Generalized Bloom Filters

Rafael P. Laufer, Pedro B. Velloso, and Otto Carlos M. B. Duarte

Abstract— **In this paper, we propose the Generalized Bloom Filter (GBF), a space-efficient data structure to securely represent a set. Different from the standard Bloom Filter, the GBF imposes an upper bound on the false-positive probability. The key idea of the GBF is to reset bits of the filter. For that purpose, the GBF uses both hash functions that set and hash functions that reset bits. 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 probabilities and show that they are both upper-bounded in the GBF. We also show that these upper bounds depend only on the number of hash functions used, k_0 and k_1, and on the relative size of the filter, m/n. As a result, a saturated filter cannot yield high false-positive ratios anymore. Simulations are conducted to validate the expressions. We show that the GBF is robust and fits well for security purposes, such as securing distributed applications that transmit the filter over the network.**

Index Terms— **Bloom Filters, data structures, network security.**

I. 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 both save space and allow efficient member lookup. The tradeoff cost associated with hashing is the introduction of false positives in the membership queries. A false positive 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 if 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 event occurs when all indicated bits were previously set by other legitimate elements. As more and more elements are inserted into the filter, the false-positive probability continuously increases since the filter becomes more and more saturated. Despite this inherent drawback, Bloom Filters are quite efficient if the probability of a false positive is kept low.

Bloom Filters have been used in many different computer science areas, including spell checkers, database applications, and networking [2], [3]. Numerous proposed networking applications use Bloom Filters to efficiently exchange information [4]. For instance, in a cache sharing application, 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 proxy server, the page is requested to this neighbor instead of to the actual web server. Bloom Filters are also used in peer-to-peer (P2P) applications and in resource routing [6]–[8]. The idea is that each node keeps track of the objects reachable through other nodes with Bloom Filters. As a consequence, each node periodically broadcasts a Bloom Filter representing the objects located at or reachable through itself. Bloom Filters are also applied in IP traceback [9], [10]. 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. From the filter of a received attack packet, the victim initiates membership tests with its neighbor routers to identify the one which forwarded the packet. A recursive procedure is performed on each identified router to reconstruct the packet path back to its true source.

Security issues, however, restrict the deployment of standard Bloom Filters in such distributed applications. For instance, a well-equipped attacker could generate offending filters with all bits set to one, a technique we define as the *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. In the web cache sharing and P2P applications, 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 reconstruction procedure, every router is recognized as a member of the filter and the path reconstruction is completely ineffective.

From the 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.

In this paper, we propose a novel tamper-resistant data structure called Generalized Bloom Filter (GBF) and explore its

Rafael P. Laufer is with the Computer Science Department, University of California at Los Angeles, USA (email: rlaufer@cs.ucla.edu).

Pedro B. Velloso is with the Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie, France (email: pedro.velloso@lip6.fr).

Otto Carlos M. B. Duarte is with the Grupo de Teleinformática e Automação (GTA), Universidade Federal do Rio de Janeiro, Brazil (email: otto@gta.ufrj.br).

properties through analytical and simulation results. The basic idea of the GBF is to reset bits of the filter. For that 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 and consequently the action of an attacker in creating false positives is 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 by using larger bit arrays. Therefore, GBFs can be successfully employed in distributed applications that require a secure and space-efficient set representation.

The rest of the paper is organized as follows. A brief overview of Bloom Filters is described in Section II. The proposed generalization is then outlined in Section III along with a theoretical analysis. In Section IV, we detail our simulation environment of the Generalized Bloom Filter and compare analytical and simulation results. The simulation results corroborate our proposed analysis and we also show that both the false-positive and the false-negative probabilities are upper bounded. Section V presents the tradeoffs between false positives and false negatives. In Section VI, we review the related work and other extensions proposed to Bloom Filters. Finally, conclusions are presented in Section VII.

II. The Bloom Filter

In this section, an overview of the Bloom Filter is described following the analysis of Fan *et al.* [5], Margoliash [11], and Mitzenmacher [12].

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 built by the following rules. First, all the 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 check 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 all set. If at least one bit is reset, 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 anomaly occurs when the bits $h_1(x), h_2(x), \ldots, h_k(x)$ were all previously set by other inserted elements.

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

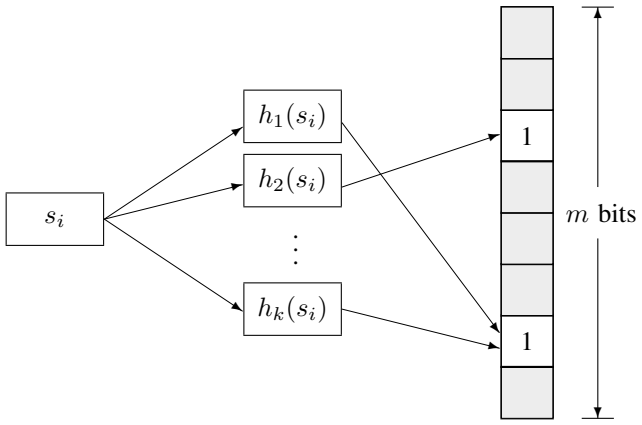


Fig. 1. Insertion of an element into a Bloom Filter.

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

Since the same computation can be applied for every bit in the array, on average, the fraction of bits reset after all the n insertions is $p \approx e^{-kn/m}$ [12]. Hence, the fraction of bits set after the n insertions is $(1-p)$. The probability of a false positive f is the probability that we find a bit set for each of the k indicated positions, that is

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

It is worth mentioning that the mean number of collisions per element is assumed to be zero in (2). A collision occurs if two or more different hash functions map an element to the same output value. Fig. 1 depicts a collision between the hash functions h_1 and h_k for the same element. According to the above assumption, on average, for each element the hash functions indicate different positions in the bit array. It is easy to check that this approximation is only valid when $m \gg k$. If this condition is not satisfied, however, we also need to consider the false-positive probabilities of the cases where $1, 2, \ldots, k-1$ collisions occur.

Two interesting remarks can be made considering the false-positive probability f in (2). First, increasing the relation m/n always reduces the false-positive probability. By using more bits per element, the fraction of bits set $(1 - e^{-kn/m})$ is reduced and thus the false-positive probability decreases. Fig. 2 depicts this behavior. 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 set and accordingly a higher false-positive rate. On the other hand, the probability of finding every indicated bit set decreases with higher values of k. Mathematically, this tradeoff is observed in (2) by noticing that the fraction of bits set $(1 - e^{-kn/m})$ increases with k and that the expression a^k decreases with k, for $0 < a < 1$. This tradeoff is depicted in Fig. 3, where an optimal value that minimizes f is clearly observed for each m/n ratio.

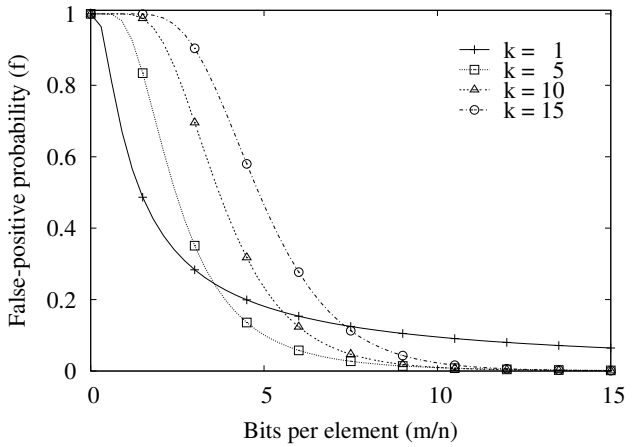


Fig. 2. False-positive probability of a Bloom Filter as a function of the number of bits per element.

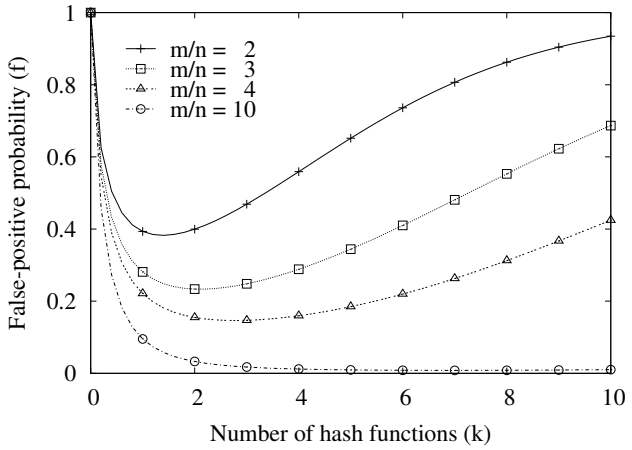


Fig. 3. False-positive probability of a Bloom Filter as a function of the number of hash functions.

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

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

Setting (3) to zero, noting that $(1-e^{-kn/m})^k$ is never null and cross-multiplying gives

$$\begin{aligned}(1-p)\ln(1-p) &= p\ln p\\ (1-p)^{(1-p)} &= p^p.\end{aligned} \tag{4}$$

Assuming k is finite and positive, the only solution to (4) is $p = 0.5$. Therefore, the minimum false-positive probability is achieved when approximately half of the bits are set and half of the bits are reset. In this case, $k = (m/n)\ln 2$ and the false-positive probability f is $(0.5)^k = (0.6185)^{m/n}$. In practice, however, k as well as m and n must be integers.

One problem of the Bloom Filter is that its false-positive probability strongly depends on the number of bits set in the array. Fig. 4 shows the false-positive probability of a Bloom Filter as a function of the fraction of bits set. 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 set increases as more elements are inserted into the filter. In distributed applications that transmit the Bloom Filter over the network [5]–[9], these inherent properties are harmful and can be easily exploited using an all-one attack. Therefore, we propose in the next section a generalization to Bloom Filters that has an upper-bounded false-positive rate independent of the number of bits set in the array. 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.

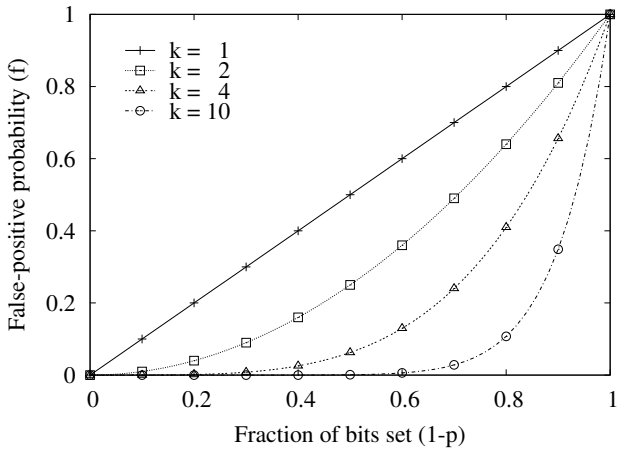


Fig. 4. False-positive probability of a Bloom Filter as a function of the fraction of bits set in the array.

III. The Generalized Bloom Filter (GBF)

As the standard filter, the Generalized Bloom Filter is also a data structure used to represent a set $S = \{s_1, s_2, \ldots, s_n\}$ 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 zero 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 or reset several times without restrictions. Fig. 5 depicts how an element is inserted into a Generalized Bloom Filter (GBF). After inserting the elements, membership queries can be easily conducted. To check if an element x belongs to S, we check if the bits of the array corresponding to the positions $g_1(x), g_2(x), \ldots, g_{k_0}(x)$ are all reset and if the bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ are all set. 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 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 reset and the bits $h_1(x), h_2(x), \ldots, h_{k_1}(x)$ are all set due to other inserted elements or due to the initial condition of the bit array.

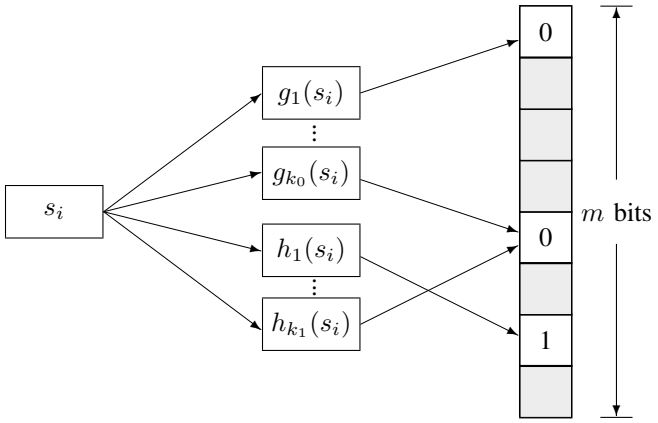


Fig. 5. Insertion of an element into a Generalized Bloom Filter.

A. False Positives

The false-positive probability of our Generalized Bloom Filter (GBF) is calculated in a similar way to the standard filter. Nevertheless, we need first to 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

$$q_0 = \left[1 - \left(1 - \frac{1}{m}\right)^{k_0}\right] \approx \left(1 - e^{-k_0/m}\right). \quad (5)$$

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

$$\begin{aligned} q_1 &= \left[1 - \left(1 - \frac{1}{m}\right)^{k_1}\right]\left(1 - \frac{1}{m}\right)^{k_0} \\ &\approx \left(1 - e^{-k_1/m}\right) e^{-k_0/m}. \end{aligned} \quad (6)$$

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

$$(1 - q_0 - q_1) = \left(1 - \frac{1}{m}\right)^{k_0+k_1} \approx e^{-(k_0+k_1)/m}. \quad (7)$$

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 at each element insertion. For a m-bit array, we have on average $b_0 = m.q_0$ bits reset, $b_1 = m.q_1$ bits set, and $(m - b_0 - b_1)$ bits neither set nor reset at each insertion.

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

$$\begin{aligned} p &= p_0 (1 - q_0 - q_1)^n + \sum_{i=0}^{n-1} q_0 (1 - q_0 - q_1)^i \\ &= p_0 (1 - q_0 - q_1)^n + q_0 \left[\frac{1 - (1 - q_0 - q_1)^n}{1 - (1 - q_0 - q_1)}\right] \\ &= p_0 (1 - q_0 - q_1)^n + \frac{q_0}{q_0 + q_1} \left[1 - (1 - q_0 - q_1)^n\right]. \end{aligned} \quad (8)$$

Since the same computation applies for every bit in the array, on average, a fraction of p bits is reset and a fraction of $(1-p)$ bits is set after n insertions.

From the bit-array distribution, the probability of a false positive is easily calculated. Since on average b_0 bits are reset and b_1 bits are set on each element insertion, the probability of a false positive f_p for the GBF is calculated as

$$f_p = p^{\,b_0} (1 - p)^{b_1}. \quad (9)$$

B. False Negatives

False positives can happen when non-inserted elements are tested against the filter. 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 by a succeeding element and that bit remains inverted until the end of the insertions. 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 $0 \leq i \leq n-1$. The probability $p_{00}(n-i)$ that a bit reset by the $(n-i)$-th element remains zero 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 $0 \leq j \leq i-1$. Therefore,

$p_{00}(n-i)$ is expressed by

$$\begin{aligned} p_{00}(n-i) &= (1-q_0-q_1)^i + \sum_{j=0}^{i-1} q_0\,(1-q_0-q_1)^j \\ &= (1-q_0-q_1)^i + q_0\left[\frac{1-(1-q_0-q_1)^i}{1-(1-q_0-q_1)}\right] \\ &= (1-q_0-q_1)^i + \frac{q_0}{q_0+q_1}\left[1-(1-q_0-q_1)^i\right]. \end{aligned} \tag{10}$$

In the same way, the probability $p_{11}(n-i)$ that a bit set by the $(n-i)$-th element remains one by the end of the following i insertions is given by

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

From (10) and (11), 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. Accordingly, this probability is

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

C. 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 (9) is reduced to the probability of the standard filter when its parameters are used, that is, $k_0 = 0$ and $p_0 = 1$. In this case, $p = e^{-k_1 n/m}$, $b_0 = 0$, and $b_1 = m(1-e^{-k_1/m})$. Noticing that $m \gg k_1$ and using an expansion expression for b_1, we get to the simplification made for the standard filter

$$\begin{aligned} b_1 &= m\left(1-e^{-k_1/m}\right) \\ &= m\left[1-\left(1-\frac{k_1}{m}+\frac{k_1^2}{2m^2}-\frac{k_1^3}{6m^3}+\cdots\right)\right] \approx k_1. \end{aligned} \tag{13}$$

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

Likewise, the false-negative probability in (12) equals zero for the standard Bloom Filter. In this case, $k_0 = 0$ and therefore $b_0 = 0$ and $p_{11}(n-i) = 1$, for $0 \le i \le n-1$. Thus, the probability of a false negative is always zero. Additionally, for the last inserted element, the n-th element, the false-negative probability also equals zero since no other element can invert any of its bits. In this case, $i = 0$ and $p_{00}(n) = p_{11}(n) = 1$; therefore, the probability of a false negative is also zero.

IV. Results

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 a given 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 and false-negative rates. For the independent and random hash functions assumed in Sections II and III, we used in our simulations a universal class of hash functions [13]. This class is defined as follows. Let the elements be integers from the universe $S = \{1, 2, \ldots, z-1\}$, 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 $x \in S$ into the respective range is

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

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 (14). For each hash function, c and d are arbitrarily chosen. The integer z is defined as a large prime.

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.

We must emphasize that the simulation results are extremely 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. It means that, besides matching the analytical results, the simulation results have also 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.

A. Upper-bounding the False-Positive Probability

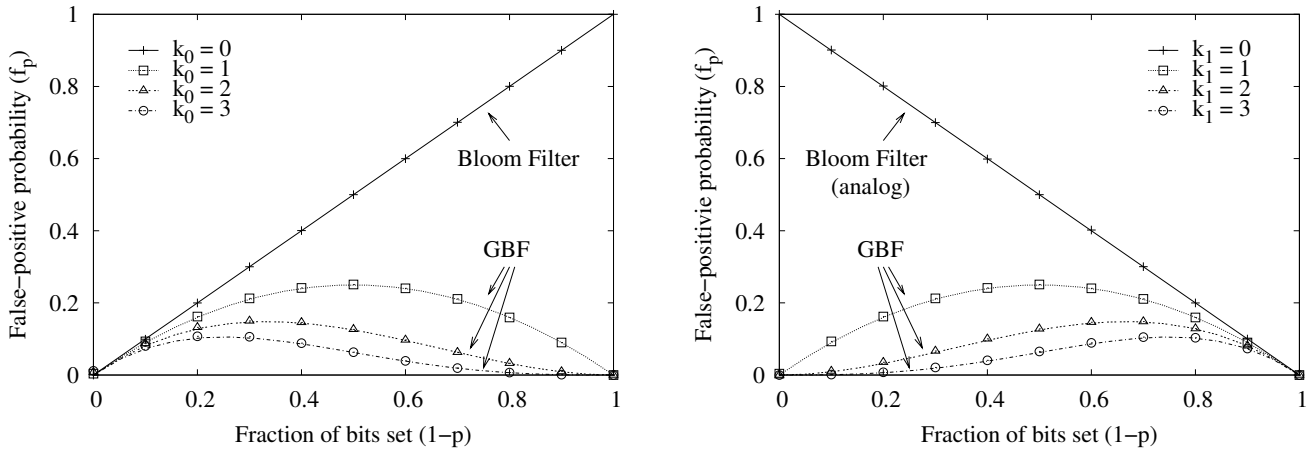
In membership queries, a false positive implies recognizing an element that does not belong to S as a legitimate one. As the false-positive probability increases, more and more non-members are incorrectly identified as belonging to the set. Therefore, a low false-positive probability is desired.

First, we note that (8) and (9) can be simplified if we assume that $m \gg k_0$ and $m \gg k_1$. This assumption is quite reasonable since usually the size of the filter is much larger than the number of hash functions used. In this case, we rewrite (8) as follows

$$p = p_0\,(1-q_0-q_1)^n + \frac{k_0}{k_0+k_1}\left[1-(1-q_0-q_1)^n\right]. \tag{16}$$

In addition, we also rewrite $b_0 \approx k_0$ and $b_1 \approx k_1$. Accordingly, the simplified probability of a false positive using these assumptions is

$$f_p \approx p^{\,k_0}(1-p)^{k_1}. \tag{17}$$



(a) $k_1 = 1$. (b) $k_0 = 1$.

Fig. 6. 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 shows the false-positive probability of a GBF f_p as a function of $(1-p)$, according to (16), (17), and our simulation results. The probability $(1-p)$ can also be seen as the fraction of bits set after inserting n elements. In Fig. 6(a), we see that the false-positive probability of the standard Bloom filter increases with $(1-p)$, which is in accordance with (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 reset but it is hard to find a bit set. On the other hand, if $(1-p)$ is high, it is easy to find a bit set and difficult to find a bit reset. In Fig. 6(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 reset. The other curves represent the analogs of the GBFs of Fig. 6(a).

Fig. 7 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 zero or one to yield a false positive. In addition, we see that the maximum false-positive probability decrease exponentially with k. As we explain next, the maximum false-positive probabilities decay with $1/4^k$, for the case where $k = k_0 = k_1$.

By differentiating (17) with respect to p, it is easy to check that the maximum false-positive probability of a GBF is reached when

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

The maximum false-positive probability F_p is determined by substituting the result of (18) back into (17). It is given by the expression

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

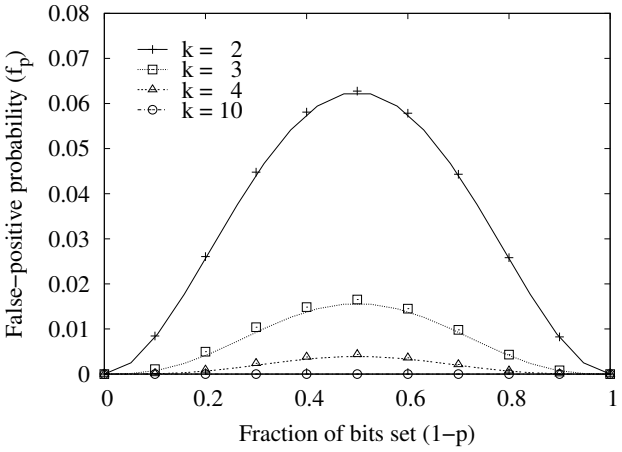
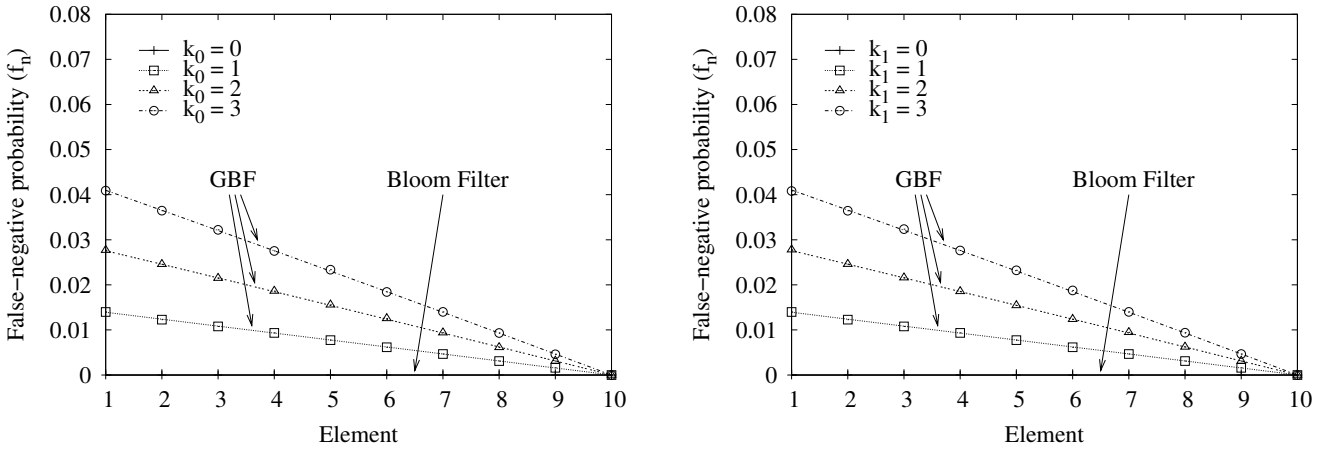


Fig. 7. False-positive probability of a GBF as a function of the fraction of bits set, for $k = k_0 = k_1$.

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%. 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 a maximum false-positive probability of only 6.3%. If we increase the number of hash functions to $k_0 = k_1 = 3$ and $k_0 = k_1 = 4$, the maximum false-positive probability drops to 1.6% and 0.4%. This probability can be further reduced if a larger number of hash functions is used.

B. Upper-bounding the False-Negative Probability

Due to the hash functions that reset bits in the filter, an upper bound is imposed on the false-positive probability.



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

Fig. 8. False-negative probability of each element inserted into a GBF, for (a) $k_1 = 1, n = 10, m/n = 128$ and (b) $k_0 = 1, n = 10, m/n = 128$.

The tradeoff cost is the introduction of false negatives in membership queries, which did not exist in the standard filter. A false negative implies not detecting an actually inserted element.

Fig. 8(a) shows the false-negative probability for each element $(n-i)$, for $0 \leq i \leq n-1$, according to (12) and our simulation results, using $k_1 = 1$, $n = 10$, and $m/n = 128$. Assuming our previous definitions, 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 the first elements have more elements inserted after them and, as a consequence, the probability of inverting their 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 is the probability of an element to have one of its marked bits inverted by a subsequent element. The result is approximately the same if k_0 is fixed and k_1 increases instead, as shown in Fig. 8(b). The figures look very similar because the false-negative probability is approximately symmetric regarding k_0 and k_1, according to (10), (11), and (12). This result becomes more evident in the simplified versions of these equations, (20), (21), and (22), explained later in this section.

Fig. 9 depicts the behavior of the Generalized Bloom Filter when we increase the number of hash functions. 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 get inverted by the last elements. From this and the previous figures, we can think of the GBF as a "memory" buffer where 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, but also on the relative size of the filter. The intuition behind this last statement is that a larger filter implies a larger output range for the hash functions. With a larger range, the probability of one element overwriting the bits of previous elements decreases.

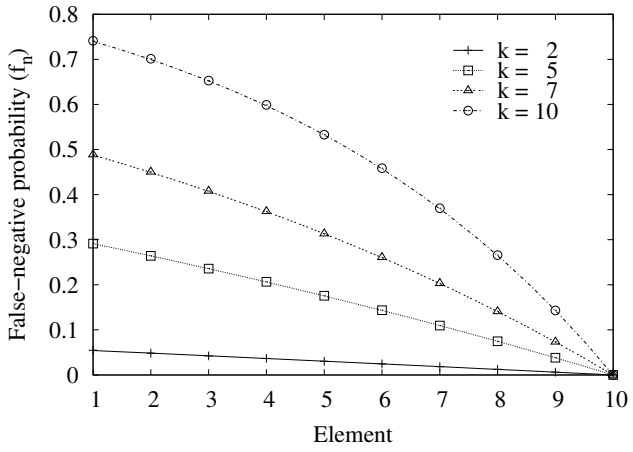


Fig. 9. False-negative probability of a GBF for each inserted element, for $k = k_0 = k_1$, $n = 10$, and $m/n = 128$.

From Fig. 7 and 9, 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 IV-A. On the other hand, it increases the false-negative probability. We return to this dual effect in Section V.

According to Fig. 8, the false-negative probability is a monotonically decreasing function of the number of inserted elements. In fact, this behavior is always true and it is analytically proven if we observe that, for two elements x and y with x inserted before y, $p_{00}(x) \leq p_{00}(y)$ and $p_{11}(x) \leq p_{11}(y)$. From (12), it is trivial to check then that $f_n(x) \geq f_n(y)$ is always true. Therefore, we 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 $f_n(0) \geq f_n(1) \geq f_n(2) \geq \ldots \geq f_n(n)$ always

holds. Accordingly, we define the maximum false-negative probability of a GBF as $f_n(0)$. An advantage in doing so is that we represent the maximum false-negative probability as a function of only k_0, k_1, and the m/n ratio.

Assuming $m \gg k_0$ and $m \gg k_1$, we first rewrite (10) and (11) for the hypothetical element 0 as

$$\begin{aligned} p_{00}(0) &= (1-q_0-q_1)^n + \frac{k_0}{k_0+k_1}\left[1-(1-q_0-q_1)^n\right] \\ &= e^{-(k_0+k_1)n/m} + \frac{k_0}{k_0+k_1}\left(1-e^{-(k_0+k_1)n/m}\right), \end{aligned} \tag{20}$$

$$\begin{aligned} p_{11}(0) &= (1-q_0-q_1)^n + \frac{k_1}{k_0+k_1}\left[1-(1-q_0-q_1)^n\right] \\ &= e^{-(k_0+k_1)n/m} + \frac{k_1}{k_0+k_1}\left(1-e^{-(k_0+k_1)n/m}\right). \end{aligned} \tag{21}$$

Substituting (20) and (21) back into (12), and noticing that $b_0 \approx k_0$ and $b_1 \approx k_1$, the maximum false-negative probability F_n is defined as

$$F_n = f_n(0) \approx 1 - p_{00}(0)^{k_0} p_{11}(0)^{k_1}. \tag{22}$$

According to (20), (21), and (22), F_n is uniquely defined by k_0, k_1, and the m/n ratio. Therefore, the system designer may first arbitrate the desired maximum false-positive probability of the GBF by defining both k_0 and k_1. The desired maximum false-negative probability 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. The chosen values for the maximum false-positive and false-negative probability are strongly related to the application. The introduction of false negatives may not be harmful to certain applications (e.g., web cache sharing) if maintained low enough.

Fig. 10 depicts the maximum false-negative probability of a GBF as a function of the m/n ratio, according to (22) 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, decreasing the probability of a bit overwriting. A similar result is presented when k_0 is fixed and k_1 changes (not shown).

According to the results mentioned so far, the GBF is capable of representing 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 GBF yields no more than 6.3% false positives and no more than 6.0% false negatives. If lower rates are desired, we can use $k_0 = 2$, $k_1 = 3$, and $m/n = 256$ bits per element. In this case, we have a maximum false-positive probability of 3.5% and a maximum false-negative probability of 4.6%. False negatives are further reduced if we use $k_0 = 2$, $k_1 = 3$, and $m/n = 512$ bits per element. In that case, we get a maximum false-negative probability of only 2.3%. The value of these parameters can be further increased as long as we are willing to reduce the false-positive and false-negative probabilities.

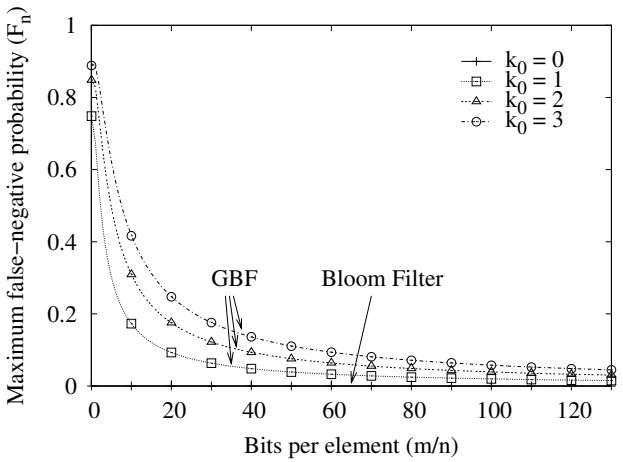


Fig. 10. Maximum false-negative probability of a GBF as a function of the number of bits per element, for $k_1 = 1$.

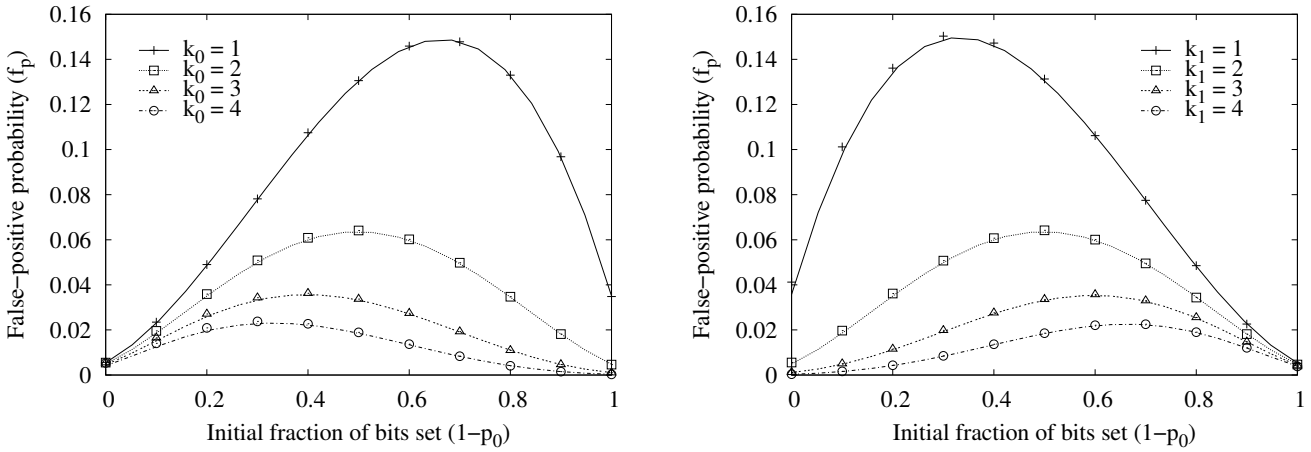
C. Interference of the Initial Condition

When all bits of the standard Bloom Filter are initially set, the false-positive probability reaches 100%, as shown in Fig. 4. In this state, the filter is useless since it cannot differentiate elements of the set from external elements. In contrast, we can always build a Generalized Bloom Filter with pre-determined maximum values for the false-positive and false-negative probabilities, regardless of the initial condition of the filter. The GBF is robust to the initial condition due to the use of hash functions that both set and reset bits. In Section IV-B, we show that false-negative probability is upper-bounded and its maximum value does *not* depend on the initial condition of the filter. The false-positive probability is the only one that changes with the initial condition, but we show in this section that its interference is limited.

Fig. 11 shows how the false-positive probability of a GBF is affected by the initial fraction of bits set, $(1-p_0)$, according to (16), (17) and simulation results. We notice from Fig. 11(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. 11(b). By differentiating (17) with respect to p_0, it is trivial to notice that the maximum false-positive probability is reached when

$$p_0 = \frac{k_0}{k_0+k_1}. \tag{23}$$

Interestingly enough, the value of p_0 which maximizes the false-positive probability is the same as the one in (18). By substituting (23) back into (17), we obtain the maximum false-positive probability achieved by initial-condition tuning, which is precisely F_p. Accordingly, the false-positive probability is much less dependent on the initial condition of the filter when a GBF is used rather than a standard Bloom Filter. It is worth mentioning, that all bits set is not the worse initial condition for the Generalized Bloom Filter, as it is for the standard Bloom Filter. In fact, all bits set or all bits reset are the initial conditions that give the best results for the false-positive probability. The worse initial condition for the GBF is when the initial condition is set as in (23).



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

Fig. 11. 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 (16), being the GBF initialized as stated in (23), p remains constant independent of n. In addition, the fraction p also remains constant when too many elements are inserted into the filter, that is, n is large. In both cases, this constant value is the same as described in (18) and it leads to the maximum false-positive probability F_p. When a GBF reaches the fraction of bits denoted in (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. 12 corroborates this result. In the figure, we notice the two ways of a GBF reaching the steady state. One can either set the initial condition of the filter according to (23) or insert too many elements into the filter. In both cases, the false-positive probability achieves its maximum value F_p.

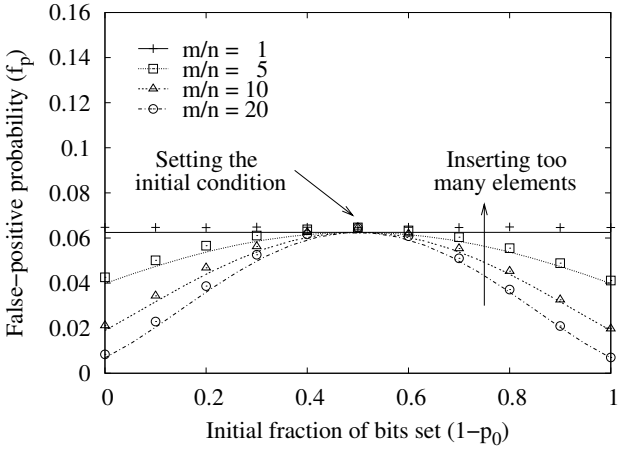


Fig. 12. False-positive probability of a GBF as a function of the initial fraction of bits set, for different values of m/n and $k_0 = k_1 = 2$.

V. Tunning False Positives and False Negatives

Depending on the application, it may be important to reduce the probability of false negatives at the expense of a higher false-positive rate. For other applications, 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 probability of false positives. On the other hand, it also increases the probability of false negatives. For simplicity, lets first assume $k = k_0 = k_1$. Therefore, as we increase k we should see the maximum false-positive probability F_p decrease and the maximum false-negative probability F_n increase. Fig. 13 depicts F_p and F_n as functions of the number of hash functions, for different values of m/n. It includes simulation results as discrete points and analytical results as continuous curves. We see that for each m/n value there is a number of hash functions that balances the maximum false-positive and the maximum false-negative probabilities.

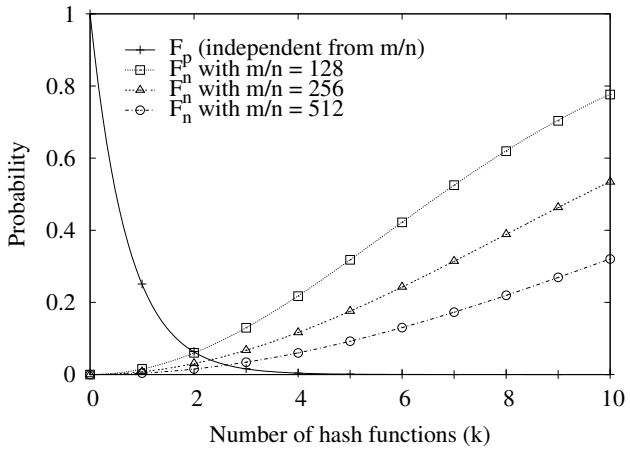


Fig. 13. The maximum false-positive probability F_p and the maximum false-negative probability F_n of a GBF as functions of the number of hash functions, for $k = k_0 = k_1$.

The above result can be further generalized if we remove the constraint $k_0 = k_1$. Lets first assume that both k_0 and k_1 are real numbers. The set K of pairs (k_0, k_1) that balances

the false-positive and false-negative in this case is defined as

$$K = \left\{(k_0, k_1) \in \mathbb{R}^2 \mid F_n = F_p\right\}. \quad (24)$$

Fig. 14 depicts the maximum false-positive probability F_p and the maximum false-negative probability F_n, for $m/n = 256$, as a function of k_0 and k_1. 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 of K in (24). Although we assumed before that both k_0 and k_1 are real numbers, in practice, these numbers must be integers. As a result, depending on the m/n ratio, it may only be possible to reach approximate values.

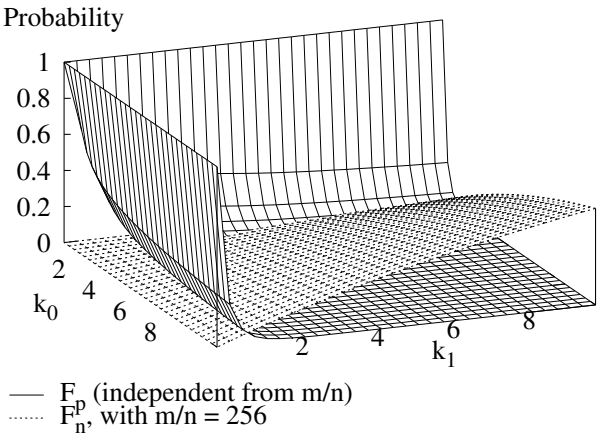


Fig. 14. The maximum false-positive probability F_p and the maximum false-negative probability F_n of a GBF as functions of the number of hash functions (analytical results only).

In (24), we consider false positives and false negatives to have the same weight. Depending on the application, however, it might be worthy to have a higher false-positive probability in exchange of a lower false-negative probability. Such tuning is easily achieved by changing the $F_n = F_p$ condition in (24). If, for instance, we would rather have three times less false negatives than false positives, we just replace the above condition for $F_n = F_p/3$. The final set of pairs in K that could be practically used in this case is

$$K = \left\{(k_0, k_1) \in \mathbb{N}^2 \mid F_n \approx F_p/3\right\}. \quad (25)$$

VI. Related Work

The Bloom Filter was initially employed in an automated hyphenation application to reduce 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. 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 [14]. Spell checkers and password enforcers also achieved significant space savings with Bloom Filters [11], [15]. More recently, these filters have been widely employed in computer networks [4].

The original Bloom Filter has been constantly adapted to fulfill application requirements. Fan *et al.* [5] modified the standard 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. In the paper, proxy servers maintain a counting 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. Note that the problem is not related to the authentication, privacy, or integrity of the transmitted filter, but rather the ability of the data structure to allow mapping the entire universe into a few bits. The GBF does not suffer from this problem since its false-positive probability is bounded.

Another variation, denoted Compressed Bloom Filters, was introduced by Mitzenmacher [12] to allow efficient filter transmission between end hosts. According to simple calculations (see Section II), optimal Bloom Filters yield minimum false-positive rates when half of the bit array is set and the other half is reset. Compression is therefore completely ineffective in such cases. Nonetheless, Mitzenmacher suggests using larger sub-optimal filters at end hosts and compressing these filters just before transmission. The author concluded that compressed filters always yield equal or lower false-positive rates than standard optimal filters for a fixed transmission size. The tradeoff cost is the additional memory and the compression/decompression processing overhead at end hosts. Mitzenmacher deals with the tradeoffs between compression and false positives in standard filters, but does not address the Bloom Filter's inherent property of allowing high false-positive rates. Instead, we propose a radical change in the filter structure to limit false positives and we also show that the false-negative probability is upper-bounded.

Dharmapurikar *et al.* [16] describe a hardware-based packet inspection technique based on Parallel Bloom Filters. The goal is to efficiently detect predefined strings of bytes, also denoted as signatures, within network packets. The authors propose grouping all the signatures according to their length and storing the signatures of each group in a different Bloom Filter. Multiple filters are used to store strings of different lengths. Accordingly, variable-length signatures are checked in parallel using the corresponding filters. Another application proposed by the authors is to use Parallel Bloom Filters for longest-prefix matching in routers [17]. The key idea of both applications is to store elements of the same size in a common Bloom Filter and to query all filters simultaneously. As a result, the lookup latency is reduced and the appropriate action can be taken depending on the matching filter.

An interesting approach is adapting the Bloom Filter to represent multisets rather than sets. A multiset is much similar

to a set, except that a given element may occur more than once. Estan and Varghese [18] introduce a multiset-capable Bloom Filter to identify heavy flows in a network router. It uses counters in the filter array rather than single bits. Upon receiving a packet, the router increments the counters of the corresponding flow by the packet size. When all counters of a given flow are above a certain threshold, the flow is classified as heavy. Cohen and Matias [19] introduce a similar approach to allow not only membership queries but also multiplicity queries. Given a specific element, the so-called Spectral Bloom Filter returns an estimate of its number of occurrences in the filter. An alternative multiset representation, the Space-Code Bloom Filter, is introduced by Kumar *et al.* [20]. The authors propose using several predefined groups of hash functions instead of one fixed group of hash functions as in the standard filter. On each element insertion, one group of hash functions is randomly selected and the bits indicated by the hash functions of the chosen group are then set in the filter. The multiplicity in this case is estimated by the number of groups whose hash functions indicate only bits set.

Shanmugasundaram *et al.* [21] introduce a data structure called Hierarchical Bloom Filter to support substring matching. Accordingly, it is possible to know if a part of a string was inserted into the filter with low false-positive rates. First, each string is split into several fixed-size blocks which are then inserted into a standard Bloom Filter. As a result, we can check for substrings with a block-size granularity. Nevertheless, incorrect substring matches may occur by combining blocks from different strings. For instance, a substring composed of two blocks from different strings would be incorrectly recognized as an inserted substring. For improved accuracy, pairs of subsequent blocks are concatenated and also inserted into the filter. Therefore, two subsequent single-block matches can be confirmed by a pair-block lookup. If more accuracy is desired, subsequent pairs can also be concatenated and inserted, and so on. More precise results, however, require increasing storage requirements.

Bonomi *et al.* [22] use Bloom Filters as state machines. In its simplest form, the authors use b bits instead of a single bit in each position of the bit array to represent 2^b-1 possible states. When the state of a given machine changes, the positions corresponding to that machine in the array are set to the new state. If two machines happen to share the same position in the filter, the shared position takes a special value to represent an undefined state.

The four previous works use standard or modified Bloom Filters for internal tasks and do not transmit the filter over the network. As a result, there is no need to worry about receiving a saturated filter that disrupts the application. The standard filter is indeed more appropriate in these cases, since less storage overhead is introduced. When filters are exchanged between end hosts, however, one must use the GBF to make the application robust to all-one attacks.

To improve the location and routing of document replicas in peer-to-peer networks, Rhea and Kubiatowicz [7] introduce the Attenuated Bloom Filters. An attenuated filter of depth d is an array of d standard Bloom Filters where the i-th 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 broadcast a Bloom Filter representing the documents that it can reach. Nevertheless, a malicious neighbor conducting an all-one attack could disrupt this mechanism by making other nodes believe that it has the replicas of all available documents. If a GBF is used instead, such scenario is impossible due to the GBF's ability to limit the false-positive ratio.

Donnet *et al.* [23] introduce the Retouched Bloom Filter (RBF). 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 in one are chosen and set to zero. To choose the 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 ratios are then reset. Even though the proposed filter allows interesting tradeoffs, the RBF is still vulnerable to all-one attacks, since the membership tests are conducted in the same way as in the standard filter. There are no proven upper bounds on the false-positive and false-negative probabilities. Additionally, the authors assume that the entire universe of elements is available and can be easily tested against the filter to derive the false-positive ratio associated with each bit.

Other variations of Bloom Filters have also been proposed for different applications [24], [25]. None of the above-mentioned 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. 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 proposition restricts the actions of malicious users on distributed applications that transmit Bloom Filters over the network.

VII. Conclusion

In this paper, we introduce and evaluate through analytical and simulation results our generalization of Bloom Filters. The so-called Generalized Bloom Filter (GBF) is motivated by distributed applications that exchange Bloom Filters over the network and applications that need to limit the false-positive probability at the cost of tolerating a bounded value of false negatives.

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 that both the false-positive and false-negative probabilities are upper bounded.

The proposed Generalized Bloom Filter (GBF) is robust to the state of the filter because the maximum false-positive probability is completely independent of the number of bits set in the filter. In the standard Bloom Filter, an attacker may disrupt distributed applications by sending filters with all bits set to one and causing a false-positive rate of 100%, a technique we define as the *all-one attack*. With a GBF, however, the false-positive probability is upper bounded and this bound depends only on the number of hash functions used, k_0 and k_1. For instance, by using as few hash functions as $k_0 = 2$ and $k_1 = 3$, the maximum false-positive probability of a GBF is only 3.5%.

The tradeoff cost is the introduction of false negatives in the membership queries. Nevertheless, false negatives are also upper bounded and this bound only depends on k_0, k_1, and on the m/n ratio. As a result, the designer first arbitrates the maximum false-positive probability accepted by fixing k_0 and k_1. The maximum false-negative probability is then achieved by tuning the m/n value. For $k_0 = 2$, $k_1 = 3$ and $m/n = 256$, we have a maximum false-negative probability of 4.6%. The false-positive and false-negative probabilities can further be traded off to satisfy the requirements of the application.

Future work includes studying how compression can be used to further improve the performance of Generalized Bloom Filters in network transmissions. Compressing the GBF just before transmission and decompressing it at the other end host can help to reduce the actual transmitted filter size while maintaining its secure and space-efficient set representation.

In conclusion, we strongly believe that data structures that allow false positives and false negatives to be traded off are useful to fulfill the different application requirements available. In particular, we share the same line of thought as Bonomi *et al.* [22] 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 would like to thank Luis Henrique Costa, Deborah Estrin, and Lixia Zhang for their remarkable and valuable comments.

REFERENCES

[1] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 7, no. 13, pp. 442–426, July 1970.

[2] L. H. M. K. Costa, S. Fdida, and O. C. M. B. Duarte, "Incremental Service Deployment Using the Hop-By-Hop Multicast Routing Protocol," *IEEE/ACM Transactions on Networking*, vol. 14, no. 3, pp. 543–556, 2006.

[3] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer, "Single-Packet IP Traceback," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 721–734, Dec. 2002.

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

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

[6] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and 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, June 2003, pp. 236–246.

[7] S. C. Rhea and J. Kubiatowicz, "Probabilistic Location and Routing," in *Proceedings of the IEEE INFOCOM 2002 Conference*, New York, NY, USA, June 2002, pp. 1248–1257.

[8] T. D. Hodes, S. E. Czerwinski, B. Y. Zhao, A. D. Joseph, and R. H. Katz, "An Architecture for Secure Wide-Area Service Discovery," *Wireless Networks*, vol. 8, no. 2/3, pp. 213–230, Mar. 2002.

[9] R. P. Laufer, P. B. Velloso, D. de O. Cunha, I. M. Moraes, M. D. D. Bicudo, and O. C. M. B. Duarte, "A New IP Traceback System against Denial-of-Service Attacks," in *12th International Conference on Telecommunications*, Capetown, South Africa, May 2005.

[10] 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," Electrical Engineering Program, COPPE/UFRJ, Tech. Rep., Dec. 2006.

[11] D. J. Margoliash, "CSPELL - A Bloom Filter-Based Spelling Correction Program," Master's thesis, The University of Western Ontario, London, Ontario, Canada, June 1987, Department of Computer Science.

[12] M. Mitzenmacher, "Compressed Bloom Filters," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 604–612, Oct. 2002.

[13] M. V. Ramakrishna, "Practical Performance of Bloom Filters and Parallel Free-Text Searching," *Communications of the ACM*, vol. 32, no. 10, pp. 1237–1239, Oct. 1989.

[14] D. G. Severance and G. M. Lohman, "Differential Files: Their Application to the Maintanance of the Large Databases," *ACM Transactions on Database Systems*, vol. 1, no. 3, pp. 256–267, Sept. 1976.

[15] E. H. Spafford, "OPUS: Preventing Weak Password Choices," *Computers and Security*, vol. 11, no. 3, pp. 273–278, May 1992.

[16] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, "Deep Packet Inspection Using Bloom Filters," *IEEE Micro*, vol. 24, no. 1, pp. 52–61, Jan. 2004.

[17] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest Prefix Matching Using Bloom Filters," in *Proceedings of the ACM SIGCOMM'03 Conference*, Karlsruhe, Germany, Aug. 2003, pp. 201–212.

[18] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice," *ACM Transactions on Computer Systems*, vol. 21, no. 3, pp. 270–313, Aug. 2003.

[19] S. Cohen and Y. Matias, "Spectral Bloom Filters," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, California, USA, June 2003, pp. 241–252.

[20] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, "Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement," in *Proceedings of the IEEE INFOCOM 2004 Conference*, Hong Kong, China, Mar. 2004, pp. 1762–1773.

[21] K. Shanmugasundaram, H. Brönnimann, and N. Memon, "Payload Attribution via Hierarchical Bloom Filters," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, Washington DC, USA, Oct. 2004, pp. 31–41.

[22] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "Beyond Bloom Filters: From Approximate Membership Checks to Approximate State Machines," in *Proceedings of the ACM SIGCOMM'06 Conference*, Pisa, Italy, Nov. 2006, pp. 315–326.

[23] B. Donnet, B. Baynat, and T. Friedman, "Retouched Bloom Filters: Allowing Networked Applications to Trade Off Selected False Positives Against False Negatives," in *Proceedings of the 2nd Conference on Future Networking Technologies (CoNEXT'06)*, Lisboa, Portugal, Dec. 2006.

[24] B. Chazelle, J. Kilian, R. Rubinfeld, and 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, Jan. 2004, pp. 30–39.

[25] J.-K. Peir, S.-C. Lai, S.-L. Lu, J. Stark, and 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, June 2002, pp. 189–198.

Rafael P. Laufer received the B. Sc. and the M.Sc. degrees in Electrical Engineering from the Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil, in 2003 and 2005, respectively. During 2002, he was with Cisco Systems, Inc. as an intern. He is now working toward the Ph.D. degree in Computer Science at the University of California, Los Angeles (UCLA). His major research interests are in security, networking, and distributed systems.

Pedro B. Velloso was born in Rio de Janeiro, Brazil, on January 11, 1977. He received the Electronic Engineer degree and the M. Sc. degree in Electrical Engineering from Universidade Federal do Rio de Janeiro, Brazil. He is a D.Sc. student of Computer Science at LIP6 from Université Pierre et Marie Currie, France. His major interests are in wireless communications, trust models, voice over IP, and security.

Otto Carlos M. B. Duarte received the Electronic Engineer and M.Sc. degrees from the Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, in 1976 and 1981, respectively, and 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 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.