Weighted Bloom Filter

Jehoshua Bruck* Jie Gao† Anxiao (Andrew) Jiang‡

* Department of Electrical Engineering, California Institute of Technology. bruck@paradise.caltech.edu.
† Department of Computer Science, Stony Brook University, Stony Brook, NY 11794. jgao@cs.sunysb.edu.
‡ Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. ajiang@cs.tamu.edu.

***Abstract*— A Bloom filter is a simple randomized data structure that answers membership query with no false negative and a small false positive probability. It is an elegant data compression technique for membership information and has broad applications. In this paper, we generalize the traditional Bloom filter to *Weighted Bloom Filter*, which incorporates the information on the query frequencies and the membership likelihood of the elements into its optimal design. It has been widely observed that in many applications, some popular elements are queried much more often than the others. The traditional Bloom filter for data sets with irregular query patterns and non-uniform membership likelihood can be further optimized. We derive the optimal configuration of the Bloom filter with query-frequency and membership-likelihood information, and show that the adapted Bloom filter always outperforms the traditional Bloom filter. Under reasonable frequency models such as the step distribution or the Zipf's distribution, the improvement of the false positive probability of the weighted Bloom filter over that of the traditional Bloom filter has been evaluated by simulations.**

Keywords: Bloom Filter, Membership Query, Combinatorics

I. Introduction

A Bloom filter [3] is a compact randomized data structure for representing a set in order to support membership queries. It encodes the elements in a set S, called *members*, in a much larger universe U, $S \subseteq U$. In short, a Bloom filter is an m-bit array such that each member in S is hashed to k positions (bits) in the array, and those bits are set to '1'. A membership query takes an input element and checks the k hashed positions. If all those bits are '1', then the query returns 'yes'. A Bloom filter has no false negative but a low false positive probability. In particular, when $k = \ln 2 \cdot m/n$, with $n = |S|$ being the number of members, the false positive probability achieves its minimum value $1/2^k$. With only a constant number of bits for each member on average, a Bloom filter answers membership queries with a small false positive probability.

The space efficiency of the Bloom filter makes it very appealing in network applications [5]. Although theoretically a hash table supports membership queries and yields an asymptotically vanishing probability of error by using only $\Theta(\log n)$ bits per element, the Bloom filter attracts a lot of interest in practice, especially in systems that need to share information about their available resources. In a typical scenario, e.g., the Web cache sharing [14], user queries for desired documents are directed to proxies instead of the original Web server, in order to reduce traffic and alleviate network bottlenecks. Upon a miss at a local cache, a proxy wants to check whether other proxies have the desired document before sending out requests to fetch the document. A Bloom filter is adopted to summarize the contents of each proxy for two reasons – a small false positive probability is tolerable, and the size of the Bloom filter is much smaller compared with the list of full URLs or documents. With a similar spirit, Bloom filters are used in many network applications such as file search in P2P networks [17], packet classification [8], trajectory sampling [13], en-route filtering of false data in the network [23], fast hash table lookup [22] and many others [19]. Motivated by these applications, the traditional Bloom filter, invented by Bloom [3], has been augmented in various ways [9], [10], [14], [17], [18], [20].

The Bloom filter takes a hidden assumption that all elements in the universe are viewed and treated identically, which is the best we can assume without further information of the query frequency distribution or their membership likelihood (likelihood of being a member). Under this assumption it can be shown that the Bloom filter is space-wise asymptotically optimal with a fixed false positive probability and zero false negative probability [5], [7]. However, it commonly occurs in practice that elements do not get queried evenly — popular elements are queried much more often than unpopular elements. In the measurement of the Internet traffic pattern, it is observed that traffic flow is highly skewed and concentrates heavily on popular files [4], [15]. If the query frequency or the membership likelihood is not uniform over all the elements in the universe, the traditional configuration of the Bloom filter does not give the optimal performance. In other words, the Bloom filter can be further optimized if we know the query frequency or/and the membership likelihood distribution of the elements in the universe. In many real systems, statistics about the traffic flow such as frequencies of elements, top k categories to which the most elements belong, can be and are already being monitored [2], [11], [12], [16]. Thus we can use such statistics and configure the Bloom filter accordingly. Indeed, information such as query frequencies has already been used to improve the performance of caching structures [4].

In this paper, we give the optimal configuration of the Bloom filter for items with varying query frequencies and membership likelihood. We call such a Bloom filter the *weighted Bloom filter*. In particular, each element $e \in U$ is assigned k_e hash functions, where k_e depends on its query frequency and its likelihood of being a member. Therefore, each non-member element has a different false positive probability. The average false positive probability is actually a

weighted sum over the query frequencies of the elements in the universe. Intuitively, an element is assigned more hash functions if its query frequency is high and its chance of being a member is low. When the query frequencies and the membership likelihoods are the same for all elements in the universe, the weighted Bloom filter degenerates to the traditional Bloom filter. Thus our model is a generalization and further optimization of the traditional Bloom filter. We also evaluate the performance gain of the weighted Bloom filter over the traditional one, under various frequency distributions such as step-like functions or Zipf distributions [4], [21], [24]. We observe that the improvement in the false positive probability is significant under reasonable frequency models.

The weighted Bloom filter can be gracefully integrated with existing frequency estimators [2], [11], [12], [16], which usually maintain a set of 'hot' categories. Such rough frequency estimations are represented by popularity buckets. All the elements in a 'hot' category share the same frequency estimation. The elements are classified into categories by attributes that are easily checkable, such as the web files of a popular website or the songs of a popular singer. Similarly the membership likelihood can also be estimated using such a category-based technique. The estimated query frequency and membership likelihood of an element determine the number of hash functions used for it in the Bloom filter. The efficient estimation of item popularity and membership likelihood justifies the practicality of the weighted Bloom filter. We discuss more on the implementation issues in Section III.

Due to the space limitation, we skip some proofs and detailed analysis in this paper. Interested readers please refer to the full version [6].

II. An Overview of Bloom Filter

We first give a quick introduction to the traditional Bloom filter [3]. A Bloom filter is used to represent a set of elements S in a big universe U. The number of elements in S (called *members*), $n = |S|$, is usually much smaller than the size of the universe, $N = |U|$. That is, $S \subseteq U$ and $n \ll N$. A Bloom filter is an m-bit array and uses k independent uniformly random hash functions $\{h_i \,|\, i = 1, 2, \cdots, k\}$ which map to range $\{1, \cdots, m\}$. For each element x in S, the $h_i(x)$-th bits in the Bloom filter, $i = 1, \cdots, k$, are set to '1'. For a membership query about whether an element y is in the set S, the answer is 'yes' if all the bits $h_i(y)$ are '1' and 'no' otherwise.

The query to a Bloom filter has no false negative – if an element is a member, the query always returns 'yes'. There is a small false positive probability. The query to a non-member may get an answer 'yes' if the bits corresponding to its hashed positions are all '1' accidentally. Assuming that the hash functions are perfectly random, the probability of a false positive for a non-member element can be calculated in a simple way. After all the n members are hashed to the Bloom filter, the probability that a specific bit remains '0' is simply

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

Therefore, the probability of a false positive is

$$P_{FP} = \big(1 - (1 - \frac{1}{m})^{kn}\big)^k \approx (1 - e^{-kn/m})^k = (1 - p)^k.$$

To obtain the best performance of the Bloom filter, we would like to choose k that minimizes the false positive probability. Intuitively, more hash functions for an element will increase the chances of finding a '0' for a non-member but will also increases the total number of '1's in the filter. The optimal number of hash functions can be obtained by taking the derivative of P_{FP} to be zero. This reveals that the Bloom filter has the best performance if k is set to $\ln 2 \cdot m/n$. In that case, the false positive probability is $P_{FP} = 1/2^k = 2^{-(m/n)\ln 2}$.

III. Weighted Bloom Filter

The traditional Bloom filter does not differentiate the query frequencies of different elements or their *a priori* likelihoods of being members. In this section, we study *Weighted Bloom Filter*, the Bloom filter optimized based on the elements' query frequencies and their probabilities of being members. In many applications, query frequencies and membership likelihoods are estimated or collected with well developed techniques [2], [11], [12], [16]. Statistics of such information are maintained, especially for a set of 'hot' categories. As will be shown later, such data is useful in optimizing the optimal configuration of a Bloom filter. The performance improvement is remarkable even with rough estimations of query frequencies and membership likelihoods represented by categories. We will present the optimal configuration for the weighted Bloom filter, and show that it generalizes the traditional Bloom filter.

Same as the traditional Bloom filter, a weighted Bloom filter uses m bits to record the n member elements in a set S. ($|S| = n$.) There are N elements in total in the universe, where $N \gg n$. We assume that the probability of an element e's being a member, denoted by x_e, is independent of all the other elements. We introduce the indicator random variable X_e for each $e \in U$ as follows:

$$X_e = \begin{cases} 1 & , \quad \text{if} \quad e \in S \\ 0 & , \quad \text{if} \quad e \notin S \end{cases}$$

We use $E\{\cdot\}$ to denote the expectation of a random variable. $E\{X_e\} = x_e$.

The basic rationale for designing the weighted Bloom filter is as follows. The filter's false positive probability is a weighted sum of each individual element's false positive probability, where the weight corresponding to an individual element is positively correlated with the element's query frequency and is negatively correlated with the element's probability of being a member. Therefore, we would in general like to assign more hash functions to an element with a higher query frequency or with a lower probability of being a member, in order to reduce the false-positive probability of an element with a higher weight.

For each element $e \in U$, denote its query frequency by f_e. Then the probability that a query is about an element $a \in U$ equals $f_a / \sum_{e \in U} f_e$. Denote the number of hash functions used for an element e by k_e. The total number

of hash functions used for elements in S is denoted by $K = \sum_{e \in S} k_e = \sum_{e \in U} X_e \cdot k_e$. The rule of answering the membership queries is the same as before. Specifically, for an element e, we check all the k_e corresponding bits in the Bloom filter, and say 'e *is a member*' if and only if all the k_e bits are 1. So there is no false negative and possibly a small false positive probability.

All the hash functions are uniformly random hash functions. Therefore the probability that a specific bit remains '0' in the Bloom filter, denoted by p, only depends on the total number of hash functions operated on the Bloom filter. Specifically, we have:

$$p = (1 - \frac{1}{m})^K \approx e^{-\frac{K}{m}}. \quad (1)$$

To simplify the analysis, we assume in the standard way [5], [18] that each bit in the Bloom filter is set to '0' with probability p and '1' with probability $1 - p$, independently of the other bits. This is a valid simplification — with n being relative large (as in the typical setting for using a Bloom filter), the fraction of 0 bits in the Bloom filter is sharply concentrated around p, as shown in [18]. In practice, the dependency between the bits in the Bloom filter is negligible [1]. The probability of a false positive is the weighted sum of the false positive probabilities of all non-members in the universe, denoted by P_{FP}:

$$\begin{aligned} P_{FP} &= \frac{\sum_{e \in U-S} f_e (1-p)^{k_e}}{\sum_{e \in U-S} f_e} \\ &= \frac{\sum_{e \in U} (1-X_e) \cdot f_e (1-p)^{k_e}}{\sum_{e \in U} (1-X_e) \cdot f_e} \\ &= \sum_{e \in U} \frac{(1-X_e) f_e}{\sum_{i \in U} (1-X_i) \cdot f_i} \cdot (1-p)^{k_e}. \end{aligned} \quad (2)$$

Denote by r_e the normalized query frequency of $e \in U$,

$$r_e = \frac{(1 - X_e) f_e}{\sum_{i \in U} (1 - X_i) \cdot f_i}. \quad (3)$$

Then the false positive probability can be represented as

$$P_{FP} = \sum_{e \in U} r_e (1-p)^{k_e}. \quad (4)$$

Then, the expectation of the false positive probability is $E\{P_{FP}\}$. Given a fixed-sized Bloom filter of m bits, we would like to minimize the expected false positive probability by optimizing the number of hash functions assigned to each element. Our main result in this paper is to derive the best configuration of the weighted Bloom filter, as shown in the following theorem.

Theorem 3.1. *In order to minimize the expected false positive probability $E\{P_{FP}\}$ of a weighted Bloom filter, the Bloom filter should be configured as follows: Assuming that k_e, for $e \in U$, can be any real number, the number of hash functions for an element $e \in U$ is*

$$k_e = \frac{m}{n} \cdot \ln 2 + (\ln E\{r_e\} - \sum_{i \in U} \frac{x_i}{n} \cdot \ln E\{r_i\}) \cdot \frac{1}{\ln 2}; \quad (5)$$

With the above k_e for $e \in U$, the probability that a bit in the Bloom filter is '0' is

$$p = 1/2; \quad (6)$$

and the expectation of the false-positive probability is:

$$E\{P_{FP}\} = 2^{-(m/n) \ln 2} \cdot N \cdot \prod_{e \in U} E\{r_e\}^{x_e/n}. \quad (7)$$

Proof: We explain here the intuition on how to derive the best configuration of the weighted Bloom filter. For detailed derivation, please refer to the full paper [6]. Basically, we see the number of hash functions assigned to an element $e \in U$ as a variable, and seek a local optimum of $E\{P_{FP}\}$ by taking its partial derivative with respect to k_e. By solving the equations we obtain the relative ratios of k_e for each $e \in U$. While adjusting the number of hash functions assigned to individual elements, we should also scale those numbers by an appropriate factor in order to control the portion of the Bloom filter bits that are set to be '1's, because the overall false-positive probability will be hurt when the portion is too large or too small. We choose the total number of hash functions, K, to be an appropriate value so as to minimize the expected false positive probability $E\{P_{FP}\}$, and show that the local optimum found is in fact also the global optimum. □

In practice, k_e needs to be a non-negative integer. So when implementing the weighted Bloom filter, we round k_e to a nearby non-negative integer. More details on the implementation of the weighted Bloom filter will be studied in subsection III-B.

A. Generalization of Bloom Filter

In this subsection, we compare the weighted Bloom filter to the traditional Bloom filter and show that our result is a generalization and further optimization of the traditional optimal configuration. In the traditional Bloom filter [3], no knowledge about an element's query frequency or membership likelihood is assumed. Its optimal configuration is:

$$\begin{cases} p = 1/2; \\ k_e = (m/n) \ln 2 \,, \forall e \in \mathrm{U}; \\ P_{FP} = 2^{-(m/n) \ln 2}. \end{cases}$$

In the weighted Bloom filter setting, all the elements should be treated the same when no knowledge about query frequencies or membership likelihood is available. Then we can set f_e and x_e to be constants, $\forall e \in U$. More specifically, $x_e = E\{X_e\} = n/N$, where $n = |S|$ and $N = |U|$. By equation 3, we see that $E\{r_e\}$ is a constant as well for any $e \in U$. Since $\sum_{e \in U} E\{r_e\} = E\{\sum_{e \in U} r_e\} = 1$, we have that $\forall\ e \in U$, $E\{r_e\} = 1/N$. By plugging $x_e = n/N$ and $E\{r_e\} = 1/N$ for all $e \in U$ into equations 5 and 7, we obtain the formulas $k_e = (m/n) \ln 2$ and $P_{FP} = 2^{-(m/n) \ln 2}$, the same as the second and the third formulas of the traditional Bloom filter's configuration. By taking equation 6 into account, we see that the traditional optimal configuration of Bloom filter is reduced to our solution as a special case.

When the elements in U have varying query frequencies or membership likelihood, the false-positive probability of the weighted Bloom filter usually becomes better than that of the traditional Bloom filter. We use here a simple scenario for illustration. Consider the case where all the elements have identical membership likelihood, but their query frequencies

vary from element to element. Then the expected false-positive probability, as shown in equation 7, becomes:

$$\begin{aligned} E\{P_{FP}\} &= 2^{-(m/n)\ln 2} \cdot N \cdot \prod_{e\in U} E\{r_e\}^{x_e/n} \\ &= 2^{-(m/n)\ln 2} \cdot \frac{\prod_{e\in U} E\{r_e\}^{1/N}}{\sum_{e\in U} E\{r_e\}/N} \\ &\leq 2^{-(m/n)\ln 2}. \end{aligned} \tag{8}$$

The first step holds because $x_e = n/N$ for all $e \in U$, and $\sum_{e\in U} E\{r_e\} = 1$. The second step, the inequality, holds because the geometric average of $E\{r_e\}$ is no greater than the arithmetic average of $E\{r_e\}$. In fact, the inequality will become equality only if $E\{r_e\} = 1/N$ for all $e \in U$, which, in turn, leads to the requirement that $f_i = f_j$ for all $i, j \in U$. (We skip the details of the proof.) So the false-positive probability of the traditional Bloom filter always exceeds that of the weighted Bloom filter unless all the elements have the same query frequency.

B. Efficient Implementation of Weighted Bloom Filter

The optimal result we have derived does not consider the constraint that the number of hash functions assigned to each element needs to be a non-negative integer. Also, computing $E\{r_e\}$ is often non-trivial — the expression for r_e (equation 3) contains N binary random variables X_i, so there are totally 2^N disjoint cases. Below we present an efficient way to approximately compute the optimal solution.

Firstly, consider the expression for r_e as shown in equation 3. We get:

$$\begin{aligned} & E\{r_e\} \\ = \; & E\{\tfrac{(1-X_e)f_e}{\sum_{i\in U}(1-X_i)\cdot f_i}\} \\ = \; & \text{Prob}\{X_e = 0\} \cdot E\{\tfrac{(1-0)f_e}{\sum_{i\in U-\{e\}}(1-X_i)\cdot f_i + (1-0)f_e}\} + \\ & \text{Prob}\{X_e = 1\} \cdot E\{\tfrac{(1-1)f_e}{\sum_{i\in U-\{e\}}(1-X_i)\cdot f_i + (1-1)f_e}\} \\ = \; & (1-x_e) \cdot E\{\tfrac{f_e}{\sum_{i\in U-\{e\}}(1-X_i)\cdot f_i + f_e}\} \end{aligned} \tag{9}$$

When the value of $\sum_{i\in U-\{e\}}(1-X_i)\cdot f_i$ well concentrates around its expectation, $E\{r_e\}$ can be approximately computed as

$$\begin{aligned} & E\{r_e\} \\ \approx \; & (1-x_e) \cdot \tfrac{f_e}{E\{\sum_{i\in U-\{e\}}(1-X_i)\cdot f_i\} + f_e} \\ = \; & (1-x_e) \cdot \tfrac{f_e}{\sum_{i\in U-\{e\}}(1-E\{X_i\})\cdot f_i + f_e} \\ = \; & (1-x_e) \cdot \tfrac{f_e}{\sum_{i\in U-\{e\}}(1-x_i)\cdot f_i + f_e} \\ = \; & \tfrac{(1-x_e)f_e}{\sum_{i\in U}(1-x_i)f_i - (1-x_e)f_e + f_e} \\ = \; & \tfrac{(1-x_e)f_e}{x_e f_e + \sum_{i\in U}(1-x_i)f_i} \end{aligned} \tag{10}$$

Thus we first compute the term $\sum_{i\in U}(1-x_i)f_i$, which will be used in computing $E\{r_e\}$ for each $e \in U$. Equation 10 provides an efficient way to compute $E\{r_e\}$. Given $E\{r_e\}$, we can compute k_e for $e \in U$ using equation 5. Then as a heuristic, we round k_e to its nearby non-negative integer.

C. Numerical Analysis of Weighted Bloom Filter

It is commonly observed that across many scales in society and economics, human behaviors exhibit inherent characteristics. Statistics about query frequencies of Web pages [4], [21] and input to search engines [15] often revealed that user queries data are skewed where a few popular items or files are searched much more often than majority unpopular ones. Numerous studies have found that the distribution of query frequencies follows Zipf's law [24], which states that the relative probability of a request for the ith most popular item is inversely proportional to the power of i. We also study a simplified analog of the Zipf's distribution, which we denote by the step distribution where there are only two categories, the popular (or hot) set and the unpopular (cold) set. This models the case when accurate frequency distributions are not known and only a rough bucketing on the popularity of the elements is maintained. Such kind of simple bucketing can be created and updated efficiently by recent algorithms on streaming data, which usually use an extremely small amount of storage space and a few passes of the data set [2], [11], [12], [16]. We have evaluated the performance of weighted Bloom filter on queries whose frequency distributions follow Zipf's law or the like. Even with a rough estimation of the query frequency distribution, such as the 2-bucketing, the performance improvement of weighted Bloom filter over the traditional configuration is impressive.

We start with a simplified analog of the Zipf's distribution, which we call the *Step Distribution*, defined as follows. The universe U is partitioned into two subsets: a *hot set* A and a *cold set* B. The percentage of A in the universe is $\alpha = |A|/|U|$. So $0 \leq \alpha \leq 1$ and $\alpha = 1 - |B|/|U|$. Each element in B has query frequency f, while each element in A has query frequency $c \cdot f$, $c \geq 1$. The value of c shows how popular the 'hot' elements are in terms of query compared to the 'cold' elements. We assume that each element in the universe has the same likelihood of being a member.

A simple calculation shows that the number of hash functions assigned to elements in set A and B are respectively,

$$k_e = \begin{cases} (m/n)\ln 2 + (1-\alpha)(\ln c/\ln 2) & , \quad \text{if} \quad e \in A \\ (m/n)\ln 2 - \alpha(\ln c/\ln 2) & , \quad \text{if} \quad e \in B \end{cases}.$$

The ratio R of the false positive probability of the weighted Bloom filter to the false positive probability of the traditional Bloom filter is:

$$R = \frac{c^\alpha}{\alpha c + (1-\alpha)}.$$

We call $1/R$, the inverse ratio of the two false positive probabilities, the P_{FP} *improvement*. The greater the P_{FP} improvement is, the better. The above formulas assume that the number of hash functions assigned to each element, k_e, can be any real number; when we round k_e to nearby non-negative integers, the two formulas should be adjusted accordingly. When all k_e take non-negative integer values, the P_{FP} improvement corresponding to different values of α and c are illustrated in Figure 1.

In Figure 1, the vertical axis is the P_{FP} improvement. The two horizontal axes are α that varies in $[0, 1]$ and c that varies in $[1, 10000]$. We see that the P_{FP} improvement increases with c, because the performance gain of weighted Bloom filter improves when the query frequencies become more skewed.

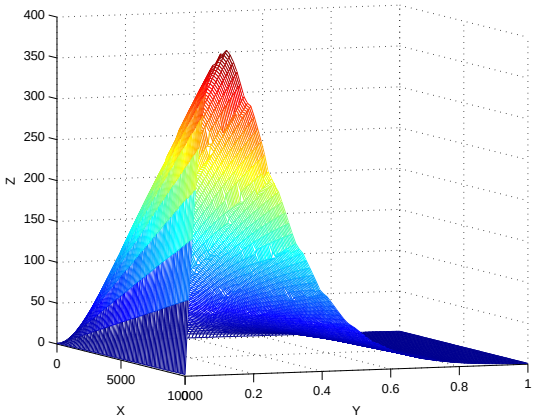

Fig. 1. P_{FP} improvement of the weighted Bloom filter, when the query frequencies are modelled with the step distribution. The vertical axis is the P_{FP} improvement. The two horizontal axes are, respectively, α that varies in $[0, 1]$ and c that varies in $[1, 10000]$. Here $m/n = 14$.

When c is fixed, the best performance improvement appears when there are neither too many nor too few hot elements, in which case the query frequencies there are more skewed. The difference in the number of hash functions assigned to hot elements and cold elements is moderate — in a typical Bloom filter configuration with $m/n = 14$, the number of hash functions assigned to hot elements and to cold elements varies between 10 and 23 and between 0 and 10, respectively. However, the improvement of the false positive probability is a lot. The maximum P_{FP} improvement here is 396.9.

We have also studied the performance of weighted Bloom filter when the query frequencies follow the Zipf distribution, or when the query frequencies have positive or negative correlations with the membership likelihoods. All these cases have natural applications, and substantial performance improvement by the weighted Bloom filter has been observed. Details can be found in the full paper [6].

IV. Conclusions

In this paper we have deepened our understanding on the well-known Bloom filter structure. When more information about the membership likelihood and query frequencies is available, we derived the optimal Bloom filter configuration and investigated efficient implementation schemes in practice. We have compared the performance of the weighted Bloom filter with the traditional Bloom filter under reasonable query frequency and membership likelihood models. For the future work, it would be interesting to evaluate the performance gain of the weighted Bloom filter on real network traces and further investigate the integration of the Bloom filter with network statistics estimators.

Acknowledgements: This work was supported in part by the Lee Center for Advanced Networking at the California Institute of Technology, and by NSF grant CCR-TC-0209042.

References

[1] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. *Random Structures and Algorithms*, 13(2):159–188, 1998.

[2] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 28–39, New York, NY, USA, 2003. ACM Press.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *Proc. IEEE INFOCOM'99*, pages 126–134, 1999.

[5] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*, pages 636–646, 2002.

[6] J. Bruck, J. Gao, and A. A. Jiang. Weighted bloom filter. Technical Report ETR072, Distributed Information Systems Group, Department of Electrical Engineering, California Institute of Technology, 2006. http://www.paradise.caltech.edu/papers/etr072.pdf.

[7] L. Carter, R. Floyd, J. Gill, G. Markowsky, and M. Wegman. Exact and approximate membership testers. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 59–65, 1978.

[8] F. Chang, W. Feng, and K. Li. Approximate caches for packet classification. In *Proceedings of INFOCOM*, volume 4, pages 2196–2207, Hong Kong, China, 2004.

[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 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 30–39, New Orleans, LA., United States, 2004.

[10] S. Cohen and Y. Matias. Spectral Bloom filters. In *Proc. SIGMOD*, pages 241–252, 2003.

[11] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 296–306, New York, NY, USA, 2003. ACM Press.

[12] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 348–360, London, UK, 2002. Springer-Verlag.

[13] N. Duffield and M. Grossglauser. Trajectory sampling with unreliable reporting. In *Proc. INFOCOM'04*, volume 3, pages 1570–1581, Hong Kong, China, 2004.

[14] 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*, 8(3):281–293, 2000.

[15] Google. Google zeitgeist – search patterns, trends, and surprises according to google. http://www.google.com/press/zeitgeist.html.

[16] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1):51–55, 2003.

[17] A. Kumar, J. Xu, and E. W. Zegura. Efficient and scalable query routing for unstructured peer-to-peer networks. In *Proc. INFOCOM'05*, Miami, Florida, U.S.A., 2005.

[18] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking*, 10(5):604–612, 2002.

[19] P. Mutaf and C. Castelluccia. Compact neighbor discovery. In *Proc. INFOCOM'05*, Miami, FL, USA, 2005.

[20] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proc. INFOCOM'02*, volume 3, pages 1248–1257, 2002.

[21] P. Rodriguez, C. Spanner, and E. W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, 9(4):404–418, 2001.

[22] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast hash table lookup using extended Bloom filter: An aid to network processing. In *Proc. SIGCOMM*, Philadelphia, PA, USA, 2005.

[23] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical en-route filtering of injected false data in sensor networks. In *Proc. INFOCOM'04*, volume 4, pages 2446–2457, 2004.

[24] C. K. Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison Wesley Press, Inc., Cambridge, MA, 1949.