ARTICLE IN PRESS

Computer Networks xxx (2011) xxx–xxx

Contents lists available at ScienceDirect

Computer Networks

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

In-packet Bloom filters: Design and networking applications

Christian Esteve Rothenberg [a,*], Carlos Alberto Braz Macapuna [a], Maurício Ferreira Magalhães [a], Fábio Luciano Verdi [b], Alexander Wiesmaier [c]

[a] University of Campinas (Unicamp), School of Electrical and Computer Engineering, Brazil
[b] Federal University of São Carlos (UFSCar), Campus Sorocaba, Brazil
[c] Technische Universität Darmstadt/CASED, Germany

ARTICLE INFO

Article history:
Received 19 January 2010
Received in revised form 11 October 2010
Accepted 13 December 2010
Available online xxxx
Responsible Editor: K. Kant

Keywords:
Bloom filter
Algorithms
Distributed systems
Packet forwarding
Inter-networking

ABSTRACT

The Bloom filter (BF) is a well-known randomized data structure that answers set membership queries with some probability of false positives. In an attempt to solve many of the limitations of current network architectures, some recent proposals rely on including small BFs in packet headers for routing, security, accountability or other purposes that move application states into the packets themselves. In this paper, we consider the design of such in-packet Bloom filters (iBF). Our main contributions are exploring the design space and the evaluation of a series of extensions (1) to increase the practicality and performance of iBFs, (2) to enable false-negative-free element deletion, and (3) to provide security enhancements. In addition to the theoretical estimates, extensive simulations of the multiple design parameters and implementation alternatives validate the usefulness of the extensions, providing for enhanced and novel iBF networking applications.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Since the seminal survey work by Broder and Mitzenmacher [1], the Bloom filter (BF) [2] has increasingly become a fundamental data aggregation component to address performance and scalability issues of very diverse network applications, including overlay networks [3], data-centric wireless networks [4], traffic monitoring, and so on. With the caveat of one-sided errors, the use of Bloom filters turns memory and computational expensive operations into simple, resource-friendly set membership problems (e.g. "*is* $x \in S$?").

In this work, we focus on the subset of distributed networking applications that use packet-header-size Bloom filters to share some state (i.e. information set S) among network nodes. The specific state carried in the Bloom filter varies from application to application, ranging from secure credentials [5,6] to IP prefixes [7] and link identifiers [8], with the shared requirement of a fixed-size packet header data structure to efficiently verify set memberships. The commonality of recent inter-networking proposals [5–10] is relying on Bloom filters to move application state to the packets themselves in order to alleviate system bottlenecks (e.g. IP multicast [7], source routing overhead [8]), enable new in-network applications (e.g. security [5,6,9]) or stateless protocol designs [11].

We refer to the BF used in this type of applications as an in-packet Bloom filter (iBF). In a way, an iBF follows a reverse approach compared to a traditional standalone BF implementation: iBFs can be issued, queried, and modified by multiple network entities at packet processing time. These specific needs benefit from additional capabilities like element removals or security enhancements. Moreover, careful design considerations are required to deal with the potential effects of false positives, as every packet header bit counts and the actual performance of the distributed system is a key goal.

* Corresponding author.
E-mail addresses: chesteve@dca.fee.unicamp.br (C.E. Rothenberg), macapuna@dca.fee.unicamp.br (C.A.B. Macapuna), magalhaes@dca.fee.unicamp.br (M.F. Magalhães), verdi@ufscar.br (F.L. Verdi), wiesmaier@cased.de (A. Wiesmaier).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.12.005

In this paper, we address common limitations of naive iBF designs and provide a practical foundation for networking application designs requiring to solve set-membership problems on a packet basis (Section 3). Our main contribution consists of assembling and evaluating a series of practical extensions (i) to increase the system *performance*, (ii) to enable false-negative-free *element deletion*, and (iii) to provide *security-enhanced* constructs at wire speed (Section 4). Via extensive simulation work, we explore the rich design space and provide a thorough evaluation of the observed trade-offs (Section 5). Finally, we relate our contributions to previous work on Bloom filter designs and briefly discuss the applicability of the iBF extensions to existing applications (Section 6).

2. Networking applications

iBFs are well suited for applications where one might like to include a list of elements in every packet, but a complete list requires too much space. In these situations, a hash-based lossy representation, like a BF, can dramatically reduce space, maintaining a fixed header size, at the cost of introducing false positives when answering set-membership queries. From its original higher layer applications such as dictionaries, BFs have spanned their application domain down to hardware implementations, becoming a daily aid in network applications (e.g., routing table lookups, DPI, etc.) and future information-oriented networking proposals [12]. As a motivation to our work and to get some practical examples of iBF usages, we first briefly survey a series of networking applications with the common theme of using small BFs carried in packets.

2.1. Data path security

The credential-based data path architecture [5] proposes the following network router security feature. During the connection establishment phase, routers authorize a new traffic flow request and issue a set of credentials (aka capabilities) compactly represented as bit positions of a BF. The flow initiator constructs the credentials by including all the router signatures into an iBF. Each router along the path checks on packet arrival for presence of its credentials, i.e., the k bits resulting from hashing the packet 5-tuple IP flow identifier and the routers (secret) identity. Hence, unauthorized traffic and flow security violations can be probabilistically avoided in a stateless, per hop fashion. Using 128 bits only, for typical Internet path lengths, the iBF-based authorization token reduces the probability that attack traffic reaches its destination to a fraction of a percent.

2.2. Wireless sensor networks

A typical attack by compromised sensor nodes consists of injecting large quantities of bogus sensing reports, which, if undetected, are forwarded to the data collector(s). The statistical en-route filtering approach [6] proposes a detection method based on an iBF representation of the report generation (collection of keyed message authentications), that is verified probabilistically and dropped en-route in case of incorrectness. The iBF-based solution uses 64 bits only and is able to filter out 70% of the injected bogus reports within 5 hops, and up to 90% within 10 hops along the paths to the data sink.

2.3. IP traceback

The packet-marking IP traceback method proposed in [9] relies on iBFs to trace an attack back to its approximate source by analyzing a single packet. On packet arrival, routers insert their mark (IP mask) into the iBF, enabling a receiver to reconstruct probabilistically the packet path(s) by testing for iBF presence of neighboring router addresses.

2.4. Loop prevention

In Icarus [10], a small iBF is initialized with 0s and then filled as forwarding elements add their Bloomed interface mask (setting k bits to 1). If the `OR` operation does not change the iBF, then the packet might be looping and should be dropped. If the Bloom filter changes, the packet is definitely not looping.

2.5. IP multicast

Revisiting the case of IP multicast, the authors of [7] propose inserting an iBF above the IP header to represent domain-level paths of multicast packets. After discovering the dissemination tree of a specific multicast group, the source border router searches its inter-domain routing table to find the prefixes of the group members. It then builds an 800-bit shim header by inserting the path labels (AS_a: AS_b) of the dissemination tree into the iBF. Routers receiving the iBF check for presence of next hop autonomous systems and forward the packet accordingly.

2.6. Source routing & multicast

The LIPSIN [8] forwarding fabric leverages the idea of having interface identifiers in BF-form (m-bit Link ID with only k bits set to 1). A routing iBF can be constructed by ORing the different Link IDs representing a source route. Forwarding nodes maintain a small Link ID table whose entries are checked for presence in the iBF to take the forwarding decision. In a typical WAN topology and using 256-bit iBFs, multicast trees with around 40 links can be constructed to reach up to 24 users while maintaining the false positive rate (≈3%) and the resulting forwarding efficiency within reasonable performance levels.

3. Basic design

The basic notation of an iBF is equivalent to the standard BF, that is an array of length m, number of independent hash functions k, and inserted elements n. On insertion, the element is hashed to k hash values and the corresponding bit positions are set to 1 (see example in Fig. 1). On element check, if any of the bits determined by the hash outputs is 0, we can be sure that the element

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

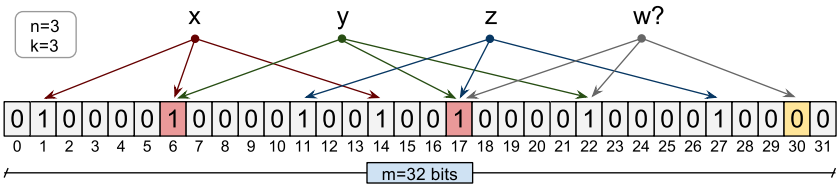


Fig. 1. Overview of the Bloom filter probabilistic data structure.

was not inserted (no false negative property). If all the k bits are set to 1, we have a probabilistic argument to believe that the element was actually inserted. The case of collisions to bits set by other elements causing a non-inserted element to return 'true' is referred to as a *false positive*. In the example of Fig. 1, a false positive for w would be returned if all three hashes would map to 1s.

For the sake of generality, we refer simply to *elements* as the objects carried in the iBF. Depending on the application, elements may take different forms such as interface names, IP addresses, certificates, and so on. False positives manifest themselves with different harmful effects such as bandwidth waste, security risks, computational overhead, etc. Thus, a system design goal is keeping false positives to a minimum.

3.1. False positive estimates

The *a priori* false positive estimate, *fpb*, is the expected false positive probability for a given set of parameters (m,n,k) *before* actually adding the elements. Let $p = 1-(1-1/m)^{kn}$ be the probability that a particular bit is set to 1. Then,

$$fpb = (1-(1-1/m)^{kn})^k. \tag{1}$$

The number k that minimizes the false positive probability can be obtained by setting the partial derivative of *fpb* with respect to k to 0. This is attained when $k = m/n \ln 2$, and is rounded to an integer to determine the optimal number of hash functions to be used [1].

While Eq. (1) has been extensively used and experimentally validated as a good approximation, for small values of m the actual false positive rate is larger. Recently, Bose et al. [13] have shown that *fpb* is actually only a lower bound, and a more accurate estimate can be obtained by formulating the problem as a balls-into-bins experiment:

$$p_{k,n,m} = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} i^k i! \binom{m}{i} \left\{ {kn \atop i} \right\}. \tag{2}$$

According to [13, Theorem 4], Eq. (2) can be lower- and upper-bounded as follows:

$$p^k < p_{k,n,m} < p^k \cdot \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}\right)\right). \tag{3}$$

Hence, the difference between the observed false positive rate and the theoretical estimates can be significant for small size BFs, a fact that we (and others) have empirically observed (see evaluation in Section 5.1.2). Thus, small iBFs are prone to more false positives than larger iBFs for equivalent m/n ratios.

Both Eqs. (1) and (2) do not involve knowing exactly how many bits are actually set to 1. A more accurate estimate can be given once we know the fill factor ρ; that is the *observed* fraction of bits that are actually set to 1 after elements have been inserted. We can define the posterior false positive estimate, *fpa*, as the expected false positive probability *after* inserting the elements:

$$fpa = \rho^k. \tag{4}$$

Finally, the *observed* false positive rate (*fpr*) can be obtained after testing for the presence of elements:

$$fpr = \frac{\text{Observed false positives}}{\text{Tested elements}}. \tag{5}$$

Note that the *fpr* is an experimental quantity obtained via simulation or system measurements and not a theoretical estimate. Hence, the *fpr* is the key performance indicator we want to measure in a real system, where every observed false positive will cause some form of degradation. Therefore, practitioners are less interested in the asymptotic bounds of the hash-based data structure and more concerned with the actual false positive rates, especially in the case of space-constrained iBFs.

3.2. Naming and basic operations

A nice property of hash-based data structures is that they do not depend on the form of the inserted elements. Independent of its size or representation, every element carried in the iBF contributes with at most k bits set to 1. In order to meet the line speed requirements of iBF operations, one design recommendation is to have the elements readily in a pre-computed BF-form (m-bit vector with k bits set to 1), avoiding thereby any hashing at packet processing time. Element *insertion* becomes a simple, parallelizable bitwise `OR` operation. Analogously, an iBF element *check* can be performed very efficiently in parallel via fast bitwise `AND` and `COMPARE` operations.

A BF-ready element name, also commonly referred to as element *footprint*, can be stored as an bit vector of size m or, for space efficiency, it can take a *sparse representation* including only the indexes of the k bit positions set to 1. In this case, each element entry requires only $k\log_2 m$ bits.

4. Extensions

In this section, we describe three useful extensions to basic in-packet Bloom filter designs in order to address the following practical issues:

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

(i) **Performance**: *Element Tags* exploit the notion of power of choices in combining hashing-based element names to select the best iBF according to some criteria, for instance, less false positives.
(ii) **Deletion**: *Deletable regions* introduce an additional header to code collision-free zones, enabling thereby safe (false-negative-free) element removals at an affordable packet header bit space.
(iii) **Security**: *Secure constructs* use packet-specific information and distributed time-based secrets to provide protection from iBF replay attacks and bit pattern analysis, preventing attackers from misusing iBFs or trying to infer the identities of the inserted elements.

4.1. Element tags

The concept of *element Tags* (eTags) is based on extending BF-compatible element naming with a set of equivalent footprint candidates. That is, instead of each element being identified with a single footprint, every element is associated with d alternative names, called eTags, uniformly computed by applying some system-wide mapping function (e.g., $k \cdot d$ hash functions). That allows us to construct iBFs that can be optimized in terms of the false positive rate and/or compliance with element-specific false positive avoidance strategies. Hence, for each element, there are d different eTags, where d is a system parameter that can vary depending on the application. As we see later, a practical value of d is in the range of multiples of 2 between 2 and 64.

We use the notion of *power of choices* [14] and take advantage of the random distribution of the bits set to 1 to select the iBF representation among the d candidates that leads to a better performance given a certain optimization goal (e.g., lower fill factor, avoidance of specific false positives). This way, we follow a similar approach to the Best-of-N method applied in [15], with the main differences of (1) a distributed application scenario where the value d is carried in the packet header, and (2) the best candidate selection criterion is not limited to the least amount of bits set but may include other optimization criteria (e.g., Section 5.2 bit deletability), including those that involve counting false positives against a training set (e.g. Section 4.1.2 *fpr*-based selection).

The caveats of this extension are, first, it requires more space to store element names, and second, the value d needs to be stored in the packet header as well, consuming bits that could be used for the iBF. However, knowing d at element query time is fundamental to avoid checking multiple element representations, which would traduce in potentially more false positives (cf. [14]). Upon packet arrival, the iBF and the corresponding eTag entries can be `AND`ed in parallel.

4.1.1. Generation of eTags

To achieve a near uniform distribution of 1s in the iBF, k independent hash functions per eTag are required. In general, k may be different for each eTag, allowing to adapt better to different fill factors and reducing the false positives of more sensitive elements. Using the *double hashing* technique [16] to compute the bits set to 1 in the d eTags, only two independent hash functions are required without any increase of the asymptotic false positive probability. That is, we rely on the result of Kirsch and Mitzenmacher [16] on linear combination of hash functions, where two independent hash functions and can be used to simulate i random hash functions of the form:

$$g_i(x) = [h_1(x) + i \cdot h_2(x)] \bmod m. \quad (6)$$

As long as $h_1(x)$ and $h_2(x)$ are system wide parameters, sharing $i = d \cdot k$ integers is only required to derive the eTags for any set of elements. For space efficiency, another optimization for the sparse representation of the candidates consists of defining the d eTags by combinations among $k + x$ iBF positions, i.e., $d = \binom{k+x}{k}$.

4.1.2. Candidate selection

Having "equivalent" iBF candidates enables to define a selection criteria based on some design-specific objectives. To address *performance* by reducing false positives, we can select the candidate iBF that presents the best posterior false positive estimate (*fpa-based selection*; Eq. (4)). If a reference test set is available to count false positives, the iBF choice can be done based on the lowest observed rate (*fpr-based selection*; Eq. (5)). Other types of selection policies can be specified to favor the candidate presenting less false positives for certain "system-critical" elements (*fp-element avoidance selection*).

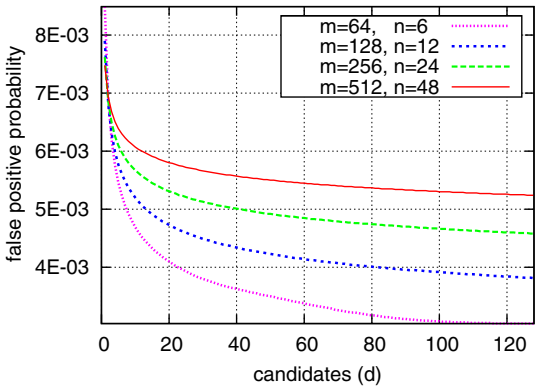
4.1.3. False positive improvement estimate

Following the same analysis as in [15], the potential gain in terms of false positive reduction due to selecting the iBF candidate with fewer 1s can be obtained by estimating the least number of bits set after d independent random variable experiments (see Appendix A for the mathematical details). Fig. 2 shows the expected gains when using the *fpa-based selection* after generating d candidate iBF for a given element set. With a few dozen candidates, one can expect a factor 2 improvement in the observed *fpr* when selecting the candidate with fewer ones. Note that the four iBF configurations plotted in Fig. 2 have the same m/n ratio. In line with the theoretical predictions [13], smaller bit vectors are subject to slightly larger false positive probabilities. However, as shown in Fig. 2(b), the *fpr* improvement factor of smaller iBFs due to the d-eTag extension is larger. Hence, especially for small iBFs, computing d candidates can highly improve the false positive behavior, a fact that we have validated experimentally in Section 5.

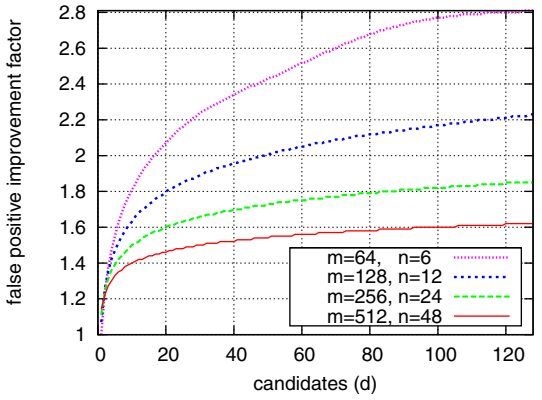
4.2. Deletable regions

Under some circumstances, a desirable property of iBFs is to enable element deletion as the iBF packet is processed along network nodes. For instance, this is the case if some inserted elements are to be processed only once (e.g., a hop within a source route), or, if bit space is required to add more elements upfront. Unfortunately, due to its compression nature, bit collisions hamper naive element removals unless we allow introducing false negatives into the

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005



(a) A priori false positive estimate of the iBF candidate with the lowest fill factor.



(b) Potential false positive improvement when being able to choose among d iBFs.

Fig. 2. False positive probability gains of the power of choices extension.

system. To overcome this limitation (with high probability), so-called counting Bloom filters (CBF) [17] were proposed to expand each bit position to a cell of c bits. In a CBF, each bit vector cell acts as a counter, increased on element insertion and decreased on element removal. As long as there is no counter overflow, deletions are safe from false negatives. The main caveat is the c times larger space requirement, a prohibitive price for the tiny iBFs under consideration.

The key idea of the deletable region extension is to keep track of where the collisions occur at element insertion time. By using the property that bits set to 1 by just one element (collision-free bits) are safely deletable, the proposed extension consists of encoding the deletable regions as part of the iBF header. Then, an element can be effectively removed if at least one of its bits can be deleted.

Encoding the deletable region information should consume a minimum of bits from the allocated iBF space. A straightforward coding scheme is to divide the iBF bit vector into r regions of m'/r bits each, where m' is the original m minus the extension header bits. As shown in Fig. 3, this extension uses r bits to code with 0 a collision-free region and with 1 a non-deletable region. The probability of element deletion, i.e., the chances of an element having at least one bit in a collision-free region, can be approximated to (see Appendix B for the mathematical details):

$$p_d = (1 - (1 - p_c)^{m'/r})^k. \quad (7)$$

Fig. 4(a) plots p_d against the number of regions r and confirms the intuition that increasing r results in a larger proportion of elements being deletable. As more elements are inserted into the iBF, the number of collisions increases and the deletion capabilities (i.e., bits in collision-free regions) are reduced (see Fig. 4(b)). As a consequence, the target element deletion probability p_d and the number of regions r establish a practical limitation on the capacity n_{max} of a deletable iBF.

Fig. 4(b) plots p_d against the filter density m/n for different memory to regions ratios m/r. As expected, increasing r results in a larger portion of deletable elements. As more elements are inserted (lower m/n and more collisions), the deletion capabilities are reduced. Hence, the parameter r can be chosen by defining a target element deletion probability p_d and estimating the upper bound of the set cardinality n. For instance, allocating only 5 % of the available bits ($m/r = 20$) to code the collision bitmap, we can expect to remove around 90 % of the elements when the bits per element ratio m/n is around 16.

From a performance perspective, enabling deletions comes at the cost of r bits from the iBF bit space. However, removing already processed elements decreases the fill factor and consequently reduces the probability of false positives upfront. Later in Section 5.2 we explore the trade-offs between the overhead of coding the deletable regions, the impact on the *fpr*, and the implications of the candidate selection criteria.

4.3. Secure constructs

The hashing nature of iBFs provides some inherent security properties to obscure the identities of the inserted elements from an observer or attacker. However, there are

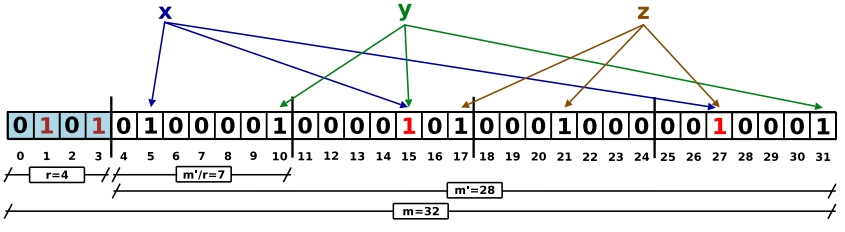
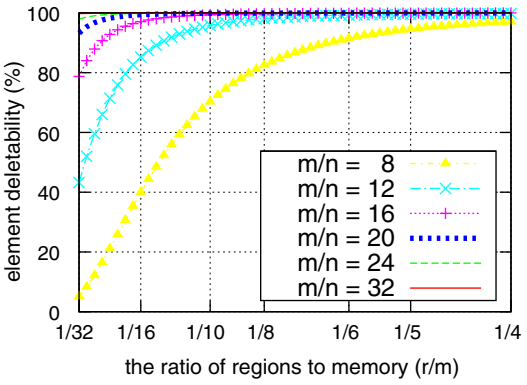
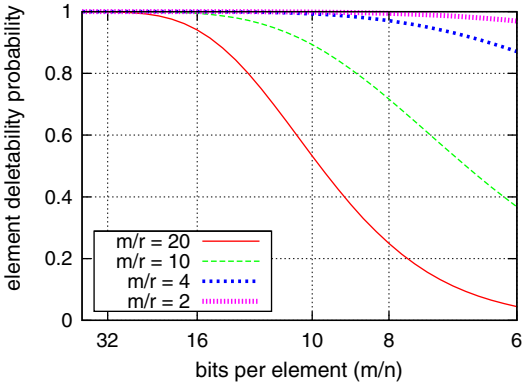


Fig. 3. An example of the DlBF with $m = 32$, $k = 3$ and $r = 4$, representing the set x,y,z. The 1s in the first r bits indicate that a collision happen in the corresponding region and bits therein cannot be deleted. Since each element has at least one bit in a collision-free zone, all of them are deletable.

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005



(a) Deletability as function of number of regions.



(b) Deletability as function of inserted elements.

Fig. 4. Element deletability probability ($m = 256$).

a series of cases where improved security means are desirable. For instance, an attacker is able to infer, with some probability, whether two packets contain an overlapping set of elements by simply inspecting the bits set to 1 in the iBFs. In another case, an attacker may wait and collect a large sample of iBFs to infer some common patterns of the inserted elements. In any case, if the attacker has knowledge of the complete element space (and the eTags generation scheme), she can certainly try a dictionary attack by testing for presence of every element and obtain a probabilistic answer to what elements are carried in a given iBF. A similar problem has been studied in [18] to secure standalone BFs representing a summary of documents by using keyed hash functions. Our solution follows the same approach i.e. obscuring the resulting bit patterns in the filter by using additional inputs to the hashes. However, our attention is focused to the specifics of distributed, line-speed iBF operations.

The main idea to improve the security is to bind the iBF element insertion to (1) an invariant of the packet or flow (e.g., IP 5-tuple, packet payload, etc.), and (2) system-wide time-based secret keys. Basically, the inserted elements become packet- and time-specific. Hence, an iBF gets expirable and meaningful only if used with the specific packet (or authorized packet flow), avoiding the risk of an iBF replay attack, where the iBF is placed on a different packet.

4.3.1. Binding to packet contents

We strive to provide a lightweight, bit mixing function $O = F(K,I)$ to make an element name K dependent on additional in-packet information I. For this extension, an element name K is an m-bit hash of the element and not the eTag representation with only k bits set to 1. The function F must be fast enough to be done at packet processing time over the complete set of elements to be queried by a node processing the iBF. The output O is the k bit positions to be set/checked in the iBF. Using cryptographic hash functions (e.g., MD5, SHA1) for F becomes unpractical if we want to avoid multiple (one per element) cycle-intense hashing per packet.

Algorithm 1: Secure iBF element set/check algorithm

Input: element name K, packet-specific id. I, number of choices d
Output: k bit positions to be set/checked
1. Let $O = k \otimes I$
2. Divide O into k segments of m/k bits: $O_1, O_2, \ldots, O_k$
3. Divide each O_j into a $clog_2 m$ bit matrix: $Oj_1, Oj_2, \ldots, Oj_c$ where $c = \lceil \frac{m}{klog_2 m} \rceil$
4. **for each** $O_j \in [O_1, \ldots, O_k]$ **do**
   Set/check bit position i in the iBF where:
   $i = O_{j1} \otimes O_{j2} \otimes \cdots \otimes O_{jc}$
   $i \ll d$
end

As an example resource-efficient implementation of F, we propose the lightweight Algorithm 1 to mix each element K with a fixed bit string I. Taking I as an input, the algorithm runs in parallel on each element K and returns the k bit positions in the iBF to be set or checked. After an initial bitwise XOR operation (Step 1), the output O is divided into k segments of m/k bits (Step 2). To build the folding matrix in Step 3, each segment is transformed into a matrix of $clog_2 m$ bits.[1] For instance, with $m = 256$ and $k = 4$, each segment O_k would be a 64-bit bit vector transformed into a 8×8 matrix. Finally, each of the k output values is computed by XORing the rows of each matrix into a $\log_2 m$ bit value that returns the bit position to be set/checked (Step 4). The d-bit shifting enables the power of choices.

We are faced with the classic trade-off between security and performance. An heuristic evaluation suggests that the proposed F provides a good balance between performance and security. First, F involves only bit shifting and XOR operations that can be done in a few clock cycles in parallel for every K. Second, the k bit positions depend on all the bits, within an m/k bit segment, from the inputs I and K. The security of F depends on how well I and K are mixed. For security sensitive applications, the XOR operation in Step 1 should be replaced with a more secure transformation $P(K,I)$ i.e., using lightweight hash functions or line-speed stream ciphers (see e.g. [19]). The final choice of F

[1] Note that depending on the values of m and k, some padding bits (e.g., from within segments) may be required to complete the matrix.

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

should take the application specifics into account (e.g., nature of K, computation of I per-packet) and the target security level.

4.3.2. Time-based keyed hashing

A more elaborate security extension consists of using a *keyed* element name construction, and change the secret key $S(t)$ regularly. We can define $S(t)$ as the output of a pseudo random function $S_i = F(seed, t_i)$, where *seed* is the previous value and t a time-based input. Then, we can include the current S value in the algorithm for element check/insertion e.g., $O = F(K, I, S(t))$. Thereby, we obtain a periodically updated, shared secret between iBF issuers and iBF processing entities, with the benefit that an iBF cannot be re-utilized after a certain period of time or after an explicit re-keying request. Moreover, by accepting S_i and S_{i-1} the system requires only loose synchronization similar to commercial time-coupled token generators. At the cost of initial synchronization efforts and computational overhead, this method provides an effective means to make iBF applications secure (e.g., forwarding availability [20,21]).

4.4. Density factor

Finally, a basic security measure for iBFs, also proposed in [5], is to limit the percentage of 1s in the iBF to 50%–75%. A *density factor* ρ_{max} can safely be set to $k \cdot n_{max}/m$, as each legitimate element contributes with at most k bits. Then, the probability of an attacker guessing a bit combination that causes a single false positive can be upper bounded by ρ_{max}^k.

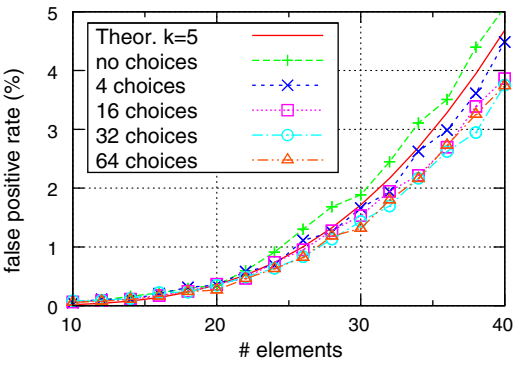
5. Practical evaluation

We now turn our attention to the practical behavior of the iBF in function of the multiple design parameters and carry out extensive simulation work to validate the usefulness of the three extensions under consideration. For these purposes, we use randomly generated bit strings as input elements and the double hashing technique using SHA1 and MD5. The section concludes exploring the potential impact of different types of iBF elements (flat labels, IP addresses, dictionary entries) and the hash function implementation choice.

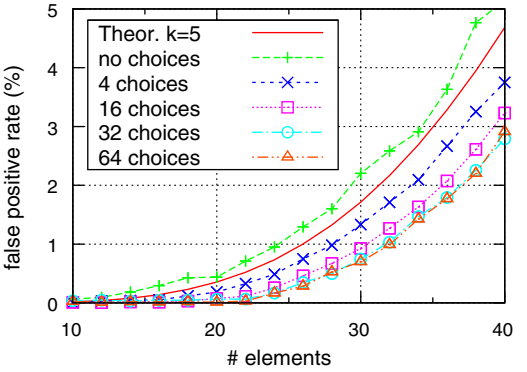
5.1. Element tags

We are interested in evaluating the gains of the power of choices that underpins the element Tag extension (Section 4.1), where any element set can be equivalently represented by d different iBFs, different in their bit distribution but equivalent with regard to the carried element identities. We first explore the case where $k = 5$ and then the impact of using a distribution around 5 for candidate naming.[2]

[2] We choose $k = 5$ to have a probabilistically sufficient footprint space for the eTags ($m!/(m-k)! \approx 10^{12}$ with $m = 256$) when targeting an m/n of about 8 bits per element.



(a) Best fill rate candidate



(b) Best observed fpr

Fig. 5. Power of choice gains ($m = 256$, $k = 5$).

5.1.1. Power of choices (d)

We run the simulations varying d from 2 to 64 and updating m accordingly to reflect the overhead of including the value d in the packet header. Fig. 5 compares the observed *fpr* for different values of d. In accordance with the theoretical predictions (Section 4.1.3), increasing d and choosing the candidate iBF just by observing its fill factor after construction (Fig. 5(a)) leads to better performing iBFs. In the region where the iBF is more filled (30–40 elements), the observed *fpr* drops between 30% and 50% when 16 or more candidate iBFs are available. Another interpretation is that for a maximal target *fpr* we can now insert more elements. As expected, the performance gain is more significant if we consider the best performing iBF after testing for false positives. Observing Fig. 5(b), the number of false positives is approximately halved when comparing the best iBF among 16 or more against a standard 256-bit iBF.

We also note that the observed *fpr* is slightly larger than the commonly assumed theoretical estimate (Eq. (1)), confirming thus the findings (Eq. (4)) by [13]. As shown in Table 1, this difference is more noticeable for smaller m, becoming negligible for m larger than 1024.

5.1.2. Distribution of the number of hash functions (k)

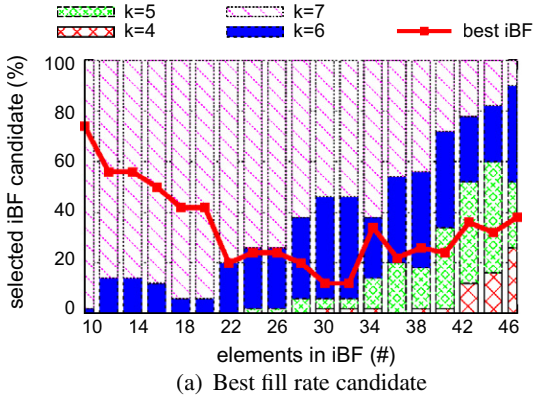
Now, we explore allowing a different number of bits k per candidate. For instance, with $d = 8$ the distribution of

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

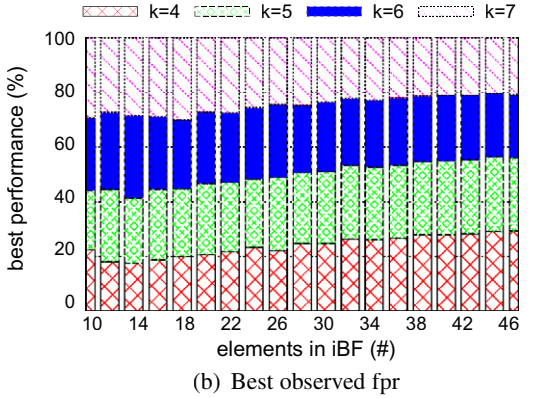
Table 1
Observed *fpr* for iBFs with 16 eTag choices.

m	*n*	Std. (%)		fpa-opt. (%)		fpr-opt. (%)	
		Th.	*fpr*	k_{cte}	k_{dst}	k_{cte}	k_{dst}
128	6	0.04	0.16	0.14	0.19	0.04	0.05
	12	0.75	1.12	0.88	0.86	0.37	0.32
	18	3.33	4.39	2.80	3.10	2.18	2.37
256	12	0.04	0.09	0.08	0.08	0.01	0.03
	24	0.74	0.95	0.74	0.71	0.26	0.30
	36	3.31	3.63	2.69	2.75	2.07	2.15
512	24	0.04	0.08	0.07	0.04	0.01	0.01
	48	0.74	0.83	0.64	0.64	0.22	0.25
	72	3.29	3.46	2.87	3.05	2.09	2.21

k among the candidates could be {4,4,5,5,6,6,7,7}. Intuitively, this naming scheme adapts better to the final number of elements in the iBF (as k_d closer to $k_{opt} = m/n \ln(2)$). The *fpa-based selection* criterion (Section 4.1) is now choosing the candidate with the lowest estimate $min\{\rho_0^{k_0}, \ldots, \rho_d^{k_d}\}$. Fig. 6(a) shows the distribution of the selected 256-bit iBFs for the case of $d = 16$ and k evenly distributed between 4 and 7. The line shows the percentage of times that the selected iBF actually yielded the best performance among the candidates. Disregarding the scenarios with fewer elements, the *fpa-based selection* strategy succeeded to choose the optimal candidate in about 30% of the times.



(a) Best fill rate candidate



(b) Best observed fpr

Fig. 6. Distribution of iBF candidates for different number of hash functions k. ($d = 16$, $m = 256$).

Fig. 6(b) shows the percentile distribution of the best performing iBF after *fpr* testing. As expected, in more filled iBFs scenarios, setting less bits per element is beneficial. However, the differences are relatively small. As shown in Table 1, the observed *fpr* in the case of $k_{const.} = 5$ is practically equivalent (if not slightly better) to the case where k is distributed. We can also observe what the theory in Section 2 predicts with regard to smaller iBFs: (i) inferior *fpr* performance for the same m/n ratio, and (ii) larger potential to benefit from the power of choices extension.

5.1.3. Discussion

Based on our experimental evaluation, having more than 32 candidates per element is not compelling in terms of additional proportional *fpr* benefits beyond approximately a factor 2 depending on the specific parameters. The results are consistent with the theoretical estimates in Section 4.1.3. However, if the system design choice is based on selection criteria optimized for the non occurrence of specific false positives (i.e. element-avoidance Section 4.1), increasing the number of choices d allows complying to a larger set of false positive avoidance policies. The practical limitations would be how much space the application designer is willing to pay to store the candidate element representations in the nodes and code the index d in the packets.

5.2. Deletion

We explore two important aspects of the deletable regions extension. First, from a *qualitative* point of view we examine the actual capabilities to successfully delete elements for different m/n ratios, number of *regions* r and choices d. Second, we evaluate the *quantitative* gains in terms of false positive reduction after element bits are deleted. Obviously, both aspects are related and intertwined with the ability to choose among candidate iBF representations to favor the deletion capabilities. Now, the application can choose the iBF candidate with the most number of bits set in collision free-zones, increasing thus the *bit deletability*. Alternatively, one may want to favor the *element deletability*, recalling that removing a single element bit is traduced into a practical deletion of the element.

Using our basic coding scheme (Section 4.2), we consume one bit per region to code whether collision

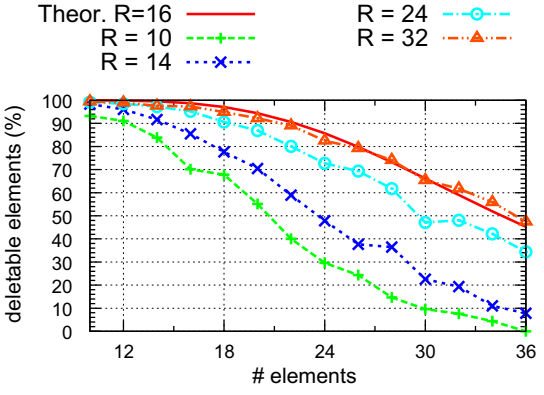
Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

happened and deletion is prohibited or not. Thus, the bits available for iBF construction are reduced to $m' = m - \log_2 d - r$.

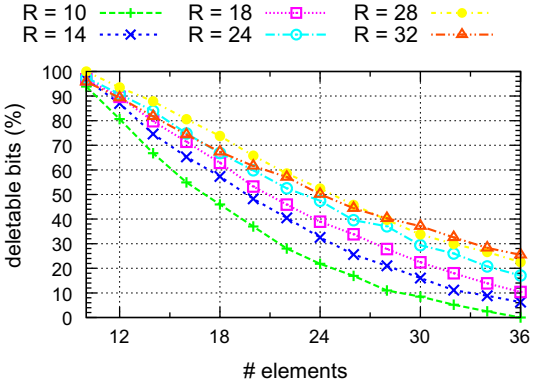
On each experiment round, we randomly select n elements from a pool of 1 million unique bit strings, and insert them updating the r bitmap accordingly. We then try to remove every inserted element and measure the quality and quantity of the deletion capabilities.

5.2.1. Quality: how many elements can be removed in practice?

Fig. 7(a) plots the average percentage of elements that could be deleted. As expected, partitioning the iBF into more regions results in a larger fraction of elements (and bits) being deletable. For instance, in the example of a 256-bit iBF with 32 regions (Fig. 7), when 24 elements are inserted, we are able to delete an average of more than 80% of the elements by safely removing around 50% of the bits (Fig. 7(b)). Playing with the candidate choices, we can enhance the bit (Fig. 8(a)) and element (Fig. 8(b)) deletability considerably. The actual deletability rates are lower than expected by theory (Fig. 4) but behave as predicted by the mathematical model of the element deletability probability (Eq. (7)). This divergence can be explained by the theoretical assumptions on random bit distributions

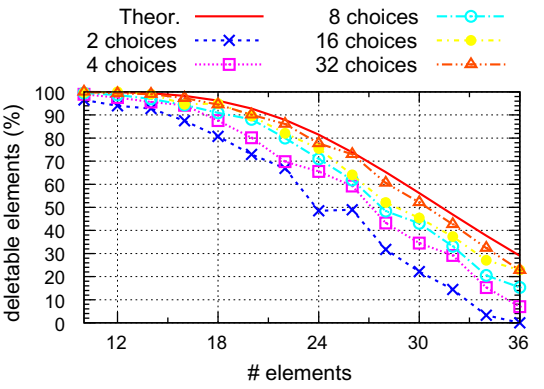


(a) Deletability - elements

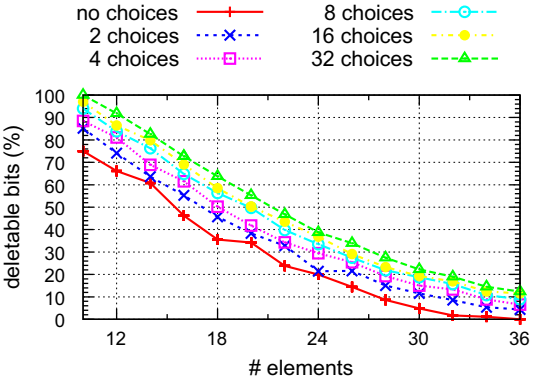


(b) Deletability - bits (d=16)

Fig. 7. Deletability as function of r ($m = 256$).



(a) Deletability - elements



(b) Deletability - bits

Fig. 8. Deletability as $f(d)$. $m = 256$, $r = 16$.

and the actual behaviour of hash functions, especially in small size bit vectors.

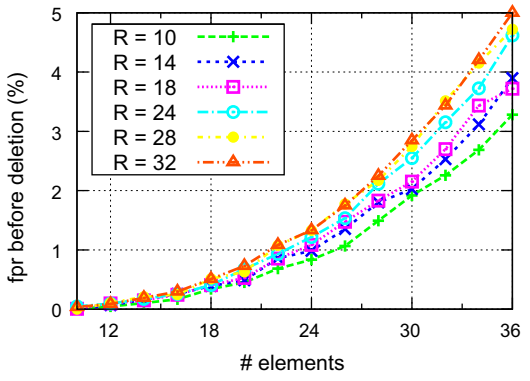
5.2.2. Quantity: what are the false positive rate gains due to bit deletability?

On the one hand, we have the potential gains of removing bits from collision-free zones. On the other, the cost of (1) coding the deletable regions (r bits), and (2) having more filled iBFs due to the rarefication of colliding bits. While Fig. 9(a) shows the price of having to code more regions (*fpr* before deleting elements), Fig. 9(b) illustrates the potential gains of removing every deletable bit. If we average the *fpr* before and after elements are deleted, the iBF performance appears equivalent to the *fpr* of a standard non-deletable m-bit iBF. In comparison, a counting BF with 2 bits per cell[3] would behave like an iBF of size $m/2$, which would have its element capacity prohibitively constrained.

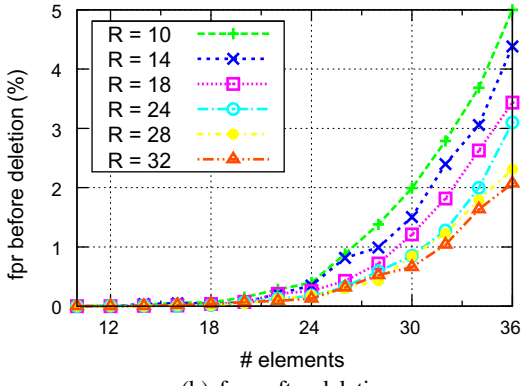
Analyzing the impact of the *power of choices*, Fig. 10 shows that choosing the best deletable iBF candidate causes the colliding bits to "thin out" (greater ρ), yielding a higher *fpr* before deletion (Fig. 10(a)) and a smaller *fpr* after elements are removed (Fig. 10(b)).

[3] Using the power of choices, we could have with very high probability a candidate that does not exceed the counter value of 3, avoiding false negatives as long as no new additions are considered.

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

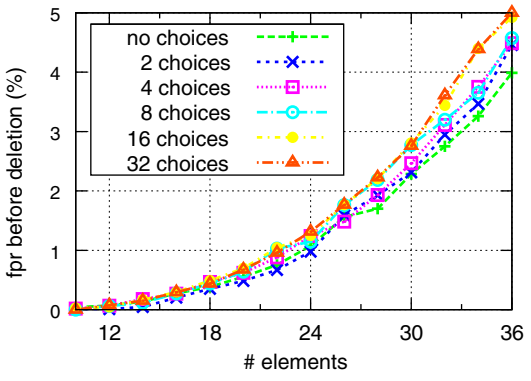


(a) fpr - before deletion

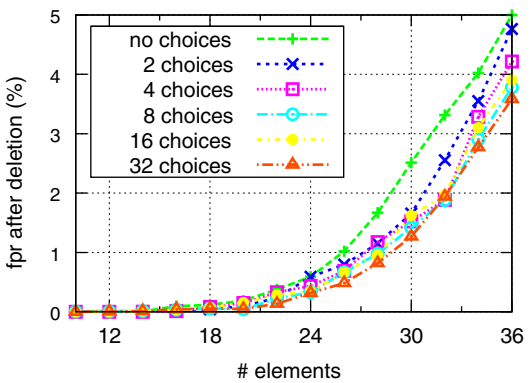


(b) fpr - after deletion

Fig. 9. False positives in function of r ($m = 256$, $d = 16$).



(a) fpr - before deletion



(b) fpr - after deletion

Fig. 10. False positives as $f(d)$. ($m = 256$, $r = 16$).

5.2.3. Discussion

There is a tussle between having a smaller fill factor ρ, with more collisions at construction time reducing the *fpa*, and the deletability extension that benefits from fewer collisions. Deletability may be a key property for some system designs, for instance, whenever an element in the iBF should be processed only once and then be removed, or when space is needed to add new elements on the fly. One key property is that just by inspecting the bitmap r, any node can safely (without introducing false negatives) remove an element from the iBF. A more detailed evaluation should consider the specific application dynamics into consideration, i.e., the nature and frequency of deletions/insertions at runtime.

From a *fpr* performance perspective, the cost of coding the deletable regions is only a slight increase in the *fpr* due to r being only a small fraction of m. However, reducing m seems to hinder the average *fpr* gains due to bit deletions upfront. Nonetheless, especially for space-restricted iBFs, the proposed extension is a far more attractive approach to enable (probabilistic) deletions than alternative solutions based on counting BFs. An open question is whether there is a better coding scheme for the deletable regions, for instance, using error correcting codes. Finally, the power of choices again proved to be a very handy technique to deal with the probabilistic nature of hash-based data structures, enabling candidate selection for different criteria like better *fpr* or certain element/bit deletability.

5.3. Security

Besides fast computation, the main requirements for the security extension are that (i) the random distribution of the iBF bits is conserved, and (ii) given a collection of packets I and the securely constructed iBFs, one cannot easily reveal information about the inserted elements (K). More generally, given a set of (I,*iBF*) pairs, it must be at best very expensive to retrieve information about the identities of K.

We first measured the randomness of the secure iBF construction outputs from Algorithm 1 by fixing a set of 20 elements and changing the per-packet 256-bit randomly generated I value on each experiment run. Table 2 gathers the average results of 100 experiments with 1000 runs per experiment. The observed distribution of outputs within an experiment, measured as the Hamming distance between output bit vectors (BV), was very close to the mean value of $m/2$ bits (128) with a small standard deviation.[4] The observed average number of bits set and their distribution were comparable to standard iBF constructs. Additionally, we analyzed whether the 20 most frequent bit positions set in secure iBFs corresponded to bits set in

[4] In future work we will extend these results and the hashing techniques evaluation of Section 5.4 with standard randomness tests such as those included in the Diehard suite (http://www.stat.fsu.edu/pub/diehard).

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

Table 2
Evaluation of the secure iBF algorithm ($m = 256$, $k = 4$, $n = 20$). Avg. (Stdev) after 1000 runs.

	Sec. iBF	Plain iBF	Random BV
Hamming dist.	127.94 (8.06)	0	127.95 (8.03)
# Bits set	96.27 (3.20)	96.29 (–)	127.97 (7.97)
Correlation	0.371		–

plain iBFs. We defined the *correlation factor* as the fraction of matches and obtained a value of 0,371, which is close to the probability of randomly guessing bits in a 256-bit iBF with $k = 4$ and $n = 20$ elements ($Pr \approx 96/256 \approx 0{,}37$).

The results indicate that, assuming a random packet identifier I, first, no actual patterns can be inferred from the securely inserted elements, and second, the random bit distribution of an iBF is conserved when using the proposed algorithm. However, we recognize the limitations of Algorithm 1. For instance, if provable protection against more elaborated attacks is required, then, a more secure and computationally expensive bit mixing procedure (Step 1 in Algorithm 1) should be considered, in addition to a time-based shared secret as suggested in Section 4.3.

5.4. Hashing technique

Finally, we investigate the impacts of the hash function implementation choice and the nature of the input elements in small size iBFs. There are two factors that determine the "quality" of the bit distribution and consequently may impact the observed *fpr*: (1) the *input* bit string, and (2) the *implementation* of the hash function.

5.4.1. Input data sets

Instead of considering elements as simple random bit strings, we now explore three types of elements that cover typical inputs of iBF applications:

- **32-bit IP addresses:** Nearly 9M IP addresses were generated by expanding the subnet values of IP prefixes advertised in the CAIDA database.[5] In addition, private IP addresses (10.0.0.0/16,192.168.0.0/16) were also used in the experiments.
- **256-bit random labels:** A set of 3M random labels was generated constructing each 256-bit label by picking randomly 64 hex characters and checking for uniqueness.
- **Variable-bit dictionary words:** A set formed by 98.568 entries of the American dictionary.[6]

5.4.2. Hash function choice

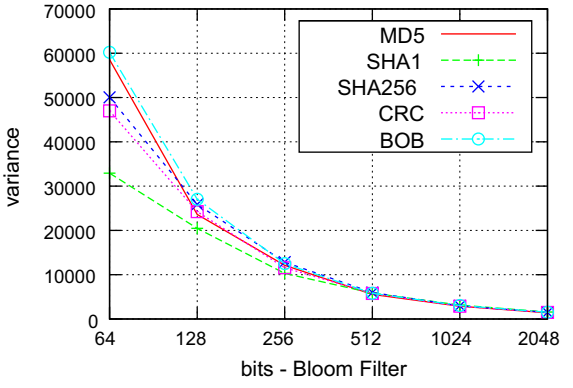
We chose 3 commonly used cryptographic hash functions (MD5, SHA1 and SHA256) and 2 general purpose hash functions (CRC32 and BOB).[7]

[5] ftp.ripe.net/ripe/stats/delegated-ripencc-20090308.
[6] /usr/share/dict/american-english.
[7] Related work has investigated the properties of 25 popular hash functions, pointing to BOB as a fast alternative that yields excellent randomized outputs for network applications [22]. Although MD5 and SHA1 are considered broken due to the recent discovery of collisions, they are perfectly valid for our randomness purposes.

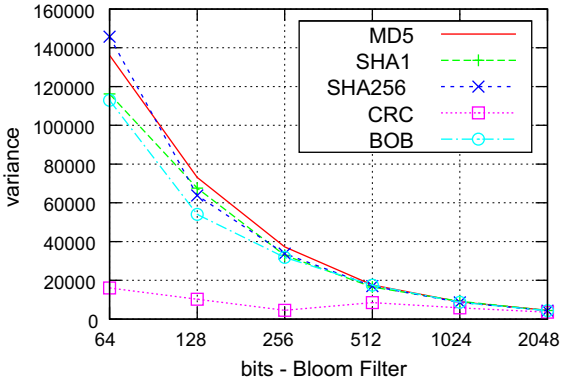
Table 3
Observed *fpr* in 256-bit iBF using double hashing with SHA1 & MD5 and with 8-bit segments of CRC32. Avg. (StdDev); 1000 tests.

n	DoubleHash	IP	Random	Dict.
16	SHA1& MD5	0.340 (0.035)	0.338 (0.032)	0.328 (0.034)
	CRC32 segm.	0.345 (0.037)	0.349 (0.034)	0.338 (0.034)
32	SHA1& MD5	2.568 (0.436)	2.576 (0.449)	2.519 (0.385)
	CRC32 segm.	2.541 (0.418)	2.532 (0.403)	2.570 (0.444)

The observed *fpr* (Table 3) imply that, on average, the input type does not affect the iBF performance. Fig. 11 plots the observed normalized *sample variance* for different bit vector sizes (m). For lower m values the variances show a larger difference and start converging for $m > 512$. CRC presents the best output distribution when dealing with IP addresses as inputs. This may be explained by the 32-bit match of inputs and outputs. In general, the functions exhibit similar behavior, leading to the conclusion that all 5 hash functions can be used independently from the nature of the elements. This result experimentally confirms, also in the case of small m values, the observation by Mitzenmacher and Vadhan [23] that given a certain degree of randomness in the input, simple hash functions work well in practice.



(a) Random labels - 3M #



(b) IP - 8,957,184 #

Fig. 11. Normalized bit distribution variance of hash outputs.

5.4.3. Hash segmentation technique

For the purposes of iBF construction, there is a waste of hash output bits due to the *mod m* residual restrictions. Hence, we want to know whether we can divide the output of a hash function into $\log_2 m$ segments and use each segment as an independent hash value. We compare the bit distribution and *fpr* performance of iBFs constructed using the double hashing technique with MD5 and SHA1 against iBFs generated with CRC32 segments as $h_i(x)$. The differences of the observed *fpr* (Table 3) are negligible, which suggests that this hashing technique may be practical. Hence, we can reduce the two independent hash function requirement of the double hashing technique to a single hash computation based on e.g., CRC32 or BOB. This result can be applied to iBF networking applications with on-line element hashing instead of pre-computed element names. Moreover, this efficient hash segmentation technique may be useful in other multiple-hashing-based data structures (e.g., d-left hash tables) that require hashing on a packet basis.

6. Related work

Although multiple variants of Bloom filter designs and applications have been proposed in the last years (e.g., *Bloomier, dynamic, spectral, adaptive, retouched*, etc.), to the best of our knowledge, none of the previous work focuses on the particular requirements of distributed networking applications using small Bloom filters in packet headers.

Prior work on improved Bloom filters include the Power of Two Choices filter [14] and the Partitioned Hashing [24], which rely on the power of choices at hashing time to improve the performance of BFs. False positives are reduced in [24] by a careful choice of the group of hash functions that are well-matched to the input elements. However, this scheme is not practical in distributed, highly dynamic environments. The main idea of [14] is to reduce the number of 1s by choosing the "best" set of hash functions. Besides our in-packet-header scope, our approach differs in that we include the information of which group of hash functions was used (d value) in the packet itself, avoiding thereby the caveat of checking multiple sets. On the other hand, we need to stick to one set of hash functions for all elements in the BF, whereas in [14] the optimal group of hash functions can be chosen on an element basis. To our benefit, due to the reduced bit vector scenario, we are able to select an optimal BF after evaluating all d candidates, which leads to improved performance even in very dense BF settings (small m/n ratios).

Regarding the extension to choose the best candidate filter, the Best-of-N method [15] only considers a standalone application where the best BF selection is based on the least dense filter constructed using the optimal number of hash functions. In contrast, our distributed iBF applications include candidates with different amount of bits set, as the maximum set cardinality may be unknown a priori. An optimized candidate iBF selection is possible whenever the iBF application is able to test for presence of elements that are known to be queried upfront. Moreover, selection criteria may be beyond reducing *fpr*, for instance benefiting the deletion of elements or avoiding specific false positives.

The closest BF design innovations to support deletions, other than counting BFs or d-left fingerprint hash tables [17], are the Variable-length Signatures (VBF) [25]. Similarly based on resetting at least one bit from element signatures, the main caveat is being prone to false negatives. In contrast, our deletable regions do not introduce false negatives at the cost of providing only probabilistic element deletions.

Security and privacy preserving extensions for standalone BFs have been previously proposed in different contexts (e.g., [18,26]). The novelty of our application resides in taking distributed systems and data packets specifics into consideration (e.g., flow-identifier, time-based loose synchronization of distributed secrets).

7. Relevance and extensions in practice

Our work on iBFs is mostly an outcome from our research on compact packet forwarding mechanisms. The idea of element Tags has its roots in the work on a link identifier based forwarding fabric [8], rendering the system more useful (network policy compliance, loop avoidance, security) and efficient (*fpr* control, larger multicast groups). Recently, we applied the notion of power of choices in the design of scalable data center forwarding services [27,28] based on 96-bit iBF encoding of valiant load balanced fat tree network paths between virtual machines hosted in rack servers.

In general, the element tag extensions can be applied to similar use cases of compact source routing. For instance, in the IP multicast proposal [7], having multiple choices would reduce false positives and enable compliance to inter-domain AS policies in the case of false positives. When applied to the credentials-based architecture proposed in [5], multiple candidates may allow iBFs to transverse larger paths before reaching the maximum density. Additionally, the security extension may provide extra protection from an en-route attacker spoofing the source IP address and re-using the flow credentials for unauthorized traffic.

In the field of secure packet forwarding mechanisms, we contributed to the development of self-routing capabilities for DDoS protection in Bloom filter based forwarding services [20]. In addition, recent work has validated the effectiveness of loop prevention with a new extension based on performing per-hop bit permutations [29]. A joint application of these iBF algorithmic techniques aims at solving forwarding anomalies of naive approaches to iBF based networking. Related work has explored the application of secure iBF forwarding methods to provide fast host mobility [30], scalable multicast VPN services in GMPLS-enabled networks [31], and an edge-controlled approach to stateless inter-domain 'bloomcasting' [21].

Last but not least, we are looking for new use cases for our deletable Bloom filter design [32]. Due to its space efficiency and the tunable probability of safe bit removals, the deletable regions extension may have interesting applications beyond the scope of iBFs. Similarly, the hash segmentation

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

technique appears useful to lower the burden on any system requiring multiple expensive hash computations.

8. Conclusions

This paper explores an exciting front in the Bloom filter research space, namely the special category of small Bloom filters carried in packet headers. Using iBFs is an appealing approach for networking application designers choosing to move application state to the packets themselves. At the expense of some false positives, fixed-size iBFs are amenable to hardware and present a way for new networking applications.

We studied the design space of iBFs in depth and evaluated new ways to enrich iBF-based networking applications without sacrificing the Bloom filter simplicity. First, the power of choices extension shows to be a very powerful and handy technique to deal with the probabilistic nature of hash-based data structures, providing finer control over false positives and enabling compliance to system policies and design optimization goals. Second, the space-efficient element deletion technique provides an important (probabilistic) capability without the overhead of existing solutions like counting Bloom filters and avoiding the limitations of false-negative-prone alternatives. Third, security extensions were considered to couple iBFs to time and packet contents, providing a method to secure iBFs against tampering and replay attacks. Finally, we validated the extensions in a rich simulation set-up, including useful recommendations for efficient hashing implementations. We hope that this paper motivates the design of more iBF extensions and new networking applications.

Acknowledgments

We thank the reviewers of earlier versions of this paper. We are also thankful for the fruitful discussions with Mats Naslund, Pekka Nikander and Andras Zahemszky. The work presented in this paper is supported by Ericsson Research, CNPq and FAPESP.

Appendix A. Mathematical model for *d*-candidate *fpa* optimization (adapted from [15])

Given the iBF parameters m, n, k, and letting d be the number of different iBF candidates for the same element set, the probability of setting arbitrary but fix s bits in an iBF candidate can be formulated as an independent random variable experiment:

$$E^2[s] = m\left(1-\left(1-\frac{1}{m}\right)^{kn}\right) + m(m-1)\left(1-2\left(1-\frac{1}{m}\right)^{kn}+\left(1-\frac{2}{m}\right)^{kn}\right), \quad (A.1)$$

$$\sigma^2[s] = m\left(\frac{m-1}{m}\right)^{kn} + m^2\left(\frac{m-2}{m}\right)^{kn} + m\left(\frac{m-2}{m}\right)^{kn} + m^2\left(\frac{m-1}{m}\right)^{2kn}. \quad (A.2)$$

Defining $\mu = E[s]$ and $\sigma = \sigma[s]$, the minimum continuous probability density function is:

$$f_{min}(s) = \left(\frac{1}{2^{d-1}}\right) d\left(erfc\left(\frac{s-\mu}{\sigma\sqrt{2}}\right)\right)^{d-1}\left(\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(s-\mu)}{2\sigma^2}}\right). \quad (A.3)$$

Consequently, the expectation of the least number bits (s_{min}) set by any of d candidates:

$$E(S_{min}) = \int_{-\infty}^{\infty} s f_{min}(s) ds. \quad (A.4)$$

Finally, the probability of a false positive once the smallest fill ratio has been estimated:

$$pr[false\ positive] = \left(\frac{E[s_{min}]}{m}\right)^k. \quad (A.5)$$

Appendix B. Element deletability probability

Consider a bit array of size $m' = m - r$ with $\lceil m'/r \rceil$ bit cells per region. The probability that a given cell has at least one collision is $p_c = 1 - p_0 - p_1$, where p_0 denotes the probability that a given cell is set to 0 and p_1 is the probability that a given cell is set to 1 only once after inserting n elements:

$$p_0 = (1 - 1/m')^{kn} \quad (B.1)$$

and

$$p_1 = (kn)(1/m')(1 - 1/m')^{kn-1}. \quad (B.2)$$

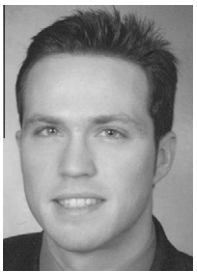
Then, the probability that a m'/r bit region is collision-free is given by $(1 - p_c)^{m'/r}$. Finally, for $r \geqslant k$ and $m \gg k$, the probability of an element being deletable (i.e., with one of its k bits in a collision-free region) can be approximated to:

$$p_d = (1 - (1 - p_c)^{m'/r})^k.$$

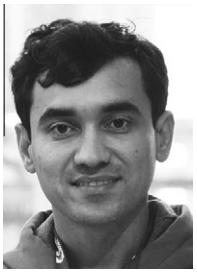
References

[1] A.Z. Broder, M. Mitzenmacher, Network applications of Bloom filters: A survey, Internet Math. 1 (4) (2003).
[2] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[3] H. Cai, P. Ge, J. Wang, Applications of Bloom filters in peer-to-peer systems: Issues and questions, in: NAS'08: Proceedings of the 2008 International Conference on Networking, Architecture, and Storage, Washington, DC, USA, 2008, pp. 97–103.
[4] P. Hebden, A. Pearce, Data-centric routing using Bloom filters in wireless sensor networks, in: M. Palaniswami (Ed.), Fourth International Conference on Intelligent Sensing and Information Processing (ICISIP-06), IEEE Press, Bangalore, India, 2006, pp. 72–78.
[5] T. Wolf, Data path credentials for high-performance capabilities-based networks, in: ANCS'08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ACM, New York, NY, USA, 2008, pp. 129–130.
[6] F. Ye, H. Luo, S. Lu, L. Zhang, S. Member, Statistical en-route filtering of injected false data in sensor networks, in: INFOCOM, 2004, pp. 839–850.
[7] S. Ratnasamy, A. Ermolinskiy, S. Shenker, Revisiting IP multicast, in: Proceedings of ACM SIGCOMM'06, Pisa, Italy, 2006.
[8] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, P. Nikander, LIPSIN: line speed publish/subscribe inter-networking, in: Proceedings of ACM SIGCOMM'09, Barcelona, Spain, 2009.
[9] R.P. Laufer, P.B. Velloso, D. d. 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: The 32nd IEEE Conference on Local Computer Networks (LCN'07), Washington, DC, USA, 2007, pp. 548–555.
[10] A. Whitaker, D. Wetherall, Forwarding without loops in icarus, in: Proceedings of OPENARCH 2002, 2002.
[11] T. Aura, P. Nikander, Stateless connections, in: ICICS'97, Springer-Verlag, London, UK, 1997, pp. 87–97.
[12] C.E. Rothenberg, F. Verdi, M. Magalhães, Towards a new generation of information-oriented internetworking architectures, in: First Workshop on Re-Architecting the Internet, Madrid, Spain, 2008.
[13] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, Y. Tang, On the false-positive rate of Bloom filters, Inf. Process. Lett. 108 (4) (2008) 210–213.
[14] S. Lumetta, M. Mitzenmacher, Using the power of two choices to improve Bloom filters, Internet Math. 4 (1) (2007) 17–33.
[15] M. Jimeno, K. Christensen, A. Roginsk, A power management proxy with a new best-of-n bloom filter design to reduce false positives, in: 21st IEEE International Performance, Computing, and Communications Conference, 2002, pp. 125–133.
[16] A. Kirsch, M. Mitzenmacher, Less hashing, same performance: building a better Bloom filter, in: ESA'06, Springer-Verlag, London, UK, 2006, pp. 456–467.
[17] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, An improved construction for counting Bloom filters, in: ESA'06: Proceedings of the 14th conference on Annual European Symposium, Springer-Verlag, London, UK, 2006, pp. 684–695.
[18] E.-J. Goh, Secure indexes, Cryptology ePrint archive, Report 2003/216, 2003. <http://eprint.iacr.org/2003/216/>.
[19] D. Hwang, M. Chaney, S. Karanam, N. Ton, K. Gaj, Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates, in: SASC'08, February 2008. <http://mason.gmu.edu/dhwang/pdf/2008eSTREAM.pdf>.
[20] C. Rothenberg, P. Jokela, P. Nikander, M. Sarela, J. Ylitalo, Self-routing denial-of-service resistant capabilities using in-packet Bloom filters, in: 5th European Conference on Computer Network Defense (EC2ND), 2009, pp. 46–51.
[21] M. Särelä, C.E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, J. Ott, BloomCasting: security in Bloom filter based multicast, in: Springer, NordSec, Lecture Notes in Computer Science, 2010.
[22] C. Henke, C. Schmoll, T. Zseby, Empirical evaluation of hash functions for multipoint measurements, SIGCOMM Comput. Commun. Rev. 38 (3) (2008) 39–50.
[23] M. Mitzenmacher, S. Vadhan, Why simple hash functions work: exploiting the entropy in a data stream, in: SODA'08: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008, pp. 746–755.
[24] F. Hao, M. Kodialam, T.V. Lakshman, Building high accuracy Bloom filters using partitioned hashing, in: SIGMETRICS'07, ACM, New York, NY, USA, 2007, pp. 277–288.
[25] Y. Lu, B. Prabhakar, F. Bonomi, Bloom filters: design innovations and novel applications, in: 43rd Annual Allerton Conference, 2005.
[26] R. Nojima, Y. Kadobayashi, Cryptographically secure bloom-filters, Trans. Data Privacy 2 (2) (2009) 131–139.
[27] C.E. Rothenberg, C.A.B. Macapuna, F. Verdi, M. Magalhães, A. Zahemszky, Data center networking with in-packet bloom filters, in: SBRC 2010, 2010.
[28] C.A.B. Macapuna, C.E. Rothenberg, M. Magalhães, In-packet Bloom filter based data center networking with distributed OpenFlow controllers, in: IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010), Miami, Florida, USA, 2010.
[29] M. Särelä, C.E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, J. Ott, Forwarding anomalies in Bloom filter based multicast, INFOCOM, Shanghai, China, 2011, accepted for publication.
[30] M. Särelä, J. Ott, J. Ylitalo, Fast inter-domain mobility with in-packet Bloom filters, in: MobiArch'10: Proceedings of the Fifth ACM International Workshop on Mobility in the Evolving Internet Architecture, ACM, New York, NY, USA, 2010, pp. 9–14.
[31] A. Zahemszky, P. Jokela, M. Sarela, S. Ruponen, J. Kempf, P. Nikander, MPSS: Multiprotocol Stateless Switching, in: Global Internet Symposium 2010, 2010, pp. 1–6.
[32] C.E. Rothenberg, C.A.B. Macapuna, F. Verdi, M. Magalhães, The deletable Bloom filter: a new member of the Bloom family, IEEE Commun. Lett. 14 (6) (2010) 557–559.



Christian Esteve Rothenberg is a Research Scientist in the areas of IP systems and networking at Fundação Centro de Pesquisa e Desenvolvimento (CPqD), Campinas, Brazil. His current research interests span Internet routing and packet forwarding, data center networks, NGN/IMS, OpenFlow, and named data networking. He received his Ph.D. in Electrical and Computer Engineering from University of Campinas UNICAMP) in 2010. In his doctoral studies he worked on probabilistic data structures applied to packet forwarding in content-oriented networks. He was a visiting researcher at Ericsson Research Nomadiclab, Helsinki, Finland, and contributed to the EU FP7 Publish/Subscribe Internet Routing Paradigm (PSIRP) project. He holds the Telecommunication Engineering degree from the Technical University of Madrid (UPM), Spain, and the M.Sc. (Dipl. Ing.) degree in Electrical Engineering and Information Technology from the Darmstadt University of Technology (TUD), Germany, 2006. During his master thesis at Deutsche Telekom he worked on IMS-based fixed mobile convergence and mobility management, and was engaged in R&D activities on converged access networks (ScaleNet) and self-optimizing radio access networks.



Carlos Macapuna obtained the Computer Engineering degree from the Federal University of Pará (UFPA) in 2008. Currently, he is finishing the Master degree at University of Campinas (UNICAMP), where he will start his PhD in 2011. During the graduate studies he worked with ADSL technologies, and currently, he is interested in OpenFlow technologies, data center architectures and new packet forwarding paradigms.

Maurício F. Magalhães received the B.S. degree in Electrical Engineering from the University of Brasilia, Brazil, the M.S. degree in Electrical Engineering/automation from the University of Campinas (UNICAMP), Brazil, and the Dr. Engineer degree from the Laboratoire d'Automatique de Grenoble (LAG)/INPG, France, in 1975, 1979, and 1983, respectively. Currently, he is a Professor in the Faculty of Electrical and Computer Engineering, UNICAMP. His research interests include next-generation Internet and service architectures.

Fábio Luciano Verdi is a Full Professor of Computer Science at Federal University of São Carlos (UFSCar) Campus Sorocaba. He received his Master degree in Computer Science and Ph.D. degree in Electrical and Computer Engineering both from University of Campinas (UNICAMP). His research interests include computer network management, data centers, cloud computing and smart grid.

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005

Dr. Alexander Wiesmaier studied computer science with the focus on cryptography, communication networks, and software engineering. He did his final year thesis on IT security and software engineering, especially in the field of public key infrastructures (PKI). In his doctoral studies he deepened and broadened his knowledge in many areas of IT security. Besides PKI and amongst other topics he also dealt with embedded systems, information systems, electronic voting, and electronic identities. During and after his studies he gained practical work experience in IT security companies such as FlexSecure GmbH (Eberstadt, Germany) and Safelayer Secure Communications (Barcelona, Spain). Currently, he holds a position as Senior Researcher at Technische Universität Darmstadt (Department of Cryptography and Computer Algebra – CDC/Center for Advanced Security Research Darmstadt – CASED). He is in charge of research projects in the field of applied cryptography.

Please cite this article in press as: C.E. Rothenberg et al., In-packet Bloom filters: Design and networking applications, Comput. Netw. (2011), doi:10.1016/j.comnet.2010.12.005