The Deletable Bloom filter - A new member of the Bloom family -

Christian Esteve Rothenberg, Carlos A. B. Macapuna, Fábio L. Verdi† and Maurício F. Magalhães
University of Campinas (Unicamp), †Federal University of São Carlos (UFSCar)
{chesteve, macapuna, mauricio}@dca.fee.unicamp.br, †verdi@ufscar.br

***Abstract*—We introduce the Deletable Bloom filter (DlBF) as a new spin on the popular data structure based on compactly encoding the information of where collisions happen when inserting elements. The DlBF design enables false-negative-free deletions at a fraction of the cost in memory consumption, which turns to be appealing for certain probabilistic filter applications.**

I. Introduction

The Bloom filter (BF) [1] is a popular data structure capable of answering questions of the form "is element x in set S?", with some tunable probability of returning false positives, i.e., claiming that x belongs to S even when this is not true. Due to its simplicity and wide applicability, BFs have become very interesting objects of study and a daily aid of system implementations. The 40-year-old hash-based data structure is beloved by theoreticians due to the mathematics that underpin the randomized flipping of 0s into 1s, and is beloved by practitioners as a powerful ally when aggregating data sets. BFs turn resource-intense (memory, computation) operations into simple, resource-friendly set membership problems. The Bloom domain spans from hardware implementations, all the road up the system stack to the software application domain, where it first saw the light to perform space- and time-efficient dictionary look-ups.

The design of BFs is fundamentally about tradeoffs, i.e., striking the right balance between memory, computation and (false positive) performance. Several variations have been proposed to modify the behavior of the standard Bloom filter (SBF) beyond its natural limits, for instance, sacrifying its zero false negative characteristic in favor of less false positives, e.g., [5]. Due to its broad scope of applications, such metamorphoses are commonly needed to meet application-specific requirements or render additional features like frequency queries, deletions, coding values, security, and so on.

In this letter, we give birth to a new BF spin-off: The Deletable Bloom filter (DlBF). The DlBF inherits the plainness of its progenitor and introduces only a simple yet powerful idea, namely keeping record of the bit regions where collisions happen. The proposed design tradeoff turns out to be useful for applications with the following requirements:

- R1: Probabilistic guarantees of element deletability.
- R2: No false negatives upon element deletion.
- R3: Fixed memory allocation.
- R4: Low impact on the false positive rate (fpr) i.e. comparable to a SBF of the same bit size m.

Like other Bloom scions in the past, our needs for another Bloom variant come from a specific networking application (see in-packet BF examples in Sec. V). However, the DlBF is well suited for other use cases where re-constructing the filter upon set membership changes is either unfeasible or too costly. For standalone applications, removal of element fingerprints is commonly desirable for functionality or optimization purposes. For distributed applications, a deletable filter can be thinned out as queried elements are processed in order to (i) avoid repeated matches upfront, (ii) reduce false positives, and/or (iii) enable fresh bit space for new additions.

II. Related work

The first Bloom descendant with genes for deletability is the Counting Bloom filter (CBF) [6], which basically extends the 1-bit cells to c-bit counters. Unfortunately, this c-fold reduction of practical bit space, typically 3 or 4 bits to avoid counter overflows, is a price too high in memory consumption (e.g., on-chip memory). Bloom relatives that improve this waste of space include the Spectral Bloom filter [4], and "an optimal Bloom filter replacement" [9]. While proven by theory to be more space-efficient, both alternatives come with a non-negligible complexity overhead, missing thereby the implicit requirement of *simplicity*, a critical factor for actual implementations. The d-left CBF (dlCBF) [2] is probably the best alternative construction for a CBF. Based on d-left hashing and element fingerprints, the dlCBF is simple, and given a target fpr, it requires about half the bit-space m of a 4-bit CBF. However, aiming at a fpr comparable to a SBF (R4), we can not afford around $2m$ for construction.

Closest to our design, is the Bloom filter with variable-length signatures (VLF) by Lu et al. [8], which presents an elegant solution to the deletion problem by resetting only a fraction of the k bits. Unfortunately, the main caveat of the VLF is that it is prone to false negatives, missing thereby R2. To the best of our knowledge, there is no Bloom filter variation which simultaneously satisfies all requirements R1-R4.

III. Design

The DlBF is built on the simple idea of tracking where bit collisions occur when inserting elements and exploits that bits set by only one element can be safely deleted. The proposed amendment consists of compactly encoding the regions of deletable bits using a fraction of the filter memory. An element can be effectively removed if at least one of its bits is reset,

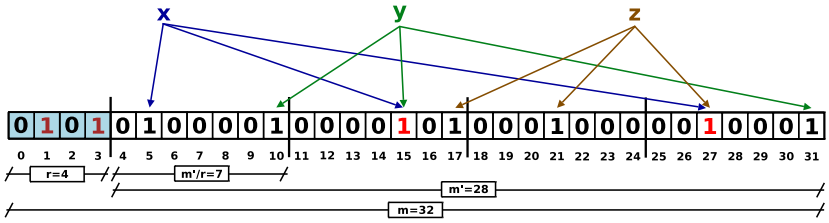

Fig. 1. Example of a DlBF with $m = 32$, $k = 3$ and $r = 4$, representing $S = \{x, y, z\}$. The 1s in the first r bits indicate collisions in the corresponding regions and bits therein cannot be deleted. All elements are deletable as each has at least one bit in a collision-free zone.

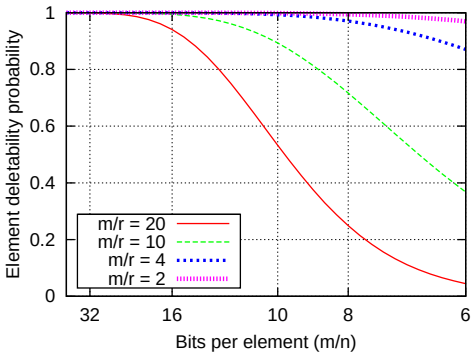

Fig. 2. Deletability estimate as function of the filter density m/n for different collision bitmap sizes r.

i.e., located in a collision-free-region. We divide a bit array of size m into r regions of $\lceil m'/r \rceil$ bits each, where m' is the original m minus the bits required to code the information of the collisions. A straightforward approach to compactly represent this information is a bitmap of size r to code with 0 a collision-free region and with 1 otherwise (see Fig. 1). Element insertion and lookup are the same as in a traditional BF. In addition to adding and maintaining a collision bitmap of size r, the DlBF adds an element removal primitive:

- *Insert(x)* maps an element x to k bit positions determined by a set of independent hashes. If one bit cell happens to be already set (collision), the corresponding region is marked in the r bitmap as non-deletable.
- *Query(x)* returns *true* if the k bit positions are set to 1.
- *Remove(x)* clears only those bit positions among k which are located in collision-free zones.

False-negatives are avoided at the cost of some elements not being deletable and accounting now as false positives, which are acceptable by the Bloom filter principle. Orphaned (non-removable) bits contribute to a larger fill factor, which, in turn, deviates the observed fpr from the expected value if all parameters were optimized. Consequently, one limitation of the DlBF appears in dynamic applications with frequent deletions and insertions where orphaned bits may fill the filter until collisions have happened in every region, hampering future deletions and increasing the residual fpr.

Since element removal is only probabilistic, a key design issue is choosing the value r and quantifying its impact on (i) the capacity to remove elements, and (ii) the false positive behavior (before and after elements are removed). First, we provide the mathematical model for the element deletability probability and then we estimate the false positive penalties.

A. 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:

$$p_0 = (1 - 1/m')^{kn} \text{and } p_1 = (kn)(1/m')(1 - 1/m')^{kn-1}$$

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

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

Figure 2 plots p_d against the filter density m/n for different memory to regions ratios m/r, confirming the intuition that increasing r results in a larger portion of deletable elements. As more elements are inserted (lower m/n), the number of collisions increase and consequently 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.

B. False positive probability

The false positive impact of consuming r bits from m can be estimated by updating m in the well-known false positive probability of a BF:

$${p_r}^k = \left[1 - \left(1 - \frac{1}{m - r}\right)^{k*n}\right]^k \quad (2)$$

Obviously, the false positive degradation is driven by the ratio m/r. With r being only a fraction of m, the false positive increase is controllable and arguably comparable to a SBF, satisfying thus $R4$ ($fpr_{m'=m-r} \approx fpr_m$).

IV. Practical Evaluation

We now evaluate via simulation the practical performance of the DlBF in terms of *deletability* and observed fpr. We answer the questions of (1) how many elements can be safely removed in practice, and (2) how many false positives are observed before and after elements are removed.

Due to space limitations, we present only the experimental results for the case where $m = 240$ and $k = 5$, which corresponds to the configuration of the in-packet BF application [7] that motivated the DlBF design (see Sec. V). On every

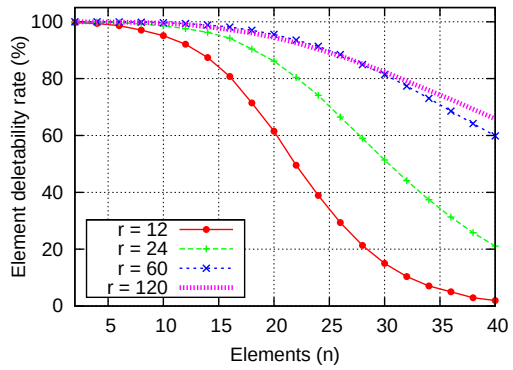

Fig. 3. Experimental deletability rate (mean values) of a 240-bit DlBF with $k = 5$ for different number of regions r.

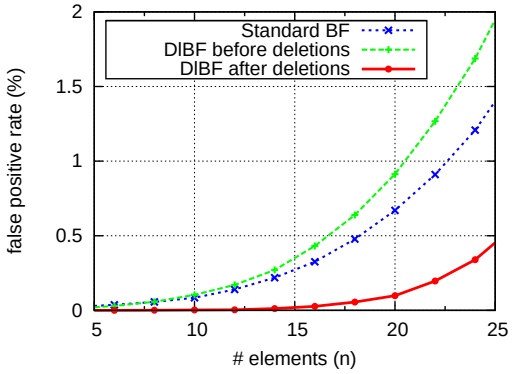

Fig. 4. Observed average fpr of a 240-bit DlBF with $r = 24$ and $k = 5$, before and after removing elements, compared to a 240-bit SBF.

trial (2000 per parameter set), we insert n elements randomly chosen from the American dictionary ($\approx$ 145K entries). We then (i) quantify how many inserted elements can be deleted (Fig. 3), and (ii) count for false positives (Fig. 4) by testing 500 randomly chosen elements (before and after deletions).

The observed deletability rate behaves as predicted by theory, but with relatively lower values (noteworthy as r tends to $m/2$ and for high m/n ratios). This can be explained by the assumption in Eq. 1 of perfectly random hash functions, an issue which can be more significant in small BFs [3]. Taking as an example the case where 10% of the memory is used to code the bitmap ($r = 24$), under a reasonably utilization ($n = 22$), on average, 80% of the elements could be removed (compared to 90% predicted by theory) by resetting around 40% of the bits (not shown in Fig. 3). Interestingly, doubling r from 60 to 120 only improves the number of deleted bits but not the actual element deletability. As expected, the price in fpr (Fig. 4) is an affordable increase before elements are removed, and a potential improvement when element bits are deleted. For other parameters (m, r, k), we could verify the adherence to theory, with the above noted divergences, too.

V. Example applications and future work

We now give a snapshot on two networking applications to illustrate the practical use of the DlBF when placed into fixed-length packet headers. In LIPSIN [7], the inserted elements are unidirectional link identifiers (LID). A 256-bit source routing BF can be constructed by including the LIDs of a multicast delivery tree. Being able to remove already processed LIDs enables (1) avoiding loops, and (2) deleting special LIDs like multi-hop virtual links or control messages.

In a second scenario, we are exploring the DlBF in a data center environment to compactly represent a sequence of middlebox services (e.g., firewall, load balancer, DPI) which a packet needs to transverse. Relying on a substrate of switch programmability (OpenFlow), the content of the DlBF is used to transparently forward packets upon match on Bloomed Service IDs, which are removed after leaving the middlebox.

Future work includes exploring *dynamics* along two axes. First, understanding the practical limits if we keep doing insertions and deletions. Second, investigating a dynamic adaptation of the amount and the bit range of the deletable regions in function on how collisions happen. An open question is if there are other compact and more flexible ways to code the information of the collision-free regions. Finally, the *power of choices* at hashing time may introduce another interesting interplay. For instance, creating d DlBF candidates with different sets of hash functions and selecting the best in terms of fpr or guarantees that certain elements are deletable.

VI. Conclusion

This letter introduces the deletable Bloom filter (DlBF), a new Bloom engenderment based on the idea of compactly encoding the information of where collisions happen when inserting elements. This allows safely (i.e. without introducing false negatives) elements removal. Depending on how much memory space one is willing to invest, different rates on element deletability and false positives can be achieved. The DlBF is simple and can be easily plugged to existing BFs. We briefly presented two packet forwarding applications benefiting from the DlBF, which we believe could be a good fit for existing (and upcoming) friends of the Bloom principle.

References

[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[2] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *ESA'06*. London, UK: Springer-Verlag, 2006, pp. 684–695.

[3] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, "On the false-positive rate of Bloom filters," *Information Processing Letters*, vol. 108, no. 4, pp. 210–213, 2008.

[4] S. Cohen and Y. Matias, "Spectral bloom filters," in *SIGMOD '03*. New York, NY, USA: ACM, 2003, pp. 241–252.

[5] B. Donnet, B. Baynat, and T. Friedman, "Retouched bloom filters: allowing networked applications to trade off selected false positives against false negatives," in *CoNEXT '06*, New York, NY, USA, 2006.

[6] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.

[7] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line speed Publish/Subscribe Inter-Networking," in *ACM SIGCOMM*, Barcelona, Spain, August 2009.

[8] Y. Lu, B. Prabhakar, and F. Bonomi, "Bloom filters: Design innovations and novel applications," in *43rd Annual Allerton Conference*, 2005.

[9] A. Pagh, R. Pagh, and S. S. Rao, "An optimal Bloom filter replacement," in *SODA '05*, Philadelphia, PA, USA, 2005, pp. 823–829.