

Adaptive 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 *Adaptive 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 adaptive Bloom filter over that of the traditional Bloom filter is usually of orders of magnitude.

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*, that exists in a much larger universe U . (So $S \subseteq U$.) 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 positive but a low false negative 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 $(1/2)^k$. A Bloom filter uses only a constant number of bits on average in the memory space for each member and answers membership queries with a very small false positive probability. It is a well known compression technique for membership information and has broad applications.

The Bloom filter is a simple, elegant and space efficient structure. 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 due to its simplicity and space efficiency. In particular, the space efficiency of the Bloom filter makes it very appealing in network applications [5], where systems need to share the 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 network [19], packet classification [6], [26], trajectory sampling [12], en-route filtering of false data in the network [27], fast hash table lookup [25] and many others [22], [21]. Motivated by these applications, Bloom filter, since its first invention by Bloom [3], has also been augmented in various ways [20], [8], [9], [14], [18], [19], [23].

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). And 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. 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 [15], [4]. 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 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 is already being monitored [10], [17], [11], [7], [13], [2]. Thus it makes a lot of sense to use such statistics and adapt the Bloom filter with the information about query frequencies and membership likelihood. 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 variable query frequencies and membership likelihood. We call such a Bloom filter the *adapt-*

tive 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. We have derived the optimal configuration of the Bloom filter with respect to the query frequencies and membership likelihood. When the query frequencies and the membership likelihoods are the same for all elements in the universe, the adaptive Bloom filter just converges to the traditional Bloom filter. Thus our model is indeed a generalization and further optimization of the traditional Bloom filter. We also evaluate the performance gain of the adaptive Bloom filter over the traditional one, under various frequency distributions such as step-like functions or Zipf distributions [28], [4], [24]. We observe that the improvement in the false positive probability is of orders of magnitude under reasonable frequency models.

The adaptive Bloom filter can also be gracefully integrated with the frequency estimators [10], [17], [11], [7], [13], [2], which usually maintain a set of ‘hot’ categories. In practice, one can adapt the Bloom filter with such rough frequency estimations, represented by popularity buckets. All the elements in a ‘hot’ category share the same frequency estimation, and they are associated with the category by simply checkable attributes. Examples include 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. That makes the obtaining of the estimations very efficient and the design of the adaptive Bloom filter very practical. We discuss more on the implementation issues in Section III.

II. AN OVERVIEW OF BLOOM FILTER

We first give a quick introduction to 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, $N = |U|$. (Therefore, $S \subseteq U$ and $n \ll N$.) A Bloom filter is an m -bit array and uses k independent uniformly random hash functions $\{h_i | i = 1, 2, \dots, k\}$ which map to range $\{1, \dots, m\}$. For each element in S , the $h_i(x)$ -th bits in the Bloom filter, $i = 1, \dots, k$, 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’. 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

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

Therefore, the probability of a false positive is

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

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 increase 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 will reveal 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. ADAPTIVE 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 *Adaptive 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 [10], [17], [11], [7], [13], [2]. Statistics of such information are maintained, especially for a set of ‘hot’ categories. As will be shown later, such data are very useful for the Bloom filter, even if they are category-oriented and are only rough estimations for the individual elements. We will present the optimal configuration for the adaptive Bloom filter, and show that it generalizes the traditional Bloom filter.

Same as the traditional Bloom filter, an adaptive Bloom filter uses m bits to record the n member elements in a set S . ($|S| = n$.) There are totally N elements in the universe, where $N \gg n$. We assume that the probability of one element’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:

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

Notice that $E\{X_e\} = x_e$.

The basic rationale for designing the adaptive 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 larger weight.

For each element $e \in U$, denote its query frequency by f_e . Denote the number of hash functions used for it by k_e . The total number of hash functions used for elements in S is denoted by $k = \sum_{e \in S} k_e = \sum_{e \in U} X_e \cdot k_e$. (Note that there is a slight abuse of notation here: the notation ‘ k ’ here corresponds to ‘ nk ’ for the traditional Bloom filter.) 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, only false positive.

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:

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

To simplify the analysis, we assume in the standard way [5], [20] 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 because with n being relative large (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 [20]. 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. Thus the probability of false positive, P_{FP} , is:

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

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

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

We use $E\{\cdot\}$ to denote the expectation of a random variable. 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 section is to derive the best configuration of the adaptive Bloom filter. The result is shown in the following theorem.

Theorem 3.1. *In order to minimize the expected false positive probability $E\{P_{FP}\}$ of an adaptive 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*

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

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 adaptive Bloom filter. The detailed derivation is in the Appendix. 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 assign to individual elements, we should also scale those numbers by an appropriate factor in order to control the portion of the Bloom filter that are set to be ‘1’s, because the overall false-positive probability will be hurt whether 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 globally optimum. \square

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

A. Generalization of Bloom filter

In this subsection, we compare the adaptive 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:

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

In the adaptive Bloom filter setting, all the elements should be treated the same way when no knowledge about query frequencies or membership likelihoods is available. Then $\forall e \in U$, we can set f_e and x_e to be constants. 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 $e \in U$. Then 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 varied query frequencies or membership likelihoods, the false-positive probability of

the adaptive Bloom filter usually becomes better than that of the traditional Bloom filter. We use here a simple scenario for a clear illustration. The more general case is its extension. 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:

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

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 $E\{r_e\}$ must be 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 adaptive Bloom filter unless all the elements have the same query frequency.

B. Efficient implementation of adaptive 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, from which we get:

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

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

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

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 the closest non-negative integer.

If k_e falls between two non-negative integers, we can also round it probabilistically to one of them with a simple binary distribution, so that its expected value after rounding is still k_e . That is the approach adopted in the numerical analysis in the following subsection.

C. Numerical Analysis of Adaptive 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], [24] 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 [28], which states that the relative probability of a request for the i th 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 [10], [17], [11], [7], [13], [2]. In this section, we evaluate the performance of adaptive 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 adaptive Bloom filter over the traditional configuration is very impressive.

1) *Different Query Frequency Models, Uniform Membership Likelihood:* We first assume that each element in the universe has the same likelihood of being a member and its query frequency follows some known distributions. The case when the membership likelihood is correlated with its query frequency is studied in the next subsection.

a) *Step Distribution:* We start with a simplified analog of the Zipf's distribution, which we denote by the step distribution. 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.

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

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

The ratio R of the false positive probability of the adaptive 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 Fig. 1 (i).

In Figure 1 (i), 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 adaptive Bloom filter improves its performance when the query frequencies become more skewed. When c is fixed, we see that neither too many nor too few hot elements leads to large values of the P_{FP} improvement. (The ratio of the hot elements is controlled by the parameter α .) That's because in both cases, the query frequencies get closer to a uniform distribution. So the best performance improvement that the adaptive Bloom filter makes appears somewhere in the middle. The difference in the number of hash functions assigned to hot elements and cold elements is moderate — when $m/n = 14$, which is a typical Bloom filter configuration, the number of hash functions assigned to hot elements varies between 10 and 23, and the number of hash functions assigned to cold elements varies between 0 and 10. However, the improvement of the false positive probability is a lot. The maximum P_{FP} improvement here is 396.9.

b) *Zipf Distribution*: The Zipf distribution is defined as follows. The elements in U are sorted according to their query frequencies, from high to low. For the i -th element (where i is called the element's *rank*), its query frequency f is

$$f \propto \frac{1}{i^\alpha}$$

for some constant parameter α .

It is difficult to obtain a closed form expression for P_{FP} improvement because of the divergence of some summations in the related formulas. So we resort to numerical computation based on the known formulas. The result is as shown in Figure 1 (ii). There the horizontal axis is α and the vertical axis is the P_{FP} improvement. The larger α is, the more skewed the query frequencies are, so the P_{FP} performance increases. The P_{FP} improvement corresponds to $\alpha = 1.6$ is 115.7. To reduce the false-positive probability by the factor of 115.7, it will require the traditional Bloom filter to increase its size by $9.89n$ bits, where n is the number of members. That increase is about the size of a normal Bloom filter in many applications. Therefore, such a P_{FP} improvement may be seen as significant.

2) *Considering Membership Likelihoods*: Both positive correlations and negative correlations exist in real life between the membership likelihoods of elements and the query frequencies. Positive correlation means that the higher an element's query frequency is, the more likely it is a member, whose

negative correlation is the other way around. An example of positive correlation is Web caching and distributed file storage, where popular files are more likely to be stored and replicated. Examples of negative correlation appear in network anomaly detection [16] and security check systems, where suspicious events appear much less often than ordinary events. Bloom filter is an especially attractive data structure for such security-related applications for its quicker rejection time and zero false negative rate.

We assume that the correlation between the membership probabilities and the query frequencies can be expressed as a function — namely, $\forall e \in U$, the probability that $e \in S$ is

$$x_e = F(f_e).$$

The function $F(\cdot)$ can be diversified by the applications. In this subsection, we study a basic family of functions:

$$F(f_e) = a f_e^b,$$

where we let b take all possible values. (The value of a depends on b , because $F(\cdot)$ is a probability function, so $\sum_{e \in U} F(f_e) = \sum_{e \in U} x_e = n$, which gives us an equation to solve a using b .) Such polynomial functions are of fundamental interest because they can help us study $F(\cdot)$ of more general forms. To specifically explain it, let's first assume that the function $F(\cdot)$ can be expressed as the summation of a set of other probability functions $g_1(\cdot), g_2(\cdot), \dots, g_t(\cdot)$ — namely, there exist constants $\lambda_1, \lambda_2, \dots, \lambda_t$, such that

$$x_e = F(f_e) = \lambda_1 g_1(f_e) + \lambda_2 g_2(f_e) + \dots + \lambda_t g_t(f_e).$$

Since for any $i \in \{1, 2, \dots, t\}$, $g_i(\cdot)$ is a probability function, we must have $\sum_{e \in U} g_i(f_e) = n$. So we must have

$$\begin{aligned} \sum_{i=1}^t \lambda_i &= \frac{[\sum_{i=1}^t \lambda_i \sum_{e \in U} g_i(f_e)]/n}{[\sum_{e \in U} F(f_e)]/n} \\ &= \frac{[\sum_{e \in U} \sum_{i=1}^t \lambda_i g_i(f_e)]/n}{[\sum_{e \in U} F(f_e)]/n} \\ &= 1 \end{aligned}$$

Now let's look at Equation 7, the expression for the false-positive probability of the adaptive Bloom filter. We can rewrite Equation 7 as

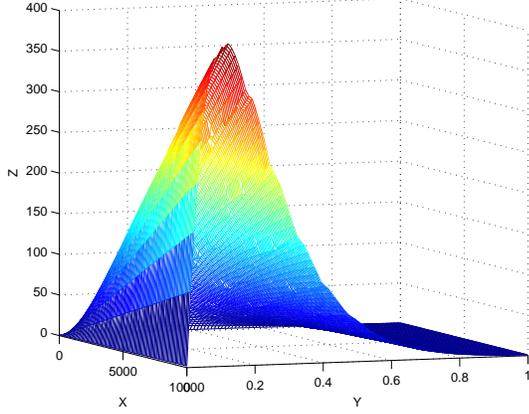
$$\begin{aligned} & E\{P_{FP}\} \\ &= 2^{-(m/n) \ln 2} \cdot N \cdot \prod_{e \in U} E\{r_e\}^{H(f_e)/n} \\ &= 2^{-(m/n) \ln 2} \cdot (\prod_{i=1}^t N^{\lambda_i}) \cdot \prod_{e \in U} E\{r_e\}^{\sum_{i=1}^t \lambda_i g_i(f_e)/n} \\ &= 2^{-(m/n) \ln 2} \cdot \prod_{i=1}^t (N \prod_{e \in U} E\{r_e\}^{g_i(f_e)/n})^{\lambda_i} \end{aligned} \quad (11)$$

We denote by R the ratio of the false-positive probability of the adaptive Bloom filter to that of the traditional Bloom filter. Then since the false-positive probability of the traditional Bloom filter is $(\frac{1}{2})^{-(m/n) \ln 2}$, we get

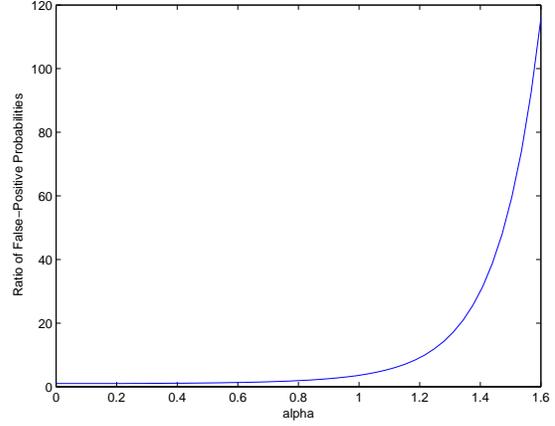
$$R = \prod_{i=1}^t (N \prod_{e \in U} E\{r_e\}^{g_i(f_e)/n})^{\lambda_i} \quad (12)$$

Let's use R_i to denote the value of R when $x_e = g_i(e)$ for all $e \in U$. Clearly, for $1 \leq i \leq t$,

$$R_i = N \prod_{e \in U} E\{r_e\}^{g_i(f_e)/n} \quad (13)$$



(i)



(ii)

Fig. 1. P_{FP} improvement of the adaptive Bloom filter, when the query frequencies are modelled, respectively, with the step distribution and the Zipf distribution. In both cases $m/n = 14$. (i) 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 $[1, 10000]$. (ii) Zipf distribution. The vertical axis is the P_{FP} improvement. The horizontal axis is α .

So

$$R = \prod_{i=1}^t R_i^{\lambda_i} \quad (14)$$

R is the performance improvement when $H(\cdot)$ is used as the membership probability function, while R_i is the performance improvement when $g_i(\cdot)$ is used. Note that $\sum_{i=1}^t \lambda_i = 1$. So R is the weighted geometry average (with λ_i as the weight) of R_1, R_2, \dots, R_t . That means if we can learn R_1, R_2, \dots, R_t , we can learn R as well. Now note that many smooth functions can be expressed as its Taylor series. In such cases, we use Taylor series to express $F(\cdot)$ as

$$F(f_e) = \sum_i \lambda_i (a_i f_e^{b_i}).$$

Here each term $a_i f_e^{b_i}$ corresponds to a function $g_i(f_e)$. Thus if we can learn the performance improvement of the adaptive Bloom filter with $F(f_e) = a f_e^b$, we can learn the performance when $F(\cdot)$ takes more general forms as well. And that is why it is interesting to study the family of functions $F(f_e) = a f_e^b$.

In the following, we let $F(f_e) = a f_e^b$, and study the performance of the adaptive Bloom filter. When $b > 0$, the membership probability x_e has a positive correlation with the query frequency — the greater f_e is, the greater $x_e = F(f_e)$ is. When $b < 0$, they have a negative correlation. We let b take any value and study both cases.

We adopt here the step distribution as the model for the elements' query frequencies for the purpose of simplified illustration, with the notations A, B, α, c and f defined as before. The analysis holds for a broad range of other query frequency models as well, but the calculation there becomes more complex.

Firstly we determine the value of a . We know that

$$\begin{aligned} n &= \sum_{e \in U} x_e \\ &= \sum_{e \in U} F(f_e) \\ &= \sum_{e \in A} a f_e^b + \sum_{e \in B} a f_e^b \\ &= |A| \cdot a \cdot (c f)^b + |B| \cdot a \cdot f^b \\ &= \alpha N a c^b f^b + (1 - \alpha) N a f^b \end{aligned}$$

So

$$a = \frac{n}{\alpha N c^b f^b + (1 - \alpha) N f^b}$$

Then

$$x_e = a f_e^b = \begin{cases} n c^b / (\alpha N c^b + (1 - \alpha) N), & \text{if } e \in A \\ n / (\alpha N c^b + (1 - \alpha) N), & \text{if } e \in B \end{cases}$$

Since N is sufficiently large, the probability that a single element e is a member, x_e , approaches 0. Therefore, from Equation 10, we get

$$\begin{aligned} E\{r_e\} &= f_e / (\sum_{i \in U} f_i) \\ &= f_e / (\sum_{i \in A} f_i + \sum_{i \in B} f_i) \\ &= f_e / (|A| \cdot c \cdot f + |B| \cdot f) \\ &= f_e / (\alpha N c f + (1 - \alpha) N f) \\ &= \begin{cases} c / (\alpha N c + (1 - \alpha) N) & , \text{ if } e \in A \\ 1 / (\alpha N c + (1 - \alpha) N) & , \text{ if } e \in B \end{cases} \end{aligned}$$

We can now compute the ratio of the false-positive probability of the adaptive Bloom filter to that of the traditional

Bloom filter, R :

$$\begin{aligned}
R &= N \cdot \prod_{e \in U} E\{r_e\}^{r_e/n} \\
&= N \cdot \prod_{e \in A} E\{r_e\}^{r_e/n} \cdot \prod_{e \in B} E\{r_e\}^{r_e/n} \\
&= N \cdot \left(\frac{c}{\alpha N c + (1-\alpha)N}\right)^{\frac{nc^b}{\alpha N c^b + (1-\alpha)N}} \cdot \frac{\alpha N}{n} \\
&\quad \cdot \left(\frac{1}{\alpha N c + (1-\alpha)N}\right)^{\frac{n}{\alpha N c^b + (1-\alpha)N}} \cdot \frac{(1-\alpha)N}{n} \\
&= N \cdot \left(\frac{c}{\alpha N c + (1-\alpha)N}\right)^{\frac{\alpha c^b}{\alpha c^b + 1 - \alpha}} \\
&\quad \cdot \left(\frac{1}{\alpha N c + (1-\alpha)N}\right)^{\frac{1-\alpha}{\alpha c^b + 1 - \alpha}} \\
&= \frac{1}{\alpha c + 1 - \alpha} \cdot c^{\frac{\alpha c^b}{\alpha c^b + 1 - \alpha}}
\end{aligned} \tag{15}$$

Figure. 2 shows the P_{FP} improvement. Comparing it to the result in Figure. 1 (i) (where there is no correlation between membership likelihoods and query frequencies), we see that the correlation makes a clear difference. Positive correlation decreases the P_{FP} improvement, while negative correlation increases it, as expected.

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 adaptive 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 adaptive Bloom filter on real network traces and further investigate the integration of the Bloom filter with network statistics estimators.

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] 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.
- [7] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 693–703, London, UK, 2002. Springer-Verlag.
- [8] 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.
- [9] S. Cohen and Y. Matias. Spectral Bloom filters. In *Proc. SIGMOD*, pages 241–252, 2003.

- [10] 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.
- [11] 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.
- [12] N. Duffield and M. Grossglauser. Trajectory sampling with unreliable reporting. In *Proc. INFOCOM'04*, volume 3, pages 1570–1581, Hong Kong, China, 2004.
- [13] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB '98: Proceedings of the 24th International Conference on Very Large Data Bases*, pages 299–310, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [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] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley. Worm detection, early warning and response based on local victim information. In *Proc. of the 20th Annual Computer Security Applications Conference (ACSAC '04)*, December 2004.
- [17] 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.
- [18] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li. Space-code bloom filter for efficient per-flow traffic measurement. In *Proc. INFOCOM'04*, volume 3, pages 1762–1773, Hong Kong, China, 2004.
- [19] 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.
- [20] M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking*, 10(5):604–612, 2002.
- [21] R. Morselli, B. Bhattacharjee, J. Katz, and P. Keleher. Trust-preserving set operations. In *Proceedings of INFOCOM*, volume 4, pages 2231–2241, Hong Kong, China, 2004.
- [22] P. Mutaf and C. Castelluccia. Compact neighbor discovery. In *Proc. INFOCOM'05*, Miami, FL, USA, 2005.
- [23] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proc. INFOCOM'02*, volume 3, pages 1248–1257, 2002.
- [24] 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.
- [25] 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.
- [26] D. E. Taylor and J. S. Turner. Scalable packet classification using distributed crossproducting of field labels. In *Proc. INFOCOM'05*, Miami, FL, USA, 2005.
- [27] 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.
- [28] C. K. Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison Wesley Press, Inc., Cambridge, MA, 1949.

V. APPENDIX

Proof: [Theorem 3.1] In the derivation of the best configuration of the adaptive Bloom filter, we assume that the orders of N and n , as well as the size of the Bloom filter, m , are sufficiently large. Also, we assume that the distributions of the query frequencies of different elements and their probabilities of being members are not overly skewed, so that the term $\sum_{e \in U} X_e \cdot k_e$ is sharply concentrated around its expected value $\sum_{e \in U} E\{X_e\} \cdot k_e$.

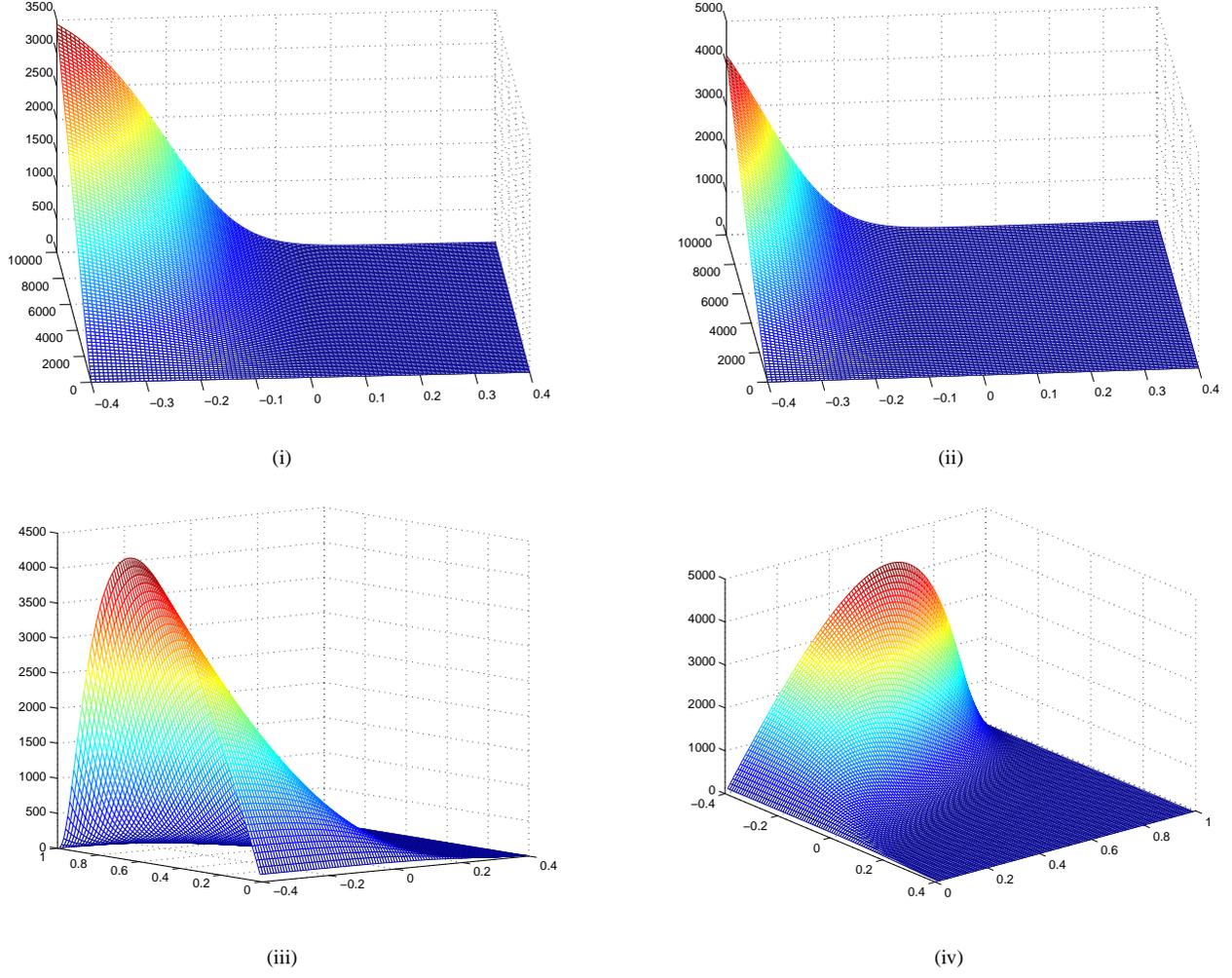


Fig. 2. P_{FP} improvement with correlated membership likelihood and query frequencies. The vertical axis in all four figures is the P_{FP} improvement. (i) $\alpha = 0.4$. The two horizontal axes are, respectively, b that varies in $[-0.4, 0.4]$ and c that varies in $[1, 10000]$. (ii) $\alpha = 0.7$. The two horizontal axes are the same as in (i). (iii) and (iv) the same figure viewed from two different angles. Here $c = 10000$. The two horizontal axes are, respectively, b that varies in $[-0.4, 0.4]$ and α that varies in $[0, 1]$.

Define A as

$$A = \sum_{e \in U} k_e \quad (16)$$

and for $e \in U$, define t_e as

$$t_e = \frac{k_e}{A} = \frac{k_e}{\sum_{i \in U} k_i}. \quad (17)$$

Also, define q as

$$q = (1 - p)^A, \quad (18)$$

then equation 4 becomes:

$$P_{FP} = \sum_{e \in U} r_e (1 - p)^{A \cdot \frac{k_e}{A}} = \sum_{e \in U} r_e q^{t_e}. \quad (19)$$

We have assumed that p almost always equals its expectation. Therefore the expectation of P_{FP} can be written as

$$E\{P_{FP}\} = \sum_{e \in U} E\{r_e\} \cdot q^{t_e}. \quad (20)$$

Below we minimize the expectation of the false positive probability, $E\{P_{FP}\}$.

Let $e_0 \in U$ be an element in U . Then,

$$t_{e_0} = 1 - \sum_{i \in U - \{e_0\}} t_i \quad (21)$$

and

$$E\{P_{FP}\} = \sum_{e \in U - \{e_0\}} E\{r_e\} q^{t_e} + E\{r_{e_0}\} q^{1 - \sum_{i \in U - \{e_0\}} t_i}. \quad (22)$$

Then, for any element $e \in U - \{e_0\}$, we take the partial derivative of equation 22 for t_e :

$$\frac{\partial E\{P_{FP}\}}{\partial t_e} = E\{r_e\} \cdot \ln q \cdot q^{t_e} - E\{r_{e_0}\} \cdot \ln q \cdot q^{1 - \sum_{i \in U - \{e_0\}} t_i}. \quad (23)$$

By setting equation 23 to be 0, we get:

$$E\{r_e\} \cdot q^{t_e} = E\{r_{e_0}\} \cdot q^{1 - \sum_{i \in U - \{e_0\}} t_i}. \quad (24)$$

Multiplying equations 24 for all $e \in U - \{e_0\}$, we get:

$$\prod_{e \in U - \{e_0\}} (E\{r_e\} \cdot q^{t_e}) = \prod_{e \in U - \{e_0\}} (E\{r_{e_0}\} \cdot q^{1 - \sum_{i \in U - \{e_0\}} t_i}),$$

which gives

$$\begin{aligned} & q^{\sum_{e \in U - \{e_0\}} t_e} \prod_{i \in U - \{e_0\}} E\{r_i\} \\ &= (q^{1 - \sum_{i \in U - \{e_0\}} t_i} \cdot E\{r_{e_0}\})^{N-1}. \end{aligned} \quad (25)$$

By reorganizing equation 25, we have

$$\begin{aligned} & q^{(N) \cdot \sum_{e \in U - \{e_0\}} t_e} \prod_{i \in U - \{e_0\}} E\{r_i\} \\ &= (q \cdot E\{r_{e_0}\})^{N-1}, \end{aligned} \quad (26)$$

which gives

$$q^{\sum_{e \in U - \{e_0\}} t_e} = q^{\frac{N-1}{N}} \cdot \prod_{i \in U - \{e_0\}} \left(\frac{E\{r_{e_0}\}}{E\{r_i\}} \right)^{\frac{1}{N}}. \quad (27)$$

By equations 24 and 27, we get that for any $e \in U - \{e_0\}$:

$$\begin{aligned} & E\{r_e\} \cdot q^{t_e} \\ &= E\{r_{e_0}\} \cdot q / q^{\sum_{i \in U - \{e_0\}} t_i} \\ &= E\{r_{e_0}\}^{\frac{1}{N}} \cdot q^{\frac{1}{N}} \cdot \prod_{i \in U - \{e_0\}} E\{r_i\}^{\frac{1}{N}} \\ &= q^{\frac{1}{N}} \cdot \prod_{i \in U} E\{r_i\}^{\frac{1}{N}}. \end{aligned} \quad (28)$$

By using equation 21, we find that equation 28 holds also for $e = e_0$. So for any element $e \in U$, we have:

$$E\{r_e\} \cdot q^{t_e} = q^{\frac{1}{N}} \cdot \prod_{i \in U} E\{r_i\}^{\frac{1}{N}}. \quad (29)$$

Combining equations 17, 18 and 29, we get that for any $e \in U$,

$$E\{r_e\} \cdot (1-p)^{k_e} = (1-p)^{\frac{A}{N}} \cdot \prod_{i \in U} E\{r_i\}^{\frac{1}{N}}. \quad (30)$$

Taking logarithm of both sides of equation 30, we get

$$\ln E\{r_e\} + k_e \cdot \ln(1-p) = \frac{A}{N} \cdot \ln(1-p) + \frac{1}{N} \cdot \sum_{i \in U} \ln E\{r_i\}. \quad (31)$$

Multiplying both sides of equation 31 by X_e , we get

$$\begin{aligned} & X_e \ln E\{r_e\} + X_e k_e \ln(1-p) \\ &= X_e \cdot (A/N) \cdot \ln(1-p) + X_e \cdot (1/N) \cdot \sum_{i \in U} \ln E\{r_i\}. \end{aligned} \quad (32)$$

Taking the expectation of both sides of equation 32, we get

$$\begin{aligned} & x_e \ln E\{r_e\} + x_e k_e \ln(1-p) \\ &= x_e \cdot (A/N) \cdot \ln(1-p) + x_e \cdot (1/N) \cdot \sum_{i \in U} \ln E\{r_i\}. \end{aligned} \quad (33)$$

Summing over $e \in U$ for both sides of equation 33, we get

$$\begin{aligned} & \sum_{e \in U} x_e \ln E\{r_e\} + [\ln(1-p)] \cdot \sum_{e \in U} x_e k_e \\ &= (A/N) \cdot [\ln(1-p)] \cdot n + (1/N) \cdot [\sum_{i \in U} \ln E\{r_i\}] \cdot n. \end{aligned} \quad (34)$$

Remember that we have defined k as

$$k = \sum_{e \in S} k_e = \sum_{e \in U} X_e \cdot k_e, \quad (35)$$

then by equation 1 we can determine the value of k :

$$k \approx -m \ln p. \quad (36)$$

Since p is seen as a constant, so should k by equation 36. Therefore by equation 35, we have:

$$\sum_{e \in U} x_e \cdot k_e = E\{k\} = k \approx -m \ln p. \quad (37)$$

By plugging $\sum_{e \in U} x_e \cdot k_e = k$ into equation 34, we get

$$\begin{aligned} & \sum_{e \in U} x_e \ln E\{r_e\} + B \ln(1-p) \\ &= \frac{A}{N} \cdot [\ln(1-p)] \cdot n + \frac{1}{N} \cdot [\sum_{i \in U} \ln E\{r_i\}] \cdot n. \end{aligned} \quad (38)$$

Raising both sides of equation 38 to the power of e , we get

$$(1-p)^k \prod_{e \in U} E\{r_e\}^{x_e} = (1-p)^{An/N} \cdot \prod_{i \in U} E\{r_i\}^{n/N}, \quad (39)$$

which is equivalent to

$$[(1-p)^k \prod_{e \in U} E\{r_e\}^{x_e}]^{1/n} = (1-p)^{A/N} \cdot \prod_{i \in U} E\{r_i\}^{1/N}. \quad (40)$$

By equations 30 and 40, we get that for any $e \in U$,

$$E\{r_e\} \cdot (1-p)^{k_e} = [(1-p)^k \prod_{i \in U} E\{r_i\}^{x_i}]^{1/n}. \quad (41)$$

By equations 4, 36 and 41, we get

$$\begin{aligned} & E\{P_{FP}\} \\ &= \sum_{e \in U} E\{r_e\} (1-p)^{k_e} \\ &= \sum_{e \in U} [(1-p)^k \prod_{i \in U} E\{r_i\}^{x_i}]^{1/n} \\ &= N \cdot [(1-p)^k \prod_{i \in U} E\{r_i\}^{x_i}]^{1/n} \\ &= N \cdot [(1-p)^{-m \ln p} \prod_{i \in U} E\{r_i\}^{x_i}]^{1/n} \\ &= N \cdot [e^{-m \ln p \ln(1-p)} \prod_{i \in U} E\{r_i\}^{x_i}]^{1/n}. \end{aligned} \quad (42)$$

So $E\{P_{FP}\}$ is minimized when $\ln p \ln(1-p)$ is maximized. From the symmetry of the expression $\ln p \ln(1-p)$, it is simple to find that its value is maximized when

$$p = \frac{1}{2}. \quad (43)$$

Plugging $p = 1/2$ into equation 42, we get

$$\begin{aligned} & E\{P_{FP}\} \\ &= N \cdot [2^{-m \ln 2} \prod_{e \in U} E\{r_e\}^{x_e}]^{1/n} \\ &= 2^{-m \ln 2/n} \cdot N \cdot \prod_{e \in U} E\{r_e\}^{x_e/n}. \end{aligned} \quad (44)$$

By applying equations 36 and 43 to equation 41, we get that for any $e \in U$,

$$E\{r_e\} \cdot 2^{-k_e} = [2^{-m \ln 2} \prod_{i \in U} E\{r_i\}^{x_i}]^{1/n}. \quad (45)$$

Taking logarithm of both sides of equation 45, we get

$$\ln E\{r_e\} - k_e \ln 2 = \frac{1}{n} \cdot [\sum_{i \in U} \ln E\{r_i\}^{x_i} - m \ln^2 2]. \quad (46)$$

So $\forall e \in U$, we have

$$k_e = \frac{m \ln 2}{n} + \frac{1}{\ln 2} (\ln E\{r_e\} - \frac{1}{n} \cdot \sum_{i \in U} x_i \ln E\{r_i\}). \quad (47)$$

Equations 43, 47 and 42 are the same as equations 6, 5 and 7, which is the result we want to derive. We obtained the result by taking the partial derivatives of the expected false-positive probability and setting the partial derivatives to be zero, which gave a unique solution. This unique solution is clearly a global minimum for the expected false-positive probability. \square