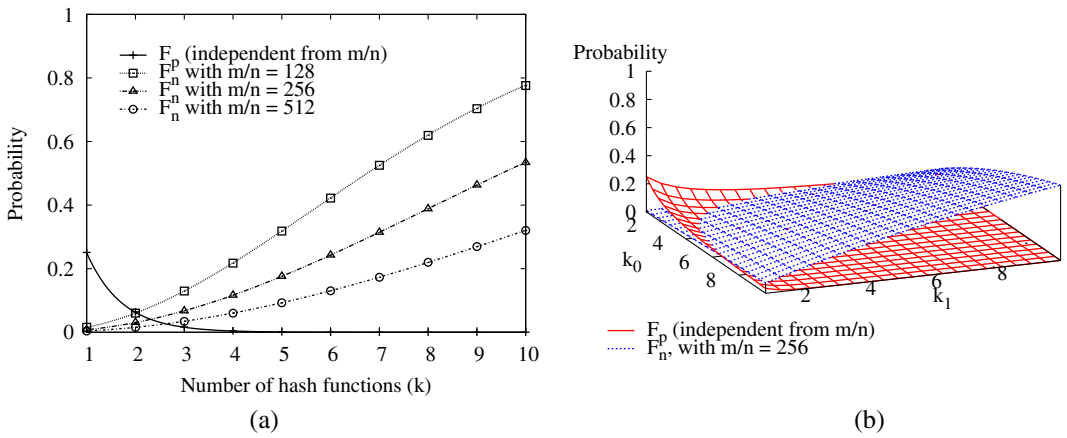


Fig. 12. (a) The upper bounds F_p and F_n of the GBF as functions of the number of hash functions, for $k = k_0 = k_1$. (b) The bounds F_p and F_n as functions of k_0 and k_1 (analytical results only).

In Table 1, we see how these metrics change with the initial condition (p_0) of the bit array. Note that this condition may not be under the control of the system designer [13,14], but we show its impact here. We use $m = 65{,}536$ bits and $n = 256$ elements for both filters, with $k = 2$ for the standard Bloom filter and $k_0 = k_1 = 2$ for the GBF. From the table we can see that $f_n = F_n = 0\%$ and $F_p = 100\%$ for the standard filter. This is always the case, since the Bloom filter does not have an upper bound on the false positives and does not have false negatives either. However, we note that the GBF is more robust to the interference of the initial condition. As the fraction p_0 of bits initially in 0 decreases, the false-positive rate of the standard filter quickly increases, reaching 100% when the filter is initially saturated. The best case for the Bloom filter is clearly when $p_0 = 100\%$. On the other hand, for these specific values of k_0 and k_1, the false-positive rate of the GBF is upper-bounded at 6.3%, and this bound is reached only when $p_0 = k_0/(k_0 + k_1) = 50\%$. Additionally, we see that f_n, F_p, and F_n do not depend on p_0 and hence do not change. Table 2 presents the same behavior, except that now we have $k_1 = 3$. Increasing the number of hash functions decreases the false positives, but it also slightly increases false negatives, as we see in the following tables.

Table 3 shows the case where the number k_1 of hash functions increases (the behavior for k_0 is similar and omitted). We have $p_0 = 50\%$, $m = 65{,}536$ bits, and $n = 256$ elements for both filters, with $k_0 = 2$ for the GBF. For the standard filter, increasing the number of hash functions is beneficial until $k = (m/n)\ln 2$, as showed in Section 2. For the GBF, increasing the number of hash functions has a dual effect. On one hand, it decreases false positives, since we need to find more bits in either zero or one for a match. On the other hand, it increases false negatives because there is a higher chance of a bit overwriting. We can see this tradeoff both in the average false-positive and false-negative rates (i.e., f_p and f_n) and in their respective upper bounds (F_p and F_n). Table 4 shows the results for the same scenario, except for a larger bit array of $m = 131{,}072$ bits. Using a larger bit array reduces false negatives (i.e., both f_n and F_n), but it has little effect on f_p and absolutely no effect on F_p.

For a better analysis of the effect of the filter size on the error rates, Table 5 shows the false-positive and false-negative rates as we increase m, the size of the bit array. We can see that the false positives of both the standard filter and the GBF are not very sensitive to a larger array, assuming the fraction p_0 of bits initially set remains the same. The false negatives, however, have a significant reduction for larger filter sizes, with roughly a $2\times$ reduction for every time we double the size of the bit array. Table 6 shows the same scenario, except for changing the initial bit distribution to $p_0 = 100\%$, which is the best case for both the

Table 1
Comparison of error rates for the Bloom filter and the GBF for different initial conditions (p_0), for $k_0 = 2$, $k_1 = 2$, $m = 65{,}536$, and $n = 256$. The false-positive probability of the standard Bloom filter increases with the reduction of p_0, whereas it remains small and bounded for the GBF.

	Bloom filter				Generalized Bloom Filter			
p_0 (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)
0	100.0	0	100	0	0.0	1.5	6.3	3.1
25	56.5	0	100	0	3.6	1.5	6.3	3.1
50	25.4	0	100	0	6.3	1.5	6.3	3.1
75	6.6	0	100	0	3.6	1.5	6.3	3.1
100	0.0	0	100	0	0.0	1.5	6.3	3.1

Table 2
Comparison of error rates for the Bloom filter and the GBF for different initial conditions (p_0), for $k_0 = 2$, $k_1 = 3$, $m = 65{,}536$, and $n = 256$. Compared to Table 1, increasing k_1 from 2 to 3 decreases the false positives at the cost of a slight increase in the false negatives.

	Bloom filter				Generalized Bloom Filter			
p_0 (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)
0	100.0	0	100	0	0.0	2.3	3.5	4.6
25	42.7	0	100	0	2.7	2.3	3.5	4.6
50	12.9	0	100	0	3.1	2.3	3.5	4.6
75	1.7	0	100	0	0.9	2.3	3.5	4.6
100	0.0	0	100	0	0.0	2.3	3.5	4.6

Table 3
Comparison of error rates for the Bloom filter and the GBF for a different number of hash functions (k_1), for $p_0 = 50\%$, $k_0 = 2$, $m = 65{,}536$, and $n = 256$. This table shows the dual effect of increasing the number of hash functions for both false positives and false negatives in a GBF.

	Bloom filter				Generalized Bloom Filter			
k_1	f_p (%)	f_n (%)	F_p (%)	F_n (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)
1	50.2	0	100	0	12.6	0.8	14.8	1.6
2	25.4	0	100	0	6.3	1.5	6.3	3.1
3	12.9	0	100	0	3.1	2.3	3.5	4.6
4	6.7	0	100	0	1.6	3.0	2.2	6.0
5	3.4	0	100	0	0.8	3.8	1.5	7.5

Table 4
Comparison of error rates for the Bloom filter and the GBF for a different numbers of hash functions (k_1), for $p_0 = 50\%$, $k_0 = 2$, $m = 131{,}072$, and $n = 256$. Compared to Table 3, increasing m from 65,536 to 131,072 bits reduces false negatives of the GBF, but has little effect on the false positives of both the Bloom filter and the GBF.

	Bloom filter				Generalized Bloom Filter			
k_1	f_p (%)	f_n (%)	F_p (%)	F_n (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)
1	50.1	0	100	0	12.6	0.4	14.8	0.8
2	25.2	0	100	0	6.3	0.8	6.3	1.6
3	12.7	0	100	0	3.1	1.2	3.5	2.3
4	6.5	0	100	0	1.6	1.5	2.2	3.0
5	3.3	0	100	0	0.8	1.9	1.5	3.8

Table 5
Comparison of error rates for the Bloom filter and the GBF for different filter sizes (m), for $p_0 = 25\%$, $k_0 = 2$, $k_1 = 2$, and $n = 256$. When p_0 is constant, larger filter sizes do not reduce false positives. However, the upper bound on the false-negative probability of the GBF is significantly reduced.

	Bloom filter				Generalized Bloom Filter			
m	f_p (%)	f_n (%)	F_p (%)	F_n (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)
8192	58.5	0	100	0	4.1	11.3	6.3	21.5
16,384	57.4	0	100	0	3.8	5.9	6.3	11.6
32,768	56.8	0	100	0	3.7	3.0	6.3	6.0
65,536	56.5	0	100	0	3.6	1.5	6.3	3.1
131,072	56.4	0	100	0	3.6	0.8	6.3	1.6

standard filter and the GBF when $k_0 = k_1$. False negatives are not affected by the initial bit distribution p_0.

Finally, Table 7 presents the best case for the standard Bloom filter as we increase the filter size m (we omitted the f_n and F_n columns for the standard filter due to space constraints). For each filter size, we use the optimal $k_1 = (m/n)\ln 2$ and the best initial condition $p_0 = 100\%$. We see that, in optimal conditions, the Bloom filter presents a low average false-positive rate (f_p), which decreases further as we increase the size of the bit array. The same behavior is seen in f_p of the GBF, which is even lower than the rate of the standard filter. On the other hand, the GBF has a high false-negative rate (f_n) of 43% and an upper bound (F_n) of roughly 70% due to the large k_1. This large number of hash functions is clearly not the optimal solution for the GBF, as seen in the previous tables, and it is

Table 6
Comparison of error rates for the Bloom filter and the GBF for different filter sizes (m), for $p_0 = 100\%$, $k_0 = 2$, $k_1 = 2$, and $n = 256$. Compared to Table 5, increasing p_0 from 25% to 100% significantly reduces false positives. False negatives at the GBF remain the same as the values in the previous table, since p_0 does not affect false negatives.

	Bloom filter				Generalized Bloom Filter			
m	f_p (%)	f_n (%)	F_p (%)	F_n (%)	f_p (%)	f_n (%)	F_p (%)	F_n (%)
8192	0.4	0	100	0	0.3	11.3	6.3	21.5
16,384	0.1	0	100	0	0.1	5.9	6.3	11.6
32,768	0.0	0	100	0	0.0	3.0	6.3	6.0
65,536	0.0	0	100	0	0.0	1.5	6.3	3.1
131,072	0.0	0	100	0	0.0	0.8	6.3	1.6

presented here just for completeness. Table 8 shows the result for same parameters, with $p_0 = 1\%$ (i.e., the filter is almost saturated from the start). In this case, we can see the high false-positive rates of the standard Bloom filter (17–90%), even for the optimal k_1. The GBF, on the other hand, is much less vulnerable to the initial condition, maintaining a low f_p (0.07–1.5%), without changing its upper bound F_p. As expected, false negatives do not change with p_0.

With this analysis, our intention was to provide a preliminary intuition on the behavior of the GBF as we change its parameters. In short, the GBF behavior can be summarized as follows:

- The GBF requires more storage space than the standard Bloom filter. However, in the presence of attackers, its main advantage is the higher offered security due to the ability of limiting the false positives.
- Increasing either k_0 or k_1 always reduces false positives at the cost of increasing false negatives. The proper balance between false positives and false negatives is application-specific.
- Increasing the m/n ratio significantly reduces the false-negative rates (i.e., f_n and F_n) and slightly reduces false positives (i.e., f_p), while having absolutely no effect on F_p. As a result, the system designer must first define the maximum tolerated false-positive rate and properly select k_0 and k_1 to reach this limit. The false-negative rate is then reduced to an acceptable level by increasing m/n.
- False positives are sensitive to the initial bit distribution p_0, specially in large bit arrays where the element insertions do not change the bit distribution by much. The false-positive rate, however, can never be higher than F_p. False negatives are not affected by the choice of p_0.

The most important property of the GBF is that we have control of false positives and false negatives and, as long as we are willing to trade space efficiency for a lower error rate, we can get these rates as low as desired.

6. Related work

The Bloom filter was initially employed in an automated hyphenation application to reduce time-consuming

Table 7
Comparison of error rates for the Bloom filter and the GBF for different filter sizes (m), using the optimal k_1 for the Bloom filter, for $p_0 = 100\%$, $k_0 = 1$, and $n = 256$. The GBF has a lower false-positive rate, but the large k_1 causes the GBF to yield a high false-negative rate. This k_1 is clearly not the best choice for the GBF.

		Bloom filter				Generalized Bloom Filter			
m	k_1	f_p	f_n	F_p	F_n	f_p	f_n	F_p	F_n
8192	22	2.1e−07	0.0e−0	1.0e−0	0.0e−0	7.9e−08	4.3e−1	1.6e−2	6.9e−1
16,384	44	4.4e−14	0.0e−0	1.0e−0	0.0e−0	1.7e−14	4.3e−1	8.3e−3	6.9e−1
32,768	89	2.0e−27	0.0e−0	1.0e−0	0.0e−0	7.2e−28	4.3e−1	4.1e−3	7.0e−1
65,536	177	3.8e−54	0.0e−0	1.0e−0	0.0e−0	1.4e−54	4.3e−1	2.1e−3	7.0e−1
131,072	355	1.5e−107	0.0e−0	1.0e−0	0.0e−0	5.4e−108	4.3e−1	1.0e−3	7.0e−1

Table 8
Comparison of error rates for the Bloom filter and the GBF for different filter sizes (m), using the optimal k_1 for the Bloom filter, for $p_0 = 1\%$, $k_0 = 1$, and $n = 256$. Compared to Table 7, decreasing p_0 from 100% to 1% significantly increases the false-positive rate of the standard filter, while false positives in the GBF still remain within an acceptable range.

		Bloom filter				Generalized Bloom Filter			
m	k_1	f_p	f_n	F_p	F_n	f_p	f_n	F_p	F_n
8192	22	9.0e−1	0.0e−0	1.0e−0	0.0e−0	1.5e−2	4.3e−1	1.6e−2	6.9e−1
16,384	44	8.0e−1	0.0e−0	1.0e−0	0.0e−0	7.9e−3	4.3e−1	8.3e−3	6.9e−1
32,768	89	6.4e−1	0.0e−0	1.0e−0	0.0e−0	4.1e−3	4.3e−1	4.1e−3	7.0e−1
65,536	177	4.1e−1	0.0e−0	1.0e−0	0.0e−0	2.0e−3	4.3e−1	2.1e−3	7.0e−1
131,072	355	1.7e−1	0.0e−0	1.0e−0	0.0e−0	6.6e−4	4.3e−1	1.0e−3	7.0e−1

dictionary word lookups [1]. Since most words can be hyphenated by applying a few simple rules, the Bloom filter is used to represent only the words that require a dictionary lookup. Each word is tested, and only those in the filter are looked up in the dictionary; the rest are managed with the simple hyphenation rules. False positives in this case cause an unnecessary lookup for a word that can be easily hyphenated. This presearch filtering algorithm was also popular in other applications, such as differential-file databases [18]. Spell checkers and password enforcers also achieved significant space savings with Bloom filters [15,19]. More recently, these filters have been widely employed in computer networks [4]. In this section, we focus on the Bloom filter variants that also allow false negatives in membership queries and on potential applications for the GBF.

6.1. Bloom filters and false negatives

Fan et al. [5] modified the standard Bloom filter to allow not only element insertion but also element deletion. The proposal is to use counters instead of a single bit in each position of the filter array. Accordingly, counters are incremented during element insertions and decremented during deletions. This so-called Counting Bloom Filter is suitable for dynamic sets, such as cache contents. The tradeoff is that a larger number of bits is required for the counters, and there is a small chance of false negatives, which happen if a counter overflows. However, using 4-bit counters reduces this probability to a negligible value [5]. False negatives can also happen if false positives are deleted from the filter. Guo et al. [20] provide a careful analysis of the error rate for this case, usually ignored in most papers.

Donnet et al. [21] introduce the Retouched Bloom Filter (RBF).[2] The key idea of the RBF is to also reset bits of the filter. In the RBF, elements are inserted the same way as in the standard Bloom filter. After the insertions, a few bits which were set during the insertions are chosen to be reset. To select these bits, the authors calculate the false-positive and false-negative rates associated with each bit set. The bits which cause high false-positive and low false-negative rates are then reset. Like the GBF, the RBF also allows interesting tradeoffs between false positives and false negatives. The key difference is that the RBF requires *a priori* knowledge of the entire universe of elements for the calculation of the false-positive and false-negative rates associated with each bit. This computation can also be intensive, since it requires testing the entire universe of elements to determine the bits responsible for most false positives. In the GBF, each element simply resets a few bits during the insertion. Another key difference is that zeros in the RBF are used to remove information from the filter, whereas zeros in the GBF may actually represent inserted elements.

Bonomi et al. [23] propose to use Bloom filters to track state machines. In its simplest form, the authors use b bits instead of a single bit in each cell of the bit array to represent $2^b - 1$ possible states. The counters are first set to an "uninitialized" state. The machine identification is then used as a key to the hash functions, and the indicated positions are set to the current state of the machine. When this state changes, the cells corresponding to that machine in the filter are set to the new state. If two machines happen to share the same cell in the filter, the shared position takes a "don't know"(DK) value to represent an undefined

[2] Note that our work [13,22] precedes the following Bloom filter variants, and also that the RBF paper refers to our technical report.

state, which is a variation of a false negative. This construction is similar to the GBF in the sense that a new element can erase information of another previously inserted element. For a simple comparison, the GBF would have to support a DK state and, if a bit is overwritten, this bit would take the DK value. False negatives can be reduced if we allow DKs to be interpreted as a wildcard, at the cost of increasing false positives. We propose a similar idea in [14] for the GBF.

Deng and Rafiei [24] introduce the Stable Bloom Filter (SBF) to detect duplicates in streaming data. Its main application is a web crawler that uses a SBF to store the recently visited pages and avoid revisiting them too often. The idea of the SBF is to probabilistically erase old pages stored in the filter in order to leave room for newer pages. Basically, the SBF uses a counter per cell instead of a single bit. Every time an element is inserted, its corresponding counters are set to the maximum value and a few randomly selected counters are decremented. During lookup, if all positions indicated by the hash functions are larger than zero, the element is assumed to be in the set. The SBF shares a few properties with the GBF, such as the tradeoff between false positives and false negatives, the convergence of p (i.e., the fraction of bits in zero), and upper-bounded false positives. The SBF is the closest variant to the GBF, and we believe this is the case due to the similarity between the process of increasing/decreasing counters in the SBF and the process of setting/resetting bits in the GBF, which happens for every element. The GBF, however, is simpler and easier to analyze; we believe this to be a great virtue for practical use.

6.2. Applications

We now describe a few applications for which the Generalized Bloom Filter could be useful. Fan et al. [5] propose that proxy servers maintain a Counting Bloom Filter to keep track of the cached web pages. Periodically, a standard Bloom filter representing the current cached pages is advertised to neighbor servers to allow cache sharing. If, however, a malicious server advertises a saturated filter, queries will be incorrectly forwarded to this server. Fake web pages can then be returned to the user, causing serious privacy issues. A GBF could be used here to limit the rate of false positives a given proxy server can yield. In the case of a false negative, a page request, which can be satisfied from a neighbor, is simply directed to the web server. As long as this probability is kept low, false negatives are not an issue in distributed caching.

Rhea and Kubiatowicz [11] introduce the Attenuated Bloom Filters for data-centric routing. An attenuated filter of depth d is an array of d standard Bloom filters where the ith filter keeps track of the documents reachable within i hops. The authors propose that nodes maintain an attenuated filter for each adjacent neighbor in the overlay network. For that purpose, each node periodically broadcasts a Bloom filter representing the documents that it can reach. False positives are harmful here because requests may be replicated towards wrong routes or even cause routing loops. Guo et al. [25] realize that false positives quickly increase in this scenario due to the aggregation effect. That is, even if the individual filters from each neighbor have a low false-positive rate, the false positives of the broadcast aggregate filter may be unacceptable. A malicious neighbor conducting an all-one attack could also disrupt this mechanism by making other nodes believe that it has the replicas of all available documents. If a GBF is used instead, false positives are upper-bounded, which avoids these problems.

The GBF is also useful in IP traceback, where the idea is to locate the source of a denial-of-service (DoS) attack. We propose to embed a GBF in packets to record the IP addresses of the traversed routers [13]. The standard Bloom filter cannot be used in this case, since the attacker can easily saturate the filter before transmission to avoid detection. We also propose an improved path reconstruction scheme where overwritten bits in the GBF can be identified and interpreted as wildcards during the path reconstruction [14].

An interesting property of the GBF that we do not explore here is that it probabilistically deletes older elements while keeping the newer ones. This interesting ability of discarding stale information can be explored in multiple applications, such as duplicate detection [24] or time-decaying aggregates in data streams [26], time-sensitive web profiling [27], data management [28], connection tracking [29], and dynamic set representation [30–32].

Other variations of Bloom filters have also been proposed for different applications [16,33–35]. None of the abovementioned variations, however, is concerned with making the false-positive probability independent of the number of bits set in the filter. We propose the Generalized Bloom Filter (GBF) which has the key idea of resetting bits of the filter during an element insertion. This procedure decreases the false positives at the expense of introducing false negatives in membership queries. We use this feature for security purposes, making the GBF robust to the state of the filter. We derive expressions relating the false-positive probability and the false-negative probability with the number of hash functions and the relative size of the filter. We show that the GBF imposes an upper bound on the false-positive probability and on the false-negative probability, which are completely independent of the state of the filter. Hence, our approach restricts the actions of malicious users on distributed applications that transmit Bloom filters over the network, and also allows trading off false positives for false negatives.

7. Conclusion

In this paper we introduce and evaluate through analytical and simulation results our generalization of Bloom filters. The Generalized Bloom Filter (GBF) is motivated by distributed applications that exchange Bloom filters over the network. The standard Bloom filter is not secure in this case since attackers may disrupt distributed applications by sending filters with all bits set to one and causing a

false-positive rate of 100% — a simple technique we define as an all-one attack. Our goal is to provide a better set representation with bounded error rates to secure these applications. The GBF has higher storage demands, but it is a requirement when sharing information between untrusted parties, such as the Internet.

The key idea of the proposed GBF is to reset bits of the filter. To accomplish this function, the GBF uses k_1 hash functions that set bits, as well as k_0 hash functions that reset bits on each element insertion. This procedure is proven to bound the false-positive probability at the expense of introducing false negatives in membership queries. We show, however, that both the false-positive and false-negative probabilities are upper-bounded in the GBF.

In conclusion, we believe that data structures that allow false positives and false negatives to be traded off are useful to fulfill different application requirements. In particular, we share the same line of thought as Bonomi et al. [23] in stating that "the best tradeoff among the different types of errors is highly application dependent, suggesting that data structures that allow such tradeoffs are more valuable."

Acknowledgments

The authors thank Luis Henrique Costa, Deborah Estrin, Natalia C. Fernandes, Marcelo D. D. Moreira, and Lixia Zhang for their comments on an earlier version of this work.

References

[1] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 7 (1970) 422–426.
[2] L.H.M.K. Costa, S. Fdida, O.C.M.B. Duarte, Incremental service deployment using the hop-by-hop multicast routing protocol, IEEE/ACM Transactions on Networking 14 (2006) 543–556.
[3] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, B. Schwartz, S.T. Kent, W.T. Strayer, Single-packet IP traceback, IEEE/ACM Transactions on Networking 10 (2002) 721–734.
[4] A. Broder, M. Mitzenmacher, Network applications of Bloom filters: a survey, Internet Mathematics 1 (2003) 485–509.
[5] L. Fan, P. Cao, J. Almeida, A.Z. Broder, Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM Transactions on Networking 8 (2000) 281–293.
[6] S. Dharmapurikar, P. Krishnamurthy, T.S. Sproull, J.W. Lockwood, Deep packet inspection using Bloom filters, IEEE Micro 24 (2004) 52–61.
[7] C. Estan, G. Varghese, New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice, ACM Transactions on Computer Systems 21 (2003) 270–313.
[8] S. Cohen, Y. Matias, Spectral Bloom filters, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, 2003, pp. 241–252.
[9] A. Kumar, J. Xu, J. Wang, O. Spatschek, L. Li, Space-code Bloom filter for efficient per-flow traffic measurement, in: Proceedings of the IEEE INFOCOM 2004 Conference, Hong Kong, China, 2004, pp. 1762–1773.
[10] F.M. Cuenca-Acuna, C. Peery, R.P. Martin, T.D. Nguyen, PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities, in: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Seattle, WA, USA, 2003, pp. 236–246.
[11] S.C. Rhea, J. Kubiatowicz, Probabilistic location and routing, in: Proceedings of the IEEE INFOCOM 2002 Conference, New York, NY, USA, 2002, pp. 1248–1257.
[12] T.D. Hodes, S.E. Czerwinski, B.Y. Zhao, A.D. Joseph, R.H. Katz, An architecture for secure wide-area service discovery, Wireless Networks 8 (2002) 213–230.
[13] R.P. Laufer, P.B. Velloso, D. de O. Cunha, I.M. Moraes, M.D.D. Bicudo, O.C.M.B. Duarte, A new IP traceback system against denial-of-service attacks, in: Twelfth international conference on telecommunications, Capetown, South Africa, 2005.
[14] R.P. Laufer, P.B. Velloso, D. de O. Cunha, I.M. Moraes, M.D.D. Bicudo, M.D.D. Moreira, O.C.M.B. Duarte, Towards stateless single-packet IP traceback, in: Proceedings of the 32nd IEEE Conference on Local Computer Networks (LCN 2007), Dublin, Ireland, 2007.
[15] D.J. Margoliash, CSPELL – A Bloom Filter-based Spelling Correction Program, Master's Thesis, Department of Computer Science, The University of Western Ontario, London, Ontario, Canada, 1987.
[16] M. Mitzenmacher, Compressed Bloom filters, IEEE/ACM Transactions on Networking 10 (2002) 604–612.
[17] M.V. Ramakrishna, Practical performance of Bloom filters and parallel free-text searching, Communications of the ACM 32 (1989) 1237–1239.
[18] D.G. Severance, G.M. Lohman, Differential files: their application to the maintenance of the large databases, ACM Transactions on Database Systems 1 (1976) 256–267.
[19] E.H. Spafford, OPUS: preventing weak password choices, Computers and Security 11 (1992) 273–278.
[20] D. Guo, Y. Liu, X. Li, P. Yang, False negative problem of counting Bloom filter, IEEE Transactions on Knowledge and Data Engineering 22 (2010) 651–664.
[21] B. Donnet, B. Baynat, T. Friedman, Retouched Bloom filters: allowing networked applications to trade off selected false positives against false negatives, in: Proceedings of the Second Conference on Future Networking Technologies (CoNEXT'06), Lisboa, Portugal, 2006, pp. 1–12.
[22] R.P. Laufer, P.B. Velloso, O.C.M.B. Duarte, Generalized Bloom Filters, Technical Report GTA-05-43, COPPE/UFRJ, 2005.
[23] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, Beyond Bloom filters: from approximate membership checks to approximate state machines, in: Proceedings of the ACM SIGCOMM'06 Conference, Pisa, Italy, 2006, pp. 315–326.
[24] F. Deng, D. Rafiei, Approximately detecting duplicates for streaming data using stable Bloom filters, in: Proceedings of the ACM SIGMOD'06 Conference, Chicago, IL, USA, 2006, pp. 25–36.
[25] D. Guo, Y. He, P. Yang, Receiver-oriented design of Bloom filters for data-centric routing, Computer Networks 54 (2010) 165–174.
[26] E. Cohen, M.J. Strauss, Maintaining time-decaying stream aggregates, Journal of Algorithms 59 (2006) 19–36.
[27] K. Cheng, M. Iwaiahra, L. Xiang, K. Ushijima, Efficient web profiling by time-decaying Bloom filters, DBSJ Letters 4 (2005) 137–140.
[28] K. Cheng, Time-decaying Bloom filters for efficient middle-tier data management, in: ICCSA, vol. 3, 2010, pp. 395–404.
[29] S.Y. Nam, H.-D. Kim, H.S. Kim, Detector SherLOCK: enhancing TRW with Bloom filters under memory and performance constraints, Computer Networks 52 (2008) 1545–1566.
[30] M. Yoon, Aging Bloom filter with two active buffers for dynamic sets, IEEE Transactions on Knowledge and Data Engineering 22 (2010) 134–138.
[31] D. Guo, J. Wu, H. Chen, X. Luo, The dynamic Bloom filters, IEEE Transactions on Knowledge and Data Engineering 22 (2010) 120–133.
[32] P.S. Almeida, C. Baquero, N. Preguita, D. Hutchison, Scalable Bloom filters, Information Processing Letters 101 (2007) 255–261.
[33] B. Chazelle, J. Kilian, R. Rubinfeld, A. Tal, The Bloomier filter: an efficient data structure for static support lookup tables, in: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, USA, 2004, pp. 30–39.
[34] J.-K. Peir, S.-C. Lai, S.-L. Lu, J. Stark, K. Lai, Bloom filtering cache misses for accurate data speculation and prefetching, in: Proceedings of the 16th International Conference on Supercomputing, New York, NY, USA, 2002, pp. 189–198.
[35] F. Hao, M. Kodialam, T. Lakshman, Building high accuracy Bloom filters using partitioned hashing, in: Proceedings of the ACM SIGMETRICS'07 Conference, San Diego, CA, USA, 2007, pp. 277–288.

Rafael P. Laufer received the B.Sc. and the M.Sc. degrees in electrical engineering from the Universidade Federal do Rio de Janeiro (UFRJ), Brazil, in 2003 and 2005, respectively. He is now working towards the Ph.D. degree in computer science at the University of California, Los Angeles (UCLA). He received the Marconi Society's Young Scholar Award in 2008. His major research interests are in networking, distributed systems, and security.

Pedro B. Velloso received the B.Sc. and M.Sc. degrees in electrical engineering from the Universidade Federal do Rio de Janeiro, Brazil, in 2001 and 2003, respectively. He received the Ph.D. degree from the Université Pierre et Marie Curie (Paris 6) in 2008. He spent one year as a post-doc researcher at Laboratoire d'Informatique de Paris 6 in 2008 and 2009. Currently, he is a research engineer at Bell Labs, France. His interests are in autonomic networks, distributed applications, wireless communications, and security.

Otto Carlos M.B. Duarte received the electronic engineer and M.Sc. degrees from the Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, in 1976 and 1981, respectively. He received a Dr. Ing. 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 months with the International Computer Science Institute (ICSI), University of California, Berkeley. In 1999 and 2001, he was an invited professor at the University Paris 6. His major research interests are in multicast, QoS guarantees, security, and mobile communications.