OMASS: One Memory Access Set Separation

Michael Mitzenmacher, Pedro Reviriego, and Salvatore Pontarelli

Abstract—In many applications there is a need to identify to which of a group of sets an element x belongs, if any. For example, in a router, this functionality can be used to determine the next hop of an incoming packet. This problem is generally known as Set Separation and has been widely studied. Most existing solutions make use of hash based algorithms, particularly when a small percentage of false positives is allowed. A known approach is to use a collection of Bloom filters in parallel. Such schemes can require several memory accesses, a significant limitation for some implementations. We propose an approach using Block Bloom Filters, where each element is first hashed to a single memory block that stores a small Bloom filter that tracks the element and the set or sets the element belongs too. In a naïve solution, when an element x in a set S is stored, it necessarily increases the false positive probability for finding that x is in another set T. In this paper, we introduce our One Memory Access Set Separation (OMASS) scheme to avoid this problem. OMASS is designed so that for a given element x, the corresponding Bloom filter bits for each set map to different positions in the memory word. This ensures that the false positive rates for the Bloom filters for element x under other sets are not affected. In addition, OMASS requires fewer hash functions compared to the naïve solution.

1 Introduction

In many computer and networking applications, one needs a data structure that can identify if an element belongs to any of a group of s sets. For example, in routing such a structure can be used to determine the outgoing port of a packet [1]. The problem is known as Set Separation. In some applications, a small probability of a false positive may be acceptable, if allowing false positives reduces the implementation complexity or improves performance. Bloom filters [2], [3] are a popular choice of data structure for similar problems when false positives are acceptable. A Bloom filter provides a data structure for approximate membership for a single set, where by approximate membership we mean there is a small chance of a false positive. If one Bloom filter is used for each of the s sets, this yields an approximate data structure for Set Separation, as each filter can be checked to determine which sets an element belongs to.

A check operation in a Bloom filter generally requires several memory accesses. To improve performance, several techniques that reduce the number of memory accesses have been proposed. For example, the Block Bloom filter [4] uses a first hash to select a block and then places all the bits to be checked inside that block. When the size of the block is that of one memory word, a check can be completed in one memory access [5]. Similarly, the Shifting Bloom filter [6] uses a set of hash functions to determine the positions of half the bits to access and then uses offsets from those bits to access the rest of the bits. This also reduces the number of memory accesses. Other structures, such as TinySet [7], have been proposed to perform approximate set membership checks in one memory access. In TinySet, fingerprints of the elements are encoded onto a memory word using a hash function. The encoding supports variable length fingerprints so that the scheme can adjust to different number of elements per memory word. Similar ideas utilizing variable length fingerprints and bit reassignment appeared previously in this context [8]. For Set Separation the membership check has to be done against s sets and therefore a straightforward implementation using a Bloom filter or a TinySet for each set could require at least s memory accesses. Another option is to use more complex schemes such as SetSep [9] that implement set separation, but SetSep again requires more than one memory access. The same applies to other options proposed for multiple set membership testing like Combinatorial Bloom filters [10] or the Bloom Trees [11] that require a large number of memory accesses. Reducing the number of memory accesses is key to improve performance in systems that use a single memory to store all the data. In specially designed system it may be possible to use several memories, for example one for each of the s sets, that can be accessed in parallel. However, using several memories increases the cost and complexity of the design.

In this paper, we present One Memory Access Set Separation (OMASS) to enable efficient implementation of Set Separation with a single access to memory. A primary further benefit of OMASS is its simplicity for implementation, which allows efficient standalone hardware implementations.

- *M. Mitzenmacher is with Harvard University, 33 Oxford Street, Cambridge, MA 02138, USA.*
  E-mail: michaelm@eecs.harvard.edu
- *P. Reviriego is with Universidad Antonio de Nebrija, C/ Pirineos, 55, E-28040 Madrid, Spain.*
  E-mail: previrie@nebrija.es
- *S. Pontarelli is with CNIT (National Inter-University Consortium for Telecommunications), Via del Politecnico 1, 00133, Rome, Italy.*
  E-mail: salvatore.pontarelli@uniroma2.it

Manuscript received November 25, 2015; revised February 11, 2016.

2 Problem definition and naïve solution

We formally describe the set separation problem. We have a universe U, a set of elements $X \subset U$, and a collection of sets $S_1, S_2, \dots, S_s$, where each element $x \in X$ is an element of one or more of the sets S_i. Each element in $U - X$ is not a member of any set. Let us define $c(x,i)$ so that $c(x,i) = 1$ if the element is in the set S_i and zero otherwise. Then the set separation problem is to provide a data structure that

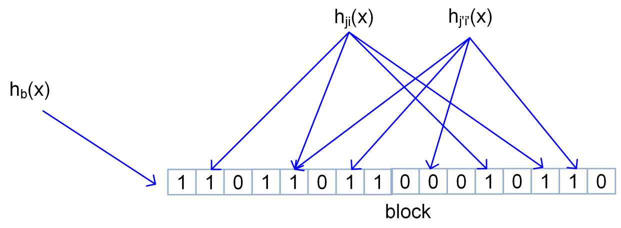

Fig. 1. Illustration of the naïve scheme.

allows us to compute, given an element x in U, the values of i for which $c(x,i) = 1$. For approximate set separation, we allow a small percentage of false positives. That is, we allow a structure that computes a function $c'(x,i)$, where $c'(x,i) = 1$ whenever $c(x,i) = 1$ (no false negatives), but when $c(x,i) = 0$, there is a small probability that $c'(x,i) = 1$ (false positives). Our goal is to design a data structure for this problem that works in one memory access. This is interesting for implementations that use external memory on which the access to the memory is a performance bottleneck. As it turns out, our data structure will also be especially effective in determining whether x is in S_i, given x and i. It is important to emphasize that there are two types of false positives : traditional and stored. A traditional false positive may give that $c(x,i) = 1$ when $x \in U - X$, that is the element is not in any set. Alternatively, a stored false positive may give for some $x \in X$ that $c(x,j) = 1$ when $c(x,j) = 0$ but $c(x,i) = 1$ for some $i \neq j$, that is an element appears to be in a set beyond the one it is in. The importance of each type depends on the specific application. For example, approximate set separation has been proposed to determine the next hop of an incoming packet in a router [1]. In this case, most packets will match a route and therefore will have an associated set. The few packets that do not have a defined route but yield the first type of false positive will eventually be dropped downstream, slightly increasing the Internet background radiation[1]. The effect of the second type of false positive is to increase the worst case latency of packets for which more than one output port is selected. Such packets must undergo more detailed analysis of some form, or may temporarily take an incorrect route toward its destination.

From the description, a natural seeming solution for approximate set separation would be to use a group of s Bloom filters. A Bloom filter is a simple data structure for approximate set membership that uses a table of bits, with all bits initially set to 0. To insert an element x, k hash functions $h_1, \ldots, h_k$ are computed and the bits with those positions in the table are set to 1. Conversely, to check if an element is present, those same positions are accessed and checked; if all of them are 1, the element is assumed to be in the set, and if any position is 0, the element is known not to be in the set. Note that there may be false positives (but not false negatives).

In particular, we could use a collection of s Block Bloom filters [4], for which the block size is that of a memory word [5]. To insert an element x in the j-th set, in the j-th Block Bloom Filter a block is selected using a hash function $h_{jb}(x)$, and then the selected block is used as a Bloom filter using k bit selection hash functions $h_{j1}, \ldots, h_{jk}$. When the size of the block is the same as a memory word, the check for an element in a filter requires only one memory access. However, in our case, as we need to check a group of s distinct Block Bloom Filters it would seem that s memory accesses are required.

A natural improvement is to have all s Block Bloom Filters use the same blocking and share memory. The naïve solution of this form uses s Block Bloom Filters such that the block selection hash function $h_b(x)$ is the same for all of them, and the bit selection hash functions $h_{j1}, \ldots, h_{jk}$, where h_{ji} is the i-th hash function for the j-th set, are different and chosen independently. This is illustrated on Figure 1. With this scheme, all s filters can be checked with a single memory access.

Inserting an element x for a set j increases the false positive probability for any other element y that hashes to that block, for all possible sets. However, potentially more problematic for the approach above is that the insertion of element x on filter j contributes to the false positive probability when considering whether x itself is in another set. Moreover, the effect will be noticeable; using a Block Bloom Filter, where the size of the block is a memory word, the number of elements stored in each word will be small to achieve a low false positive rate, and the additional element x will have substantial effect on the false positive rate for x in other sets.

We present the mathematics that elucidate why this is the case. The false positive rate of Block Bloom filters has been recently studied [12] and can be expressed as:

$$fpr = \sum_{z=0}^{n} \binom{n}{z} \cdot \left(\frac{1}{l}\right)^{z} \cdot \left(1 - \frac{1}{l}\right)^{n-z} \cdot \left(\frac{w!}{w^{k(z+1)}} \sum_{i=1}^{w} \sum_{j=1}^{i} (-1)^{i-j} \frac{j^{kz} j^{k}}{(w-i)! j! (i-j)!} \right) \quad (1)$$

where n is the number of elements stored in the filter, l the number of blocks, w the size of the blocks, and k the number of bit selection hash functions used. The last term in parentheses is the false positive rate for a Bloom filter of size w in which z elements have been stored [13], [14]. For the naïve solution, the same false positive rate would be obtained for elements that are not present in any of the s filters. However, for elements that are in one of the sets, the false positive probability for that element for any other set will be:

$$fpr|_{stored} = \sum_{z=0}^{n-1} \binom{n-1}{z} \cdot \left(\frac{1}{l}\right)^{z} \cdot \left(1 - \frac{1}{l}\right)^{n-1-z} \cdot \left(\frac{w!}{w^{k(z+2)}} \sum_{i=1}^{w} \sum_{j=1}^{i} (-1)^{i-j} \frac{j^{k(z+1)} j^{k}}{(w-i)! j! (i-j)!} \right) \quad (2)$$

That is, the number of elements present in a block will be at least one as we know that x is stored there. This can

1. Internet background radiation consists of data packets on the Internet which are addressed to unreachable destinations or are created by unsolicited network control messages, or are the result of port scans and worm activities.

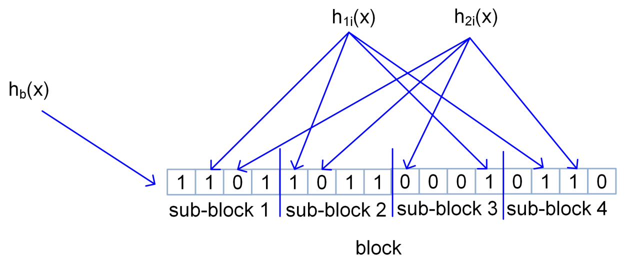

Fig. 2. Illustration of OMASS.

have a significant effect on the false positive rate when the block size w is small.

3 One Memory Access Set Separation (OMASS)

To improve the naïve solution, the One Memory Access Set Separation scheme arranges so that for an element x, the k bits selected for x for each possible set are disjoint. This avoids the problem of an element x causing false positives with itself. While there are many approaches for this, we suggest the following for when the number of sets s is at most w/k. We divide each block into k sub-blocks, and associate the i-th hash function with the i-th sub-block, so that now each hash function selects a random bit from its associated sub-block. That is, each sub-block will have size $b = w/k$, and each hash function takes values $0, \ldots, b-1$. For the first set, we use standard hash functions $(h_{11}, \ldots, h_{1k})$, but the rest of the sets use hash functions derived from those as follows: for the j-th set, $h_{ji}(x) = (h_{1i}(x) + (j-1)) \bmod b$. This construction ensures by design that $h_{ji}(x) \neq h_{j'i'}(x)$ when s is equal or smaller than b. Therefore the insertion of x in one set does not affect the false positive rate when checking x on the other $s-1$ sets. As an example, for $w = 64$ and $k = 4$, $b = 16$ and therefore up to 16 sets can be separated. This is illustrated on Figure 2. The false positive rate is the same regardless of whether the element is stored on one of the filters or not and is given by:

$$fpr = \sum_{z=0}^{n} \binom{n}{z} \cdot \left(\frac{1}{l}\right)^{z} \cdot \left(1 - \frac{1}{l}\right)^{n-z} \cdot$$
$$\left(\frac{b!}{b^{(z+1)}} \sum_{i=1}^{b} \sum_{j=1}^{i} (-1)^{i-j} \frac{j^z i}{(b-i)! j! (i-j)!} \right)^{k} \quad (3)$$

The main change is that the last term is now the false positive for a Bloom filter of size b that uses a single bit selection hash function to the power of k. This is equivalent to a Bloom filter that uses k tables instead of one. It is well known that its false positive rate increases slightly due to partitioning for small tables but is asymptotically the same as that of a Bloom filter that uses a single table when the size of the tables is large [3]. Therefore, when the tables are small the false positive rate will be slightly larger than that of the naïve solution for elements that are not present in any set. However, for elements that are present in a set, the false positive rate will be significantly smaller than in the naïve scheme. Finally, it is worth mentioning that OMASS requires only k hash functions compared to the $k \times s$ hash functions needed in the naïve scheme due to the one memory access requirement. If more memory accesses are permitted, each set can use a different block selection function and thus the same bit selection hash functions can be used for all the sets.

4 Evaluation

We have simulated and compared the proposed scheme with the naïve solution. To do so, the false positive rate for a single set membership check is used. This makes the results independent of the number of sets used and allows a direct comparison with equations (1) to (3). However, we recall that in a set separation application, for each element x, there are s set membership checks corresponding to the s possible sets that x may be in. Therefore the false positive rate per element for the application would be approximately s times the Bloom filter false positive probability for an element not present in any set and $s-1$ times this probability for an element present in one set when the probability is much smaller than one. In the simulations, we used block sizes of 64 and 512 bits, which correspond to typical values of word sizes and cache line sizes in modern computers (Intel i3/i5/i7 CPUs have a last level cache of 512 bits [15]). The total memory was 1Mbit and therefore in the first case there were 16K blocks and in the second 2K blocks. In both cases, four hash functions were used in the Bloom filters for the naïve solution. Correspondingly, four sub-blocks were used for OMASS. This enables the use of up to 16 and 128 sets for the 64 and 512 bits blocks.The tests were done for 16 sets in both cases. Therefore, 64 hash functions are needed in the naïve scheme and only 4 for OMASS. We use a range of 10 to 80 bits per element in our simulations so that we achieve small false positive rates suitable for many applications. To evaluate the false positive rate elements were inserted until the number of bits per element value was achieved. Then searches for elements not present and for elements present in another set were done. For elements present on a set, all the elements on the filter were tested. The process was repeated as many times as needed to test one hundred million elements for each configuration. The results for a word size of 64 bits are shown in Figure 3.a) for searches on an element x that is not present in any set. In this case, the naïve scheme has a slightly lower false positive rate. This is due to the use of sub-blocks as explained before. Figure 3.b) shows the results for searches on an element x in set j when element x is present in another set. It can be observed that OMASS provides a significant reduction of the false positive rate, especially when the number of bits per element is high. Finally, in both cases the simulation results match the theoretical ones predicted by equations (1) to (3). The results for a word size of 512 bits are shown in Figures 4.a) and 4.b). In this case, the differences in false positive rates are almost negligible for elements that are not present in a set. For elements present in a set, the benefits of using OMASS are also reduced but still significant for low false positive rates. As a summary, the benefits of OMASS are larger for small block sizes. Finally, it is worth mentionig that the optimal value for the number of hash functions k

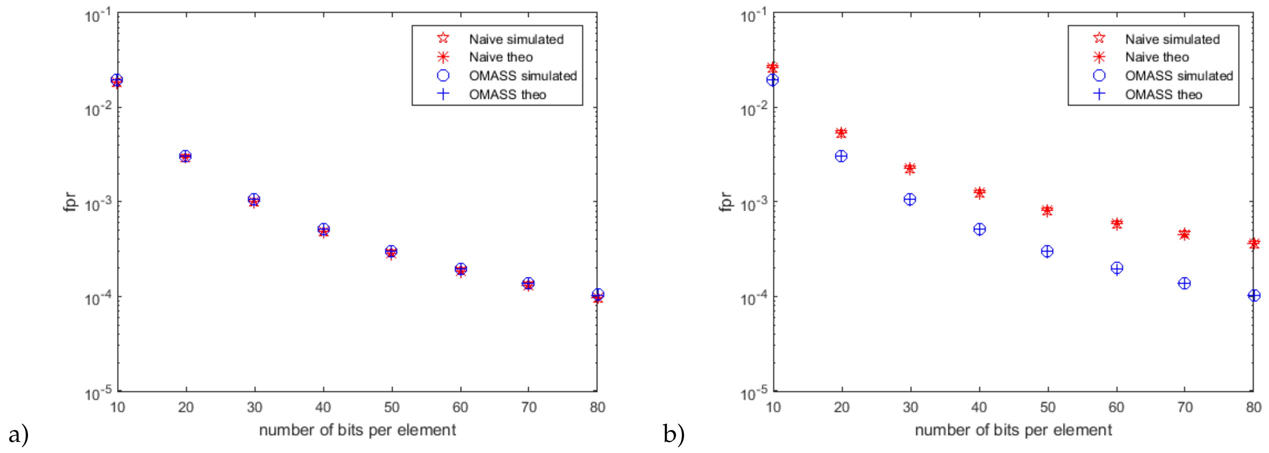

Fig. 3. a): False Positive Rate versus Number of Bits per Element for searches on elements not present in any set for a block size of 64 bits. b): False Positive Rate versus Number of Bits per Element for searches on elements present in another set for a block size of 64 bits.

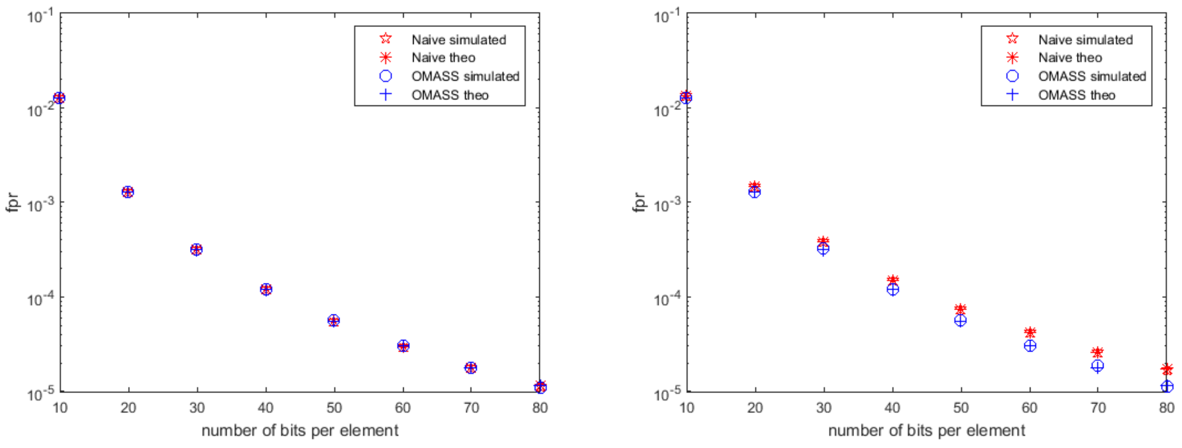

Fig. 4. a): False Positive Rate versus Number of Bits per Element for searches on elements not present in any set for a block size of 512 bits. b):False Positive Rate versus Number of Bits per Element for searches on elements present in another set for a block size of 512 bits.

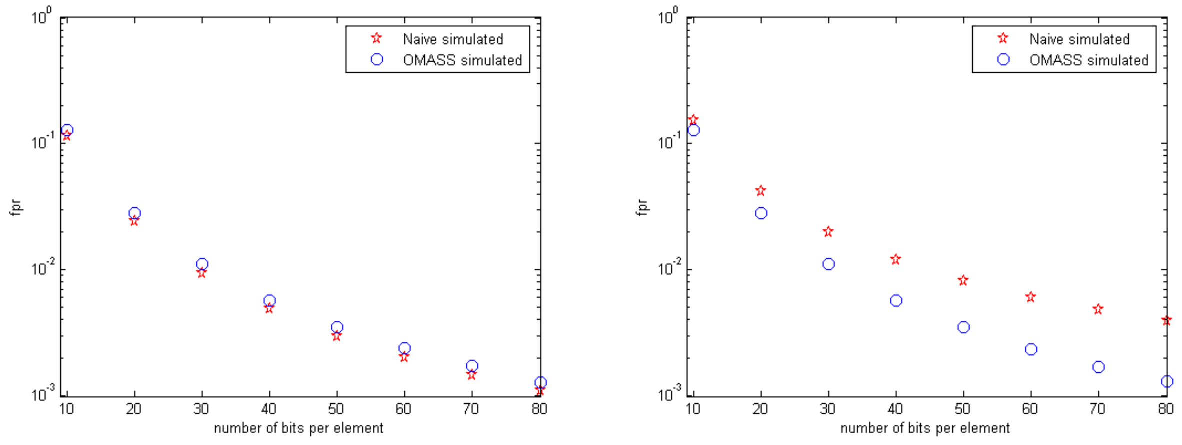

Fig. 5. Elements stored in two sets. a): False Positive Rate versus Number of Bits per Element for searches on elements not present in any set for a block size of 64 bits. b): False Positive Rate versus Number of Bits per Element for searches on elements present in another two sets for a block size of 64 bits.

for a given number of bits per element can be computed by evaluating equation 3 for the different values of k and selecting the one that gives the lowest value.

The OMASS scheme can also be used for applications on which an element may be present in more than one set. In that case, the effect of reducing the false positives caused by the same element being present in several sets will be larger, as in the naïve scheme each insertion of an element into a set affects the false positive probability of that element for the remaining sets. On the other hand, there will also be an increase on the false positive rate for other elements as each element that is inserted in more than one set will on average set more bits to one than in the naïve scheme. This is illustrated on Figures 5.a) and 5.b) for the case on which elements are always inserted in exactly two of the sets for blocks of 64 bits. In a general case, the false positive rate of OMASS when elements are inserted in several sets depends on the distribution of the sets an element is stored in. An analysis of this general case is mathematically complex and beyond the scope of this paper.

5 Conclusions

The number of memory accesses is commonly a performance bottleneck for hash-based data structures, including those for Set Separation. Memory access times are generally orders of magnitude lager than the processor clock cycle, while parallel memory accesses increase cost and complexity. In principle, a Block Bloom filter can be used to implement set separation with one memory access. However, in the naïve solution the insertion of an element in one set can induce false positives for the same element on the other sets. This increases the false positive rate and introduces a significant penalty when the memory word size is small. This paper has presented One Memory Access Set Separation (OMASS), an alternative technique to implement set separation in one memory access. In OMASS, the insertion of an element in a set does not affect the false positive rate for that element on the rest of the sets. This reduces significantly the false positive rate for elements that are stored in one of the sets when the word size is small. OMASS also reduces the number of hash functions needed to insert or check for an element and can be implemented efficiently. The only disadvantage of OMASS is a very small increase in the false positive rate when checking for elements that are not present on any of the sets.

Acknowledgment

Michael Mitzenmacher is supported in part by NSF grants CCF-1535795, CCF-1320231, and CNS-1228598. Pedro Reviriego is partially supported by the excellence network Elastic Networks TEC2015-71932-REDT funded by the Spanish Ministry of Economy and Competitiveness. Salvatore Pontarelli is partially supported by the European Union in the context of the BEBA project (Grant Agreement: 644122).

References

[1] M. Yu, A. Fabrikant and J. Rexford, BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations, Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, pp. 313-324, 2009.

[2] B. Bloom, Space/time tradeoffs in hash coding with allowable errors, Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

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

[4] U. Manber and S. Wu, An algorithm for approximate membership checking with application to password security, Information Processing Letters, vol. 50, no. 4 pp. 191-197, May 1994.

[5] Y. Qiao, T. Li, and S. Chen, Fast Bloom Filters and Their Generalization, IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 1, pp. 93-103, Jan. 2014.

[6] T. Yang, A. X. Liu, M. Shahzad, Y. Zhong, Q. Fu, Z. Li, G. Xie and X. Li A Shifting Bloom Filter Framework for Set Queries, http://export.arxiv.org/pdf/1510.03019v1.

[7] G. Einziger and R. Friedman: TinySet - An Access Efficient Self Adjusting Bloom Filter Construction, Proceedings of the 24th International Conference on Computer Communication and Networks, pp. 1-9, 2015.

[8] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. Bloom Filters via d-Left Hashing and Dynamic Bit Reassignment (Extended Abstract), Proceedings of the 44th Annual Allerton Conference, pp. 877-883, 2006.

[9] D. Zhou, B. Fan, H. Lim, D. G. Andersen, M. Kaminsky, M. Mitzenmacher, R. Wang and A. Singh, Scaling Up Clustered Network Appliances with ScaleBricks, Proceedings of ACM SIGCOMM, pp. 241-254, 2015.

[10] F. Hao, M. Kodialam, T. Lakshman, and H. Song, Fast Dynamic Multiple-Set Membership Testing Using Combinatorial Bloom Filters, IEEE/ACM Transactions on Networking, vol. 20, no. 1, pp. 295-304, Feb. 2012.

[11] M. Yoon, J. Son, and S. Shin, Bloom Tree: A Search Tree Based on Bloom Filters for Multiple-Set Membership Testing, Proc. of IEEE INFOCOM'14.

[12] P. Reviriego, K. Christensen, J.A. Maestro, A Comment on 'Fast Bloom Filters and Their Generalization', IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 1, pp. 303-304, Jan. 2016.

[13] 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. 210213, Oct. 2008.

[14] K. Christensen, A. Roginsky, and M. Jimeno, A New Analysis of the False-Positive Rate of a Bloom Filter, Information Processing Letters, vol. 110, no. 21, pp. 944-949, Oct. 2010.

[15] O. Lempel, "'2nd generation Intel core processor family: Intel core i7, i5 and i3"', Hot Chips. 2011.