First publ. in: IEEE Computer Society, 32nd IEEE Conference on Local Computer Networks, 2007, pp.183-190
http://doi.ieeecomputersociety.org/10.1109/LCN.2007.17

Bloom Filters: One Size Fits All?

Paul Hurley
IBM Research
Zurich Research Laboratory
pah@zurich.ibm.com

Marcel Waldvogel
University of Konstanz
marcel.waldvogel@uni-konstanz.de

***Abstract*— Bloom filters impress by their sheer elegance and have become a widely and indiscriminately used tool in network applications, although, as we show, their performance can often be far from optimal. Notably in application areas where false negatives are tolerable, other techniques can clearly be better. We show that, at least for a specific area in the parameter space, Bloom filters are significantly outperformed even by a simple scheme. We show that many application areas where Bloom filters are deployed do not require the strong policy of no false negatives and sometimes even prefer false negatives. We analyze, through modelling, how far Bloom filters are from the optimal and then examine application specific issues in a distributed web caching scenario. We hope to open up and seed discussion towards domain-specific alternatives to Bloom filters while perhaps sparking ideas for a general-purpose alternative.**

I. Introduction

Bloom filters [1] pop up everywhere in networking. and is the tool of first and last resort when looking at how to compactly store set membership information in some form or another. Yet it is self-evident that Bloom is not the one and only way to go. We posed the question as to whether better performance could be achieved; or if different applications would benefit from, or even necessitated, other schemes for set membership representation. This work shows that indeed Bloom may, in certain circumstances, be far from optimal, and even very simple schemes may do better and be more appropriate.

Many efforts have focused on enhancing Bloom filters, while one work [2] showed the benefit of compressing them; something which had previously been considered impossible. We take this one stage further and return to the underlying goal–the compression of set membership information. This helps in understanding the limitations (as well as advantages) of Bloom filters and aids in the derivation of other schemes that may be tailored to a particular networking application.

Bloom filters were first used to circumvent shortage of local memory with main applications in the database domain. Nowadays, they serve also as a method for set membership information exchange in distributed computing scenerios where communication cost is the main limiting factor. Some application areas include distributed caching [3], object location in P2P networks [4], approximate set reconciliation [5], and set intersection for keyword searches [6], [7].

TODO:more networking examples: ip traceback, routing, etc

A compact representation of a set is a data structure that stores information as to whether an element is contained in it. Using a representation size far below the size of explicitly listing the set members can significantly reduce the storage size with a resultant trade-off in accuracy. By far the most prominent method is the aforementioned Bloom filter. This and its derivatives and enhancements are the sole methods applied in many network applications to date. It consists of an array of bits, initially all set to zero. To add an element, several hash functions with range over the bitmap are calculated. The bits corresponding to the hash function output are set to one. Set membership is ascertained by performing an AND operation on all the bits stored at the indices that equal the hash values of the requested element. The property of the filter that stands out is the production of one-sided errors in membership determination. A member of the set will always be correctly ascertained as being such – or in other words, there are never false negatives. The flip-side is that elements not in the set may falsely be determined as belonging to the set – a false positive. This is because all hash functions could map to values that were previously set to one by other elements.

The restriction to only false positives is a strong characteristic, which, intuitively, comes at a price, which–when this characteristic is required or helpful–may be worth paying. However, Bloom filters are so handy (and currently the only kid on the block), that they seem to be used independently of such considerations. Table I illustrates some of the most prominent network application areas and their respective requirements for one-sided error.

The list of applications comes from [8], where more detailed descriptions and explanations of their uses can be found. One application that does not rely on one-sided error is distributed caching, whereby the validity of resources may expire and new ones become available. Here, the existence of both false positives and false negatives is inherent to caching and, while not particularly cherished, tolerable.

In some applications, false negatives are even less costly than false positives. For example, consider the problem of calculating set subtraction as part of approximate set reconciliation. False positives will introduce actual synchronisation errors, whereas false negatives would simply cause some overhead. Throughout, we will explore the tradeoff between the types of mistakes in set determination by use of a parameter α that controls the fraction of mistakes that may be false negatives.

TABLE I
NECESSITY OF RESTRICTION TO ONLY HAVING FALSE POSITIVES.

Application area [8]	Necessity of one-sided error
Distributed caching	No
Object location in P2P systems	No
Approximate set reconciliation	No*
Resource routing	No
Loop detection	No*
Flow detection	Yes
Multicast	Yes
Hyphenation exceptions	Yes
Set intersection	Yes
Differential files	Yes

* False negatives preferable

This paper first analytically determines the performance of the Bloom filter and a very simple scheme we call the *cropped filter* against the performance bounds given under a simple, tractable model. We then look at how well we can apply our findings in practice by a performance evaluation in a distributed cache scenario. We show that our naïve cropped filter significantly outperforms Bloom filters for high compaction rates, both in the analysis and cache simulation. While several of the techniques described here could also be used in conjunction with Bloom filters, it turns out that pure cropped filters are more efficient than a combination of cropped and Bloom filters. Cropped filters are of course by no stretch the ultimate solution, and we believe smarter alternatives will be found.

The rest of the paper is organised as follows. After presenting related work in Sect. II, Sect. III presents an average-case analysis of set compression from modelling. Section IV describes specific application issues and summarises simulation results obtained by different set membership compression methods. We conclude in Sect. V with a short discussion of our results and an outlook on related future research topics.

II. RELATED WORK

During the past 35 years, the Bloom filter captivated many researchers and a multitude of evolutions subsequently emerged. Most go in the direction of new functionality, which is not the scope of this work. One example is the *counting Bloom filter* [3], which introduces the ability to easily remove elements. Instead of a bit array, the counting Bloom filter uses a small number of bits per entry to keep count of the number of elements, incremented upon insertion and decremented upon deletion. The *bloomier filter* [9] looks at a Bloom filter as a data structure for compactly encoding a function. It extends the existing filter to encode arbitrary functions. Bloom filters have also become an integral part of more complex methods. The *attenuated Bloom filter* [4], for example, uses them for dynamic document location.

The *compressed Bloom filter* [2] takes a step in the direction of performance improvement, a goal it shares with this work. It alters the Bloom filter in a way that improves accuracy for a given size using compression, and looks at Bloom filters as both a data structure to be used at proxies as well as a message to be passed between them. A new optimization problem is set up with the message size as a newly introduced parameter. The compression rate depends on the size of the local Bloom filter and the number of hash functions used. To achieve minimum message size, infinite local memory and a huge number of hash functions would be required, clearly impractical. Thus all the compressed Bloom filter results shown in this paper describe a (generally impractical) lower bound.

In the context of image compression, Weidmann and Vetterli [10], [11] study spike processes and in particular the rate distortion of sparse memoryless sources. This can also be applied to the study of set representation, as explained later in Sect. III-F.

III. AVERAGE-CASE ANALYSIS

In this section, we analytically study how far Bloom filter performance differs from optimal compression under a certain model. To the best of our knowledge, this is the first time such a comparison has been performed, and the results, we believe, are illuminating. We also describe a simple, alternate way to compress set information which turns out to be better than one might have expected.

TODO:describe α at this point

A. Problem Definition

TODO:Completely rewrite this part Consider the set membership problem as follows. Let $S = \{s_1, ..., s_n\}$ be a subset of a global set G, i.e. $s_i \in G$. Let y be a set membership representation of S, which uses m bits of information. Then, $\hat{S}$ is the reconstruction of S from y.

$$S \to x \xrightarrow{Encode} y \xrightarrow{Decode} x \to \hat{S}$$

For example, in a Bloom filter, G is the set of all possible inputs, S the subset of G that is inserted into the filter, y the output of the filter, and $\hat{S}$ all the elements in G that according to the Bloom filter are contained in y. In practice, $\hat{S}$ would rarely be explicitly constructed.

WRITE THIS: We examine contained in y by counting the wrong answers to set membership queries in $\hat{S}$, which In information theory, such an error is referred to as a *distortion.*

B. Distortion

The probability of a membership query being answered incorrectly can be used to compare different methods. The measure used in the following analysis is the *Hamming distortion* [12], whereby the distortion occurring for input x and output $\hat{x}$ is given by

$$d(q, \hat{q}) = \begin{cases} 0 & \text{if } x = \hat{x}, \\ 1 & \text{if } x \neq \hat{x}. \end{cases}$$

For this measure, the expected distortion is equivalent to the probability of a mistake. Let $\boldsymbol{x} = \{x_1, ..., x_n\}$ be a vector of size n generated from the set S, where each x_i is set to 1 if

$s_i \in S$ and zero otherwise. This vector $\boldsymbol{x}$ is the characteristic function of S. We similarly associate a vector $\hat{\boldsymbol{x}}$ with the set $\hat{S}$. The expected distortion between the two sets is then

$$d(S, \hat{S}) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, \hat{x}_i). \tag{1}$$

So the distortion between two sets is the average of the per-symbol distortion of the individual elements.

C. Input Model

The input model used is one where the probability of an element being in a set is independently and identically distributed. This corresponds to considering the input as a discrete memoryless channel - which in this case is a Bernoulli model. We are then enabled to understand, in the first instance, this basic case, which also simplifies the analysis. Additionally, when identifiers are composed of a hash of the entire space, a Bernoulli model may be appropriate.

The model is useful not only for its tractability, but also because it is not unfair on Bloom filters. On the contrary, neglecting any possible correlation in the input shows them in a better light performance-wise than they otherwise would, since correlation is inherently neglected by the hashing mechanism.

Every element of the global set G is contained in the subset S with a fixed probability p, thus,

$$\mathsf{P}(s \in S) = p \quad \text{with} \quad \mathsf{P}(s \notin S) = 1 - p.$$

The global set G is finite (although it can be made extremely large). For simplicity, it is assumed throughout that $p \leq 0.5$, which is the typical situation under consideration. If $p > 0.5$, the problem can be interpreted as storing the complement $G \setminus S$. All $\log$ functions are base 2 unless otherwise specified.

An alternative model would be to consider the size of the set S given. There are small differences in this model, and we shall return to this difference. **TODO:improve**

In traditional Bloom filter analysis, what is meant by the probability of a false positive is in fact the conditional probability, namely the probability of a false positive given that it is known that the element being tested is not in S. We consider the marginal probability.

D. Lower Bounds

We now show lower bounds on the expected rate of communication needed must be in order to be able to transmit a given amount of information at a given fidelity [13]. The function defining this bound is called the information rate-distortion function. This returns the expected rate for a given distance measure, and thus the results here are an average-case.

The rate R is the number of bits in the encoded sequence per bit from the input. In other words, it is the compression ratio. When the distortion is zero, the source can be encoded losslessly.

The information rate-distortion function for a Bernoulli source with parameter p combined with Hamming distortion is a well known result (e.g. see [12]) and given by

$$R(D) = \begin{cases} H(p) - H(D), & 0 \leq D \leq p, \\ 0, & D > p, \end{cases} \tag{2}$$

where $H(p)$ is the entropy of a Bernoulli(p) source, namely that $H(p) = -p \log p - (1-p) \log (1-p)$.

The source is the characteristic function of the set S in G, as previously described in Sect. III-B. A lower bound for the fidelity or expected error rate of all possible set membership representations of a given size is illustrated in Fig. 1.

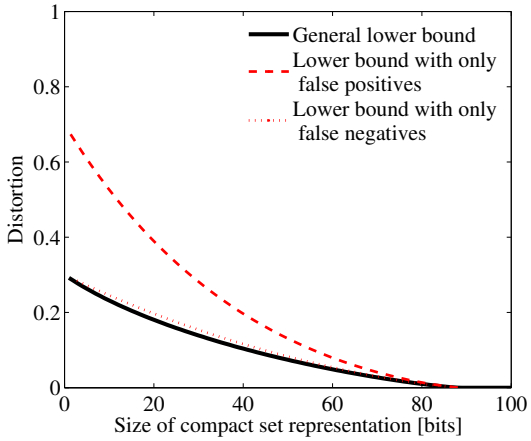


Fig. 1. The general lower bound compared the lower bounds with only false positives and only false negatives. (Assuming equiprobable set membership, $p = 0.3$, and global set size $|G| = 100$.)

We also attain absolute lower bounds for methods restricted to a one-sided error. We show in the appendices that the theoretically reachable rate at a distortion where only false positives are permitted is:

$$R(D) = \begin{cases} -p \log p + D \log D - \\ (1 - p + D) \log (1 - p + D), & D \leq p, \\ \\ 0, & \text{otherwise.} \end{cases}$$

The achievable rate when only false positives are allowed is

$$R(D) = \begin{cases} -(1-p) \log (1-p) + D \log D - \\ (D + p) \log (D + p), & D \leq 1 - p, \\ \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

These two lower bounds are also plotted in Fig. 1 which shows that, especially for high compression **TODO:this is wrong**, more bits are necessary to reach a certain distortion with only false positives than when allowing false negatives or both-sided errors.

E. Bloom Filter

Let us now compare Bloom filter performance to the lower bound given in (2). Using the probability of a false positive

[8], the expected error probability for a given size m of the Bloom filter which uses k hash functions in a space of n possible elements is approximately,

$$**TODO**.$$

DERIVE IT TOO - ALSO An example of the expected error probability as the size use to represent the set varies, is shown in Figure 2. The large gap between the general lower bound and the Bloom filter performance is partly due to the restriction of only false positives, as can be seen by the lower bound when only false positives are allowed. However, this accounts for some but not all of the difference. Even when false positives are absolutely necessary, there is potential for better coding.

As mentioned in Sect. II, the compressed Bloom filter improves fidelity using compression. A regular Bloom filter is kept in local memory and elements are inserted in a way that in the end clearly less or clearly more than 50% of the bits are set.[1] This enables significant compression and results in less distortion than with a Bloom filter using the same message size. A compressed Bloom filter can achieve a false positive rate arbitrarily close to $(0.5)^{\frac{m}{k}}$ after inserting k elements and assuming an optimal lossless compressor and message size m [2]. In practice, compression such as arithmetic coding may be used, which can be close to optimal [14]. Similar to the derivation for Bloom filters, the distortion for their compressed form is **TODO:insert from appendix** Fig. 2 confirms that the compressed Bloom filter performs clearly better than the uncompressed version. However, here also a gap between the theoretically reachable distortion of a set membership representation and the filters performance is evident.

F. Cropped Filter

We now describe a simple (and with hindsight, obvious) method dubbed the *cropped filter*. It directly stores the output of the source without any processing or compression. The resulting lossy representation of the set contains perfect information about m elements, m being the size of the representation. The reconstruction of the set consists of the m bits which are represented in the cropped filter and a maximum likelihood decoding of the remaining elements that we do not have further information about. For $p \leq 0.5$, this amounts to considering all elements for which we have no information as not being in the set S.

TODO:Results should not be in terms of m

The expected error probability of a cropped filter at this point would be:

$$D = \frac{n-m}{n}p$$

The second stage is to add a lossless compression step, and produce what we call the *compressed cropped filter*. The initial output is taken and fed into a compressor until the final output size reaches the space m available for the lossy set representation. A Bernoulli(p) sequence of length r can be compressed without loss at an expected rate $rH(p)$. The expected error probability of a compressed cropped filter then becomes

$$D = \left(1 - \frac{m}{nH(p)}\right)p.$$

Surprisingly, it can be shown that, when n large relative to m, the compressed cropped filter is, approximately optimal under identically and independently distributed input. Concretely, this means that as $p \to 0$, the performance of the compressed cropped filter approaches the information-theoretical lower bound. Given that the subset S will typically be of magnitudes smaller than the global set size $|G|$, p will often be very small. The normalised rate-distortion function for the normalised Hamming distortion $d = \frac{D}{p}$ is given by

$$\frac{R(d)}{H(p)} = 1 - \frac{H(pd)}{H(p)}.$$

It can then be shown [11], that the normalised rate-distortion function is asymptotically linear for small p:

$$\lim_{p\to 0}\frac{R(d)}{H(p)} = 1 - d.$$

Given this, along with the linearity of the compressed cropped filter, and the fact that the rate-distortion curve and the compressed cropped filter curve always have the intersection points with the x- and y-axis in common, means that they converge.

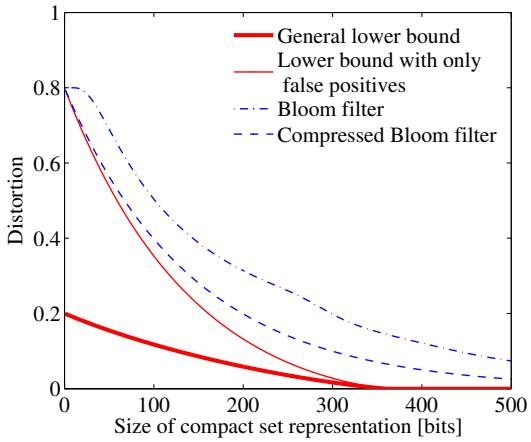
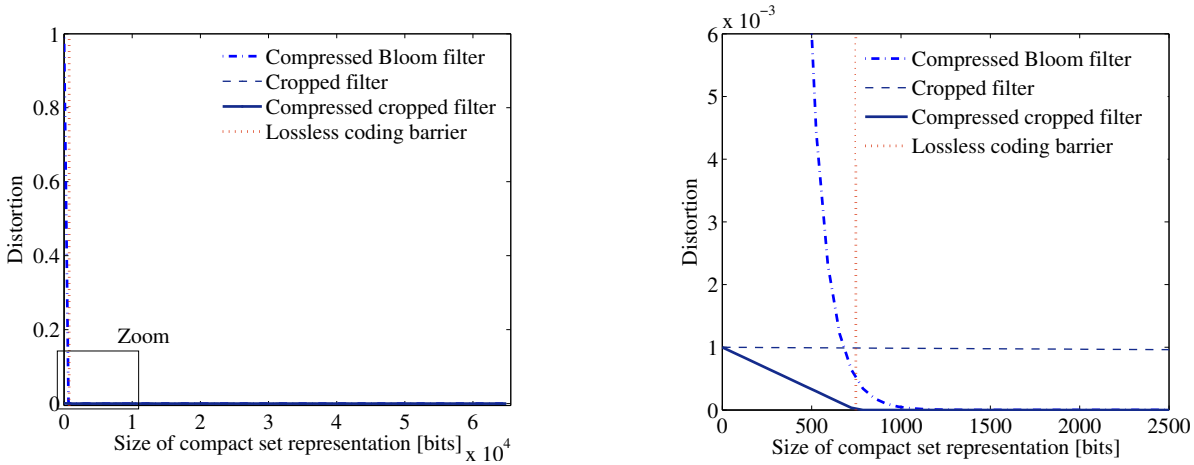


Fig. 2. Gap between theoretical lower bounds and the Bloom filter as well as its compressed version. (Assuming equiprobable set membership, $p = 0.2$, and global set size $|G| = 500$.)

G. Bloom, Cropped and Ideal Filters

Figure 3 shows different methods of set representation compared to the lower bounds. We see that the compressed cropped filter is close to the general lower bound. As a result of the significant difference between the general lower bound and the lower bound with only false positives, the cropped filter performance is below this lower bound of one-sided errors. For most representation sizes, the compressed cropped filter performs better than any method only having false positives.

[1] A traditional Bloom filter achieves maximum efficiency when 50% of the bits are set.



(a) Overview.

(b) Enlargement of interesting area.

Fig. 4. Distortion for different methods of lossy set membership representation. (Global set size $|G| = 2^{16}$ and set membership probability $p = 0.001$.)

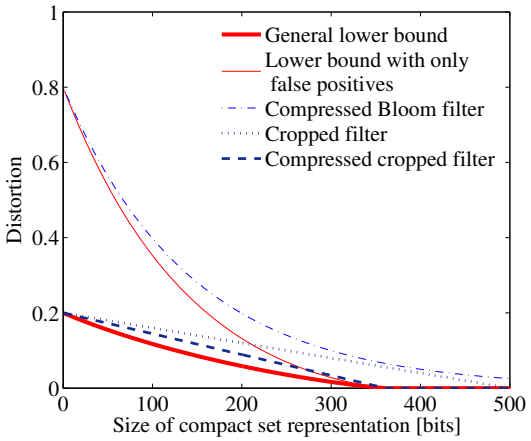


Fig. 3. Distortion behavior of several methods of set membership compressionand the theoretical error bound as well as the theoretical error bound for methods only allowing false positives. (Assuming equiprobable set membership, $p = 0.2$, and global set size $|G| = 500$.)

So far, we have considered small global set sizes. In Fig. 4, the expected error probability of the compressed Bloom filter, the cropped filter and its compressed form are plotted for a global set size $|G| = 2^{16}$, which may be more likely in practice. The interesting area lies on the left of the lossless coding barrier, since for sizes bigger than this threshold, the entire subset can be encoded without loss. One clearly sees that the compressed cropped filter yields the lowest expected error rate, confirming that, under certain conditions, there are relatively trivial ways to get better performance than from a Bloom filter.

IV. Application Specific Issues

A. Simulation Setup

In order to test the behavior and applicability of the above introduced methods of set membership compression, a trace-driven simulator of a distributed cache system was developed. We started looking at the distributed caching application. Bloom filters have often been used in this context [3], [15]. Other reasons for choosing this scenario include its simplicity and tolerance for two-sided errors – in a web environment, documents can disappear and new ones appear at any time.

Distributed caching was not a home ground for the cropped filter, given the huge and very sparsely populated URL space, as discussed below. The goal is to compare the influence of different methods for set membership compression on the response time in a distributed caching environment. The key component under study is the set representation a proxy keeps as knowledge of the documents available in neighbor proxies that are part of a distributed cache. The performance of such a system using Bloom filters, their compressed form, one using cropped filters, and one using cropped filters combined with a frequency strategy is measured and compared.

The goal in a caching system can be seen to be the minimization of response time. Since the focus of this evaluation is on the representation of documents, we only look at metrics of this subsystem that have an influence on the overall response time of the system. These are the correctness of the lookup results and the computation time per lookup.

Correctness of the Lookup Results: A false positive, namely a remote cache hit even though the remote cache does not contain the document, causes an additional superfluous request to the nearby cache. A false negative, namely a remote cache miss even though the remote cache contains the document, causes a remote request where one to a nearby cache would have been sufficient. We define the cost of a request to the internet and of a request to a remote cache to be $cInt$ and $cLoc$ respectively. Therefore, the additional cost of a false positive and of a false negative are $cLoc$ and $cInt - cLoc$ respectively. The application specific tolerance for false positives and negatives depends on the actual values of $cInt$ and $cLoc$.

In networked applications, computation time is not the

bottleneck, and we thus consider it negligible in comparison to network operations caused by false positives and negatives.

In different simulation runs, different methods of set membership compression are used and, for each one, the performance for different sizes is measured. The workload plays a very important role in the whole system because it determines the theoretically possible hit and miss rate of the cache. It should be representative, repeatable, and timely [16]. We therefore use a trace of a real web-proxy. It is a sanitised log file from the CA*netII, Canada's coast to coast broadband research network, available at http://ardnoc41.canet2.net/cache/.

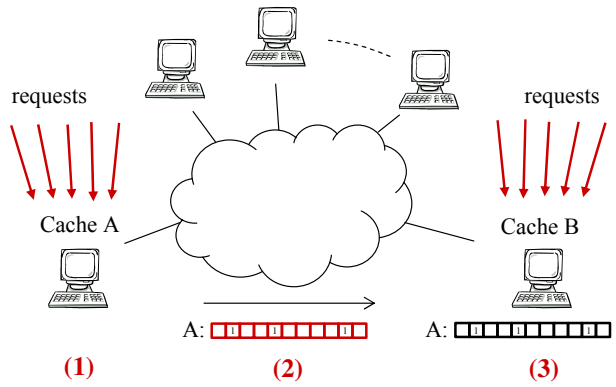


Fig. 5. Setup of the distributed cache simulation.

One simulation phase consists of three stages illustrated in Fig. 5. In the learning phase, 10 000 HTTP requests are sent to *Cache A* which stores the corresponding answers. *Cache A* then compresses by creating a representation of the identifiers of the stored resources and sends it to *Cache B*. In the Query phase, 90 000 HTTP requests are sent to *Cache B*. Using the representation of the documents at *Cache A*, it is checked whether a document is available from *Cache A*. The number of wrong answers is counted for different methods of set representation.

B. Implementing Set Membership Compression

With cropped filters and their compressed form, there are some implementation questions. One issue is that URLs generally can have a length of 1024 bytes and that storing this amount for each URL is not feasible. However, there are numerous possibilities for reducing the size of the label stored for each element. A first approach is to use a hash function on the URLs to reduce the label size. This may introduce hash collisions. The optimal size of such a hash value is determined with an optimization problem. In the optimum, a change in size of the hash range causes a change of collision probability that is equal to the marginal gain of an additionally stored document.

Another approach for reducing the label size is to exploit the fact that URLs generally contain redundancy that can

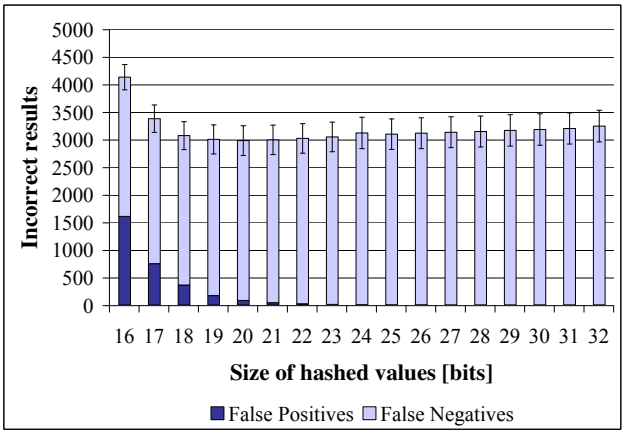


Fig. 6. Influence of hash rage size on the correctness of the lookup results in a distributed cache. The error bars reflect 95% confidence.

be removed using a lossless compression scheme such as a LZ77/78-based or arithmetic coder.

Unlike Bloom filters, cropped filters have the choice of which elements to store. A clever strategy of which elements to pick can, depending on the nature of the queries, strongly improve the performance. If an incorrectly represented element is requested multiple times, one false conclusion causes several errors. Conversely, elements never requested by the application do not cause any error in this case.

- **Pick the first few:** This is the cheapest method in terms of computation cost. Another advantage is that the compressed representation can be entirely constructed very quickly and is first available at other nodes.
- **Pick the best compressible items:** This method causes the cropped filter to contain the most elements. When no information about the nature of the queries is available, this method produces the smallest expected error rate. However, it can be quite costly to find the best compressible items.
- **Pick the most important ones:** Using information or a prediction of the nature of the queries, the elements can be chosen cleverly in order to produce the lowest error. In a distributed cache environment, we would choose the elements that have the highest expected request frequency per storage bit in the compressed representation. Other optimization metrics are conceivable such as download time, resource consumption, frequency of requests, temporal locality, or site locality.

C. Simulation Results

As mentioned previously, there are several ways to reduce the total size of a URL, before its input to the cropped filter. One possibility is to use a hash function. We examine the behavior of the resulting errors for different sizes of the hash

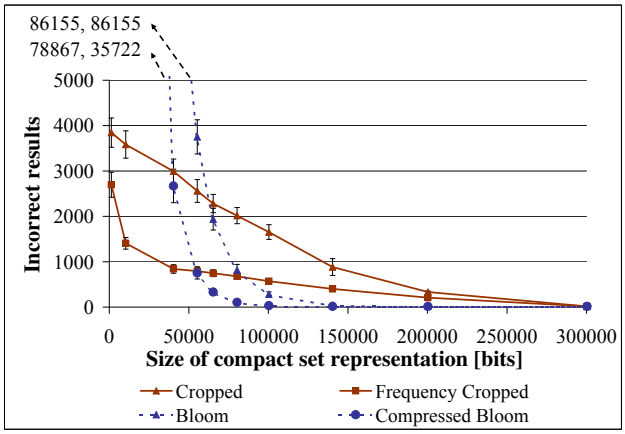


Fig. 7. Fidelity of lookup results in a distributed cache using different methods of set membership compression. The error bars reflect 95% confidence.

range. Here, the set representation simply stores the outcome of the hash function for every request until a size of 40 000 bits is reached. The incorrect results for 90 000 subsequent requests can be found in Fig. 6. The trade-of of collisions vs. size of a label is evident, with the best collision to space consumption rate of around 20 bits.

The approach of compressing the original URLs using an existing compression scheme produces unfavorable results although the achievable compression rate is around 85%. Hashing into a space of, for example, 32 bits yields better compression at a cost of almost no collisions.

The compression of the cropped filter as described in Sect. III-F is difficult, and we therefore directly store the identifiers. Application of an existing compression scheme to the list of identifiers yielded little gain and is omitted from the results.

In the subsequent simulation, the following compression methods of set membership information are compared:

- **Cropped filter:** Stores the hash values of the requests using a hash range of 20 or 32 bits, whichever yields fewer errors depending on the size of the set membership compression. The elements are stored in their request order until the filter size is reached.
- **Frequency cropped filter:** It is the same as a regular cropped filter but stores the elements that are most frequently requested first.
- **Bloom filter:** The implementation of Bloom filters is straightforward, and used in its original form as described in [1]. The URL of the HTTP request is taken as a string and the hashes calculated.
- **Compressed Bloom filter:** To the best of our knowledge, these have not been implemented to date. We use its lower bound of false positives [2] to find a lower bound for the total number of expected errors.

The compressed size of the set membership information is limited to 40 000 bits, the learning phase consists of 10 000, and the query phase of 90 000 requests. The simulation was executed 20 times using a different trace in each run. The results can be found in Fig. 7.

For filter sizes up to 40 Kbits, all different implementations of the cropped filter outperform the Bloom filter as well as the compressed Bloom filter which exhibit large distortion. For filter sizes from 60 to 300 Kbits, the (compressed) Bloom filter produces less errors than a cropped filter and above 300 Kbits, both methods become error-free. There seems to be a critical size below which the amount of false positives of a Bloom filter is extortionate. This is not astonishing since it uses at least one hash function and after inserting around 10 000 elements into an array of comparable size, we expect most bits to be set to one which in turn causes lots of false positives.

Comparing the different versions of the cropped filter, one can clearly see that choosing the elements to be inserted into a cropped filter with a frequency strategy significantly reduces the resulting mistakes compared to just inserting the first requested elements. The difference is biggest for filter sizes around 40 Kbits.

The Bloom filter does not take advantage of specific knowledge about the nature of the requests. However, it automatically exploits the correlation between the inputs and the requests. Consider the possible false positives that could arise from an URL insertion into the Bloom filter. It is extremely likely, that these strings are not proper URLs and would thus never be requested. This results in lower false positives than if all inputs were equiprobable.

Depending on the application, it is sometimes desirable to represent some elements with a low error probability and in doing so decreasing the average error rate. This is a major advantage of the cropped filter.

The frequency cropped filter using 20 bits is better than the one using 32 bits up to a filter size of 200 Kbits. This can be explained with hash collisions that start to carry weight when the overall error becomes small for larger representation sizes. For smaller representation sizes, the fact that using a small hash range allows for more elements to be stored is predominant.

Although the cropped filter is not better than the Bloom filter for all possible sizes, it has a significantly lower error rate for some. Looking at the difference between the modelling in Sect. III and the simulation results, we conclude that the (compressed) cropped filter is especially good when the elements have a reasonably small label, and when the global set is small. If a prior transformation is necessary, it should be minimised with respect to the loss of information e.g. collisions.

Using one hash function for reducing the global set is basically a special case of the Bloom filter; one with just one hash function. However, the Bloom filter typically uses an optimal amount of hash functions to prevent collisions and

is therefore superior. It combines the two steps of reducing the global set and representing a set while optimizing the overall loss.

Cropped filters have a distinct advantage. Using them, further messages containing information about more elements and thus resulting in more accuracy can easily be provided at any time whereas a Bloom filter needs to be fully constructed before transmission.

V. Conclusions

Although Bloom filters are the method of choice for compression of set membership information in a lossy fashion, we show that there can be a large price to pay. This is especially true when only false positives are permitted, but, even in this domain there is scope for improvement. As one size does not fit all, it may be worth thinking twice before their use.

A simple method of storing some elements' identifiers and ignoring the rest yields better performance under independent input. We are certain that better coding could be used to improve on this too. Additionally, using coding tailored to the application that takes into account, say, the relative cost of false positives and false negatives would certainly yield fruitful results.

Bloom filters remain strong when the application requires or benefits from restricting possible errors to false positives and when no a priori information is known about the type of the queries the filter will subsequently be used for. The cropped filter, on the other hand, is adaptable to incorporate such information. They can also be interpreted as erasure codes. Instead of simply guessing that all elements for which we have no information are not in the set, this information could be registered as missing - an erasure. Furthermore, their simplicity allows subsequent refinement of the fidelity by sending information of additional elements.

Much future work remains, particularly for suitable coding schemes. We do not contend in any way that the Bernoulli model is very representative of typical set membership. Source input is of course not independently and identically distributed. Rather, it is a significant first step, complementary to Bloom filters with already many interesting conclusions to be drawn. Extension to correlated input models is currently being investigated. In-depth analysis of the effects of reducing the global set using hash functions would be useful. The simulations considered the cache data independently between caches. Exploiting possible correlation could be done by incorporating Slepian-Wolf coding [17] in a set compression framework.

TODO:Add the enquiry model

Acknowledgements

Thanks to Andreas Diener for helping with some initial simulations and tests.

References

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

[2] M. Mitzenmacher, "Compressed bloom filters," in *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, 2001, pp. 144–150.

[3] 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, 2000.

[4] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *Proceedings of INFOCOM 2002*, 2002.

[5] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *Proceedings of ACM SIGCOMM*, Pittsburgh, Pennsivania, USA, October 2002, pp. 47–60.

[6] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," in *Proceedings of the ACM/IFIP/USENIX Middleware conference*, June 2003.

[7] D. Bauer, P. Hurley, R. Pletka, and M. Waldvogel, "Bringing efficient advanced queries to distributed hash tables," in *Proceedings of IEEE LCN*, Nov. 2004.

[8] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," in *Proceedings of the 40th Annual Allerton Conference on Communications, Control, and Computing*, 2002, pp. 636–646.

[9] 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 Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2004, pp. 30–39.

[10] C. Weidmann and M. Vetterli, "Rate-distortion analysis of spike processes," in *Proceedings of DCC'99*, 1999.

[11] ——, "Rate distortion behavior of sparse sources," EPFL, Tech. Rep., Oct. 2001.

[12] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 1991.

[13] T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1971.

[14] K. Sayood, *Introduction to Data Compression*. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc., 1996.

[15] Squid, "Squid web proxy cache," `http://www.squid-cache.org/`, 2005.

[16] R. Jain, *The Art of Computer Systems Performance Analysis*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 1991.

[17] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources," *IEEE Transactions on Information Theory*, vol. IT-19, pp. 471–480, July 1973.

Appendix A
Proofs

We derive a general equation for the rate-distortion function for a discrete memoryless source, when the encoder is explicitly constrained to satisfy a certain joint-probability distribution between output and input. Consider a coder with Bernoulli input X where $\mathsf{P}(X = 1) = p$. Let the output $\hat{X}$ also be a random variable on $\{0, 1\}$ and the joint distribution of X and $\hat{X}$ be $f(x, \hat{x}) = \mathsf{P}(X = x, \hat{X} = \hat{x})$.

The rate-distortion function $R(D)$ is then, for the Hamming distortion measure (1),

$$R(D) = \min_{\mathsf{P}(\hat{X}=\hat{x}|X=x):f(0,1)+f(1,0)\leq D} I(X; \hat{X})$$

where $I(X; \hat{X})$ is the mutual information,

$$I(X; \hat{X}) = \sum_{x=0}^{1} \sum_{\hat{x}=0}^{1} f(x, \hat{x}) \log \frac{f(x, \hat{x})}{\mathsf{P}(X = x)\, \mathsf{P}(\hat{X} = \hat{x})}.$$

which can also be written in terms of the entropy function H:

$$I(X; \hat{X}) = H(X) - H(X|\hat{X}).$$

The following result arrives easily, because we have, in effect, fixed the joint probability of X and $\hat{X}$, thus rendering the minimization in the rate-distortion curve redundant.

Theorem 1.1 (Rate-distortion constrained by α): Let the system be constrained such that the fraction of mistakes

that are false negatives is the $\min(D, p/D)$ – namely that $f(0,1) = \min(\alpha D, p)$ – where $\alpha \in [0,1]$ is a parameter to tune the amount of false negatives.

When $\alpha \in [1 + (p-1)/D, p/D]$, the rate-distortion is:

$$\begin{aligned} R(D) = H(p) + H(1 + (2\alpha - 1)D - p) + \alpha D \log \alpha D \\ + (p - \alpha D)\log(p - \alpha D) + (1-\alpha)D \log((1-\alpha)D) \\ + (1 - p - (1-\alpha)D)\log(1 - p - (1-\alpha)D) \end{aligned} \quad (4)$$

and 0 otherwise.

Proof: Consider, firstly, the case $\alpha < 1 + (p-1)/D$. In this mode of operation, independently picking the coder output as 0 with probability $\frac{\alpha D}{p}$, and as 1 otherwise, achieves a probability of a mistake not greater than D without needing to encode anything ($R(D) = 0$). It also ensures that the probability of a false negative is αD as required. What remains to be shown is that the total distortion is less than or equal to D, i.e. that

$$\alpha D + (1 - \frac{\alpha D}{p})(1-p) \leq D$$

which is true if and only if

$$\alpha D \leq \frac{D + p + 1}{2 - 1/p}.$$

This is true since $\alpha D \leq D + p + 1$ and $p \leq 1$.

Consider now the case where $\alpha \geq p/D$. Then, using a coder with output always 0, obtains no false positives and a probability of a false positive p. This coder does what is required since $p \leq \alpha D \leq D$.

The remaining case is when $\alpha \in [1 + (p-1)/D, p/D]$. The distortion is the sum of the probabilities of a wrong answer,

$$D = E[d(X, \hat{X})] = f(1,0) + f(0,1) = \alpha D + f(0,1),$$

so $f(0,1) = (1-\alpha)D$.

It also follows that $f(1,1) = p - \alpha D$ as $p = \mathsf{P}(X = 1) = f(1,0) + f(1,1)$. Similarly, $1 - p = \mathsf{P}(X = 0) = f(0,0) + f(0,1)$, implies that $f(0,0) = 1 - p - (1-\alpha)D$. The joint probability is thus completely specified:

$$f(x, \hat{x}) = \begin{bmatrix} 1 - p - (1-\alpha)D & (1-\alpha)D \\ \alpha D & p - \alpha D \end{bmatrix}.$$

The rate-distortion curve is then

$$\begin{aligned} R(D) = I(X; \hat{X}) = H(X) - H(X|\hat{X}) \\ = H(p) - \mathsf{P}(\hat{X}{=}0) H(X|\hat{X}{=}0) \\ - \mathsf{P}(\hat{X}{=}1) H(X|\hat{X}{=}1). \end{aligned}$$

Now,

$$\begin{aligned} -H(X|\hat{X} = 0) = \mathsf{P}(X = 0|\hat{X} = 0) \log \mathsf{P}(X = 0|\hat{X} = 0) \\ + \mathsf{P}(X = 1|\hat{X} = 0) \log \mathsf{P}(X = 1|\hat{X} = 0) \end{aligned}$$

and therefore

$$\begin{aligned} -\mathsf{P}(\hat{X} = 0) H(X|\hat{X} = 0) = f(0,0) \log \frac{f(0,0)}{\mathsf{P}(\hat{X} = 0)} \\ + f(1,0) \log \frac{f(1,0)}{\mathsf{P}(\hat{X} = 0)}. \end{aligned} \quad (5)$$

Similarly,

$$\begin{aligned} -\mathsf{P}(\hat{X} = 1) H(X|\hat{X} = 1) = f(0,1) \log \frac{f(0,1)}{\mathsf{P}(\hat{X} = 1)} \\ + f(1,1) \log \frac{f(1,1)}{\mathsf{P}(\hat{X} = 1)}. \end{aligned}$$

Combining this and (5), along with the facts that $\mathsf{P}(\hat{X} = 0) = f(0,0) + f(0,1)$ and $\mathsf{P}(\hat{X} = 1) = f(0,1) + f(1,1)$, it is easy to see that (4) follows. ∎

The goal is to guarantee a false negative rate of α. If the focus were on ensuring a fixed fraction of mistakes that are false positives, there would only be a minor change in the outlier cases.

TODO:check the source-coding theorem holds here

We now derive distortion (i.e. the marginal probability of a false positive) when a Bloom filter is used, under the assumption of perfect random hash functions. **TODO:Comment on general dependence difficulties in n-k model proofs**.

Theorem 1.2 (Bloom distortion given Bernoulli input): Let the Bloom filter use k independent hash functions, the size of the global set G be n, and the size of the bloom filter be m bits. Using this Bloom filter, the probability of a mistake for the ith element in G is

$$\mathsf{P}(X_i \neq \hat{X}_i) = (1-p)\left(1 - f^{n-1}\right)^k. \quad (6)$$

where $f = 1 - p + p\,(1 - 1/m)^k$.

Proof: The Bloom filter ensures no false negatives so $\mathsf{P}(X_i \neq \hat{X}_i) = \mathsf{P}(X_i = 0, \hat{X}_i = 1)$. We first derive the conditional probability $\mathsf{P}(\hat{X}_i = 1 | X_i = 0)$. Suppose b be a bit set in the Bloom filter. Let H_l be the event that hash function l doesn't set bit b with element i as input and F_j the event that b is not set by the 'actions' of jth element of G, i.e.

$$F_j = \{X_j = 0\} \cup (\{X_j = 1\} \cap H_1 \cap H_2 \cap \cdots \cap H_k).$$

Then,

$$\mathsf{P}(F_j) = 1 - p + p\,(1 - 1/m)^k = f.$$

The event that b is not set is then $B = F_1 \cup F_2 \cup \cdots \cup F_n$. The F_j are independent events so $\mathsf{P}(B|X_i = 0) = f^{n-1}$ and thus,

$$\mathsf{P}(\hat{X}_i = 1 | X_i = 0) = \mathsf{P}(\text{all } k \text{ bits are turned on}) = \left(1 - f^{n-1}\right)^k.$$

It then follows that $\mathsf{P}(X_i = 0, \hat{X}_i = 1) = (1-p)\left(1 - f^{n-1}\right)^k$. ∎

TODO:Can we approximate this like people usually do?