Noisy Bloom Filters for Multi-Set Membership Testing

Haipeng Dai Yuankun Zhong Alex X. Liu Wei Wang Meng Li
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, CHINA
{haipengdai,ww}@nju.edu.cn,alexliu@cse.msu.edu,{kun,121220046}@smail.nju.edu.cn

ABSTRACT

This paper is on designing a compact data structure for multi-set membership testing allowing fast set querying. Multi-set membership testing is a fundamental operation for computing systems and networking applications. Most existing schemes for multi-set membership testing are built upon Bloom filter, and fall short in either storage space cost or query speed. To address this issue, in this paper we propose Noisy Bloom Filter (NBF) and Error Corrected Noisy Bloom Filter (NBF-E) for multi-set membership testing. For theoretical analysis, we optimize their classification failure rate and false positive rate, and present criteria for selection between NBF and NBF-E. The key novelty of NBF and NBF-E is to store set ID information in a compact but noisy way that allows fast recording and querying, and use denoising method for querying. Especially, NBF-E incorporates asymmetric error-correcting coding technique into NBF to enhance the resilience of query results to noise by revealing and leveraging the asymmetric error nature of query results. To evaluate NBF and NBF-E in comparison with prior art, we conducted experiments using real-world network traces. The results show that NBF and NBF-E significantly advance the state-of-the-art on multi-set membership testing.

Keywords

Bloom filter, multi-set membership testing, noise, asymmetric error-correcting code, constant weight code

1. INTRODUCTION

1.1 Problem Statement and Motivation

This paper is on designing a compact data structure for multi-set membership testing allowing fast set querying. Given a set of sets $S = \{S_1, S_2, \ldots, S_\aleph\}$ where $S_i \cap S_j = \varnothing$ for any $1 \leq i < j \leq \aleph$, a multi-set membership testing algorithm builds an efficient data structure so that given an element e, the algorithm either finds $S_i \in S$ such that $e \in S_i$, or report that $e \notin S_1 \cup S_2 \ldots \cup S_\aleph$.

SIGMETRICS '16, June 14-18, 2016, Antibes Juan-Les-Pins, France

DOI: http://dx.doi.org/10.1145/2896377.2901451

Multi-set membership testing is a fundamental operation for computing systems and networking applications. For example, for frame forwarding in a layer-2 switch, multiple MAC addresses are mapped to a single port and fast querying for the associated port for a MAC address is crucial for frame forwarding [11, 25]. Here, MAC addresses correspond to elements and the associated port corresponds to the set. For Web traffic classification, URLs are classified into different groups on the fly for statistical purposes [11]. For approximate state machines, the simultaneous state of a large number of agents are tracked by a state machine [4]. For Web cache, each Web proxy maintains a compact summary of the cache directory of all other proxies and if a cache miss occurs it searches for the proxy wherein the request is a cache hit [8].

1.2 Limitations of Prior Art

Many existing solutions addressing multi-set membership testing are built upon Bloom filter [5, 6, 8, 9, 11, 13, 15, 23, 24]. Bloom filter is a compact data structure supporting membership query with no false negatives but with false positives, *i.e.*, it certainly answers yes if the query element belongs to the set and may mistakenly answer yes if not. Given a set of element S, Bloom filter first constructs a bit array with all bits initialized to 0. For inserting an element e to S, Bloom filter uses k independent hash functions with uniformly distributed outputs, say $h_1(\cdot), \ldots, h_k(\cdot)$, to map e to k bits in the array and set them to 1. For querying whether an element e belongs to S, Bloom filter returns true if all k mapped bits for e are 1.

Most of the Bloom filter based schemes regarding multi-set membership testing fall short in either storage space cost or query speed. On one hand, some prior art adopts a general framework that divides the whole storage space into multiple uniform cells and uses one or more cells to record set IDs of element [6, 9, 23]. This coarse granularity for storing set IDs leads to inefficient use of storage space. For the IBF scheme proposed in [6, 9], if two or more distinct set IDs are accidentally mapped to the same cell and mixed with each other, then no useful information can be extracted from this cell, which means the whole cell is wasted. kBF proposed in [23] improves IBF by allowing mixture from no more than 3 set IDs using a coding technique. However, kBF is still inefficient because it fails to decode any set IDs of elements mapped to the cell where more than 3 set IDs are mixed. On the other hand, other prior art mainly falls short in memory access overhead and query processing speed. Most of proposed schemes in these works assign multiple dedicated Bloom filters for recording in terms of set ID [5, 8], bit posi-

tion in the binary expression of set IDs [15], and bit position in the encoding results of set IDs [11], or organizes multiple Bloom filters into a tree [24]. Consequently, the memory accesses of these schemes are generally several times higher than standard Bloom filters depending on the number of established Bloom filters, which results in much lower query processing speed. Still another scheme encodes set ID information of an element in its location by using offsets [13]. The offset is set to be proportional to the value of set ID. As both classification failure rate and memory access overhead grow proportionally to the maximum set ID value, this scheme is vulnerable to noise and has low query processing speed. One solution to alleviate this problem is to require that set IDs be no more than 64 [13], which largely limits its applications in reality.

1.3 Proposed Approach

In this paper, we propose a Noisy Bloom Filter (NBF) scheme as well as its enhanced version Error Corrected Noisy Bloom Filter (NBF-E) scheme for multi-set membership testing. Our NBF and NBF-E schemes simultaneously improve the storage space cost and query speed issues over prior art. NBF and NBF-E have two phases: construction phase and query phase. NBF encodes set IDs of input elements and records the results in a bit array in a noisy way to achieve compact storage in the construction phase, and denoises the recorded coding information to recover the set IDs in the query phase. Compared with NBF, NBF-E enhances the resilience to noise by using asymmetric error-correcting codes.

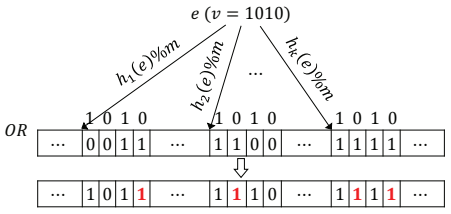

Figure 1: Construction phase of NBF/NBF-E

In the construction phase, given an element, NBF first maps it to k positions in its bit array using k hash functions. Then, NBF encodes the set ID of the element into a bit string with fixed length, and ORs the encoding result with the bitmaps starting from each of k hashed positions and lasting the same length of the encoding result. Figure 1 shows an example of the construction phase. An input element e is mapped to k positions in NBF using hash functions $h_1(e), h_2(e), \ldots h_k(e)$, its set ID is encoded as $v = 1010$, and is ORed with 4 consecutive bits starting from each of these k positions. The numbers marked in bold font and red color in the resulting NBF denote "noise" from the recorded information of other elements with respect to e.

In the query phase, NBF first computes the mapped positions for an input element e using the hash functions $h_1(e), h_2(e), \ldots h_k(e)$, and then fetches k related bitmaps starting from these k positions. To remove the potential noise from other elements, NBF performs AND operation across all bitmaps and yields the encoding result, which can be decoded to a set ID if it is valid. Figure 2 illustrates the querying procedure for the example stated in the construction phase. It can be seen that all noise is removed after the

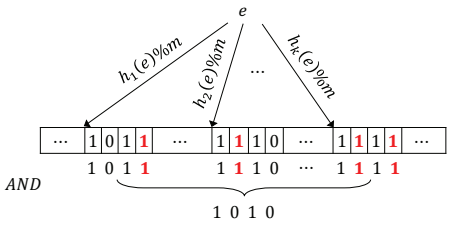

Figure 2: Query phase of NBF/NBF-E

AND operation and NBF finally obtains the correct encoding result 1010, and then decodes it to the original set ID. Besides, it is clear that when the number of total distinct sets equals 1, our NBF scheme is reduced to the standard Bloom filter as it encodes the set ID for all elements as 1 and writes it to or reads it from all k hashed bits.

Our proposed enhanced version NBF-E is built upon NBF, and it differs from NBF only in the encoding and decoding process. Particularly, NBF-E applies (binary) asymmetric error-correcting coding, or more specifically, (binary) Z channel coding, for encoding and decoding set IDs for elements. Asymmetric error-correcting coding is a type of forward error correction schemes which encode data by adding additional information such that introduced errors can be effectively detected and corrected, and it is particularly suitable for asymmetrical channels, or Z channel in our case. Z channel has $\{0, 1\}$ as input and output alphabets where the crossover $0 \rightarrow 1$ occurs with positive probability p whereas the crossover $1 \rightarrow 0$ never occurs. This precisely describes the error property of query results in our scenario, that is, only "0" bits prone to error but "1" bits do not. In our previous example as shown in Figure 2, noise only occurs to the "0" bits in the encoding result 1010 recorded in the Bloom filter and probably in the query result as we can image, but cannot occur to the "1" bits. By revealing this critical fact and using asymmetric error-correcting coding, we show theoretically that NBF-E achieves almost half of the false positive rate and twice faster decoding speed compared with schemes using traditional symmetric error-correcting coding.

Key intuition: The intuition behind storing element codes in bits, instead of in cells each of which consists of a fixed number of bits, is to save memory. Due to the abandon of delimits for storing data, encoding results for different element set IDs will overlap with each other. The intuition behind allowing this overlap is that NBF/NBF-E can effectively extract the original encoding results using AND operation to denoise mixed-in information from other elements in the query phase. The motivation behind storing encoding results in consecutive bits rather than in disjoint bits using multiple hash functions as [11, 15] do is to reduce memory access overhead. This guarantees each encoding result can be stored or fetched in one memory access. The motivation behind using asymmetric error-correcting coding technique in NBF-E is to enhance the resilience of recorded encoding results to noise by fully exploiting the asymmetric error nature of query results.

1.4 Key Technical Challenges

The first technical challenge is to minimize the classification failure and false positive for NBF scheme under a given memory constraint. To address this challenge, we first infer

the expression of error rate for "0" bits in a set ID code, which is uniform and can be reasonably approximated as independent and identically distributed. Based on this, we further show that the number of error bits in the query results can be modeled with Binomial distributions. We then derive the expressions of classification failure rate and false positive rate. Furthermore, we optimize these two metrics in terms of number of hash functions and code size.

The second technical challenge is to best tradeoff the benefits brought by incorporating asymmetric error-correcting codes in NBF-E scheme and the aggravated noise arose from such codes. We first derive the expressions of classification failure rate and false positive rate for our NBF-E scheme. Among these two metrics, classification failure rate is much more complicated than its counterpart of NBF due to the asymmetric error-correcting ability of NBF-E, which hinders further optimization analysis. To address this challenge, we propose to analyze its upper bound, which is shown to be a relatively accurate estimate and is much easier to be analyzed than its exact value. Then, we obtain optimization results of classification failure rate and false positive rate in terms of number of hash functions and code size.

The third technical challenge is to provide guidance for choosing between NBF and NBF-E schemes. As NBF-E does not always outperform NBF due to its introduced noise, we need to answer the question of under which condition which scheme is more preferable than the other in terms of critical metrics, such as classification failure. This is a challenging task because of the fundamental difference, though seemingly small, between these two schemes. We address this challenge by employing appropriate relaxation techniques and using the lower bound, rather than exact value, of the number of asymmetric error-correcting codewords to make the comparison of classification failure rates for these two schemes feasible.

1.5 Advantages over Prior Art

Previous literature falls short in either storage space cost or query speed. To evaluate our NBF and NBF-E schemes in comparison with prior art, we conduct trace-driven experiments. Our results show that NBF/NBF-E significantly advances the state-of-the-art on multi-set membership testing. Suppose all schemes are under the same storage space constraint. In comparison with COMB, NBF-E has 13% higher correctness rate, 3.7 times smaller memory accesses and 3.3 times faster query processing speed. In comparison with Summary Cache, NBF/NBF-E has comparable correctness rate, 7.7 times lower memory accesses and about 6 times faster query processing speed. In comparison with kBF, NBF/NBF-E has 8.7 times higher correctness rate and about 6 times faster query processing speed. In comparison with IBF, NBF-E has hundreds of times higher correctness rate.

2. RELATED WORK

In this section, we briefly review related works regarding Bloom filters for multi-set membership testing.

2.1 Cell Based Approach

David *et al.* proposed Invertible Bloom Filter (IBF) which can be inverted to yield some or all of the inserted key IDs of elements [6,9]. An IBF consists of an array of cells each of which contains a counter, the XOR of all key IDs that hash into that cell, as well as the XOR of the hashes of all IDs that hash into the cell. In the query phase, the cells that record only a single element are first identified and recovered. Then, the set ID information of these elements is subtracted from IBF using XOR functions. Such identification and removal process will repeat until no further elements can be recognized. The major shortage of IBF is its memory inefficiency as it uses cells instead of bits. Xiong *et al.* proposed key-value Bloom Filter (kBF) that also stores values of elements in cells [23]. Each cell contains a counter and a possibly superimposed encoding result, and kBF can only infer original encodings from superimposed encoding results from at most three elements, which constrains its success rate of querying. Both of IBF and kBF are less memory efficient compared to our NBF scheme because NBF stores encoding results of set IDs in bits and allows the bit-level overlap.

2.2 Multiple Bloom Filter Based Approach

Fan *et al.* proposed a scalable web cache sharing scheme called Summary Cache, which in essence is a straightforward Bloom filter solution that allows multi-set membership testing [8]. It generates one Bloom filter for each set in the construction phase, and performs $k \times \aleph$ hashes for recording set IDs or querying which set the input element belongs to, where k is the number of hashes for each Bloom filter and $\aleph$ is the number of distinct sets. Chazelle *et al.* proposed Bloomier filter that consists of multiple suites of Bloom filters, or equivalently, a series of Bloom filters in Summary Cache [5]. Each suite for Bloomier filter contains $\aleph$ Bloom filters for $\aleph$ sets, and applies hash functions different from other suites to store and query set IDs. Especially, during the query phase, Bloomier filter searches all suites of Bloom filters one by one until it reaches a suite returning only one positive answer among $\aleph$ Bloom filters. The goal of Bloomier filter is to alleviate the false positive of the presented Bloom filter in Summary Cache. Lu *et al.* reported an improved solution based on Summary Cache which assumes each input set ID is L-bit long and generates one Bloom filter for each bit in set ID [15]. When performing Bloom filter construction or multi-set membership query, it uses $k \times L$ hash functions to record or determine the corresponding bit value in L-bit long set ID. As normally we have $L = O(\log \aleph)$, this solution is supposed to be much faster than Summary Cache scheme in [8]. Hao *et al.* proposed COMbinatorial Bloom filter (COMB) that encodes each set ID into an L-bit binary vector with θ $1s$ and $L-\theta$ $0s$ and then uses k hash functions for each of L bits in a single Bloom filter [11]. COMB needs to compute $k \times \theta$ hash functions for construction and query for each element. In comparison, our NBF scheme only needs to compute k hash functions. Yoon *et al.* proposed Bloom tree [24], which maintains a binary search tree with each node being a Bloom filter and the leaf nodes representing distinct values. To query a set ID for a given element is to determine a unique path from the root to a leaf node, and its query speed is t times that of standard Bloom filter where t is the depth of the Bloom tree. In summary, compared with the above schemes, NBF needs only k modulo operations and k memory accesses and is thus more time efficient. For example, NBF is 6 times faster than Summary Cache and 3.3 times faster than COMB in terms of query processing speed as demonstrated by our experimental results.

2.3 Offset Based Approach

Lee *et al.* proposed a new data structure called SVBF that encodes the set information in an offset to save s-

Table 1: Notations

Symbol	Meaning
m	Size of a Bloom filter in bits
B	Bit array of a Bloom filter
n	Number of stored elements
e	An element of a set
v	Set ID of element e
$\aleph$	Number of distinct set IDs of elements
$h_i(\cdot)$	The i-th hash function
k	Number of hash functions for a Bloom filter
f	Size of set ID code in bits
w	Constant Hamming weight of codes
d	Minimum Hamming distance of codes
t	Number of correcting error bits
p_e	Error rate for "0" bits of set ID code in query result
P_{cf}	Classification failure rate
P_{fp}	False positive rate

pace [13]. In particular, unlike standard Bloom filter that sets k hash values $h_i(e)\%m$ $(i = 1, \ldots k)$ to 1 where % represents modular operation, SVBF sets the bit corresponding to $(h_i(e) + j)\%m$ $(i = 1, \ldots k)$ to 1 where j $(0 < j \leq g)$ is the set ID, g is the maximum set ID, and m is the size of SVBF. In the query phase, SVBF first reads the next g consecutive bits from each base $h_i(e)\%m$ to get a bitmap B_i. Then, SVBF computes the bit-wise AND across all bitmaps to get the final bitmap B, and outputs the minimum offset where a bit is set to 1 as the set ID. The main limitation of SVBF is its bad scalability. On one hand, the classification failure of SVBF would increase rapidly as g increases because the probability that noisy bits from other elements appear between $h_i(e)\%m$ and $(h_i(e)+j)\%m$ increases nearly proportionally with j. On the other hand, as for each hash position in the query phase, SVBF needs to fetch all next g bits starting from this position, which may lead to memory accesses proportional to g when g becomes large. For these reasons, g is required to be no more than 64 to control classification failure and memory accesses in [13], which, however, constrains its applications. Compared with SVBF, NBF can scale to a much larger number of distinct set IDs (up to 1.83×10^{18}).

3. NOISY BLOOM FILTER (NBF)

In this section, we first describe the construction phase and query phase of our proposed Noisy Bloom Filter (NBF) scheme. Then, we give detailed theoretical analysis about the performance of NBF in terms of classification failure rate and false positive rate. In addition, we discuss parameter optimization to minimize classification failure rate as well as false positive rate. Table 1 summarizes notations used in this paper.

3.1 Construction Phase

In the construction phase, NBF first constructs a bit array B of size m with all bits initialized to 0. Suppose NBF needs to store n distinct elements $e_1, e_2, \ldots, e_n$, each of which with an associated set ID v_i. Then, the recording process for an element e is to first map e into k different positions $h_1(e)\%m$, $h_2(e)\%m$, ..., $h_k(e)\%m$ using k distinct hash functions $h_1(.), h_2(.), \ldots, h_k(.)$, where the symbol % represents modular operation.

Next, instead of recording the original set ID of e, NBF stores its corresponding constant weight code. Here constant weight code refers to a type of codes with constant Hamming weight. That is, the Hamming weight, namely the number of 1s in a code, for any codeword is a predefined constant w. We use $C(v)$ to denote the corresponding constant weight code for a set ID v. In practice, $C(v)$ can be computed by a well defined function, or obtained by querying a codebook that records all the mappings of set IDs and their corresponding codes. Furthermore, the number of codewords for constant weight code, *i.e.* $\binom{f}{w}$, should be sufficiently large to express all possible values for v, which is presumed to be bounded by $\aleph$. That is, NBF should carefully choose f and w such that $\binom{f}{w} \geq \aleph$. We will list the motivations for adopting constant weight code later in this subsection.

After that, NBF records the set ID of element e by ORing its constant weight code with the next f consecutive bits starting from each of the k hashed position. Specifically, NBF sets $B[(h_i(e)\%m + j)\%m] = B[(h_i(e)\%m + j)\%m] \mid C(v)[j]$ $(1 \leq i \leq k;\ 0 \leq j < f)$. As the length of the constant code f is typically set to be small to guarantee one memory access (≤ 64, which still allows expressing at most $\binom{64}{32} \approx 1.83 \times 10^{18}$ set IDs), all f relevant bits starting from each hashed position can be fetched or updated using one memory access operation. Note that modern architectures like X86 platform CPU can access data starting at any byte, *i.e.*, access data aligned on any boundary, not just on word boundaries [1, 17]. In addition, if $h_i(e)\%m + j > m$, NBF needs to wrap around and update an appropriate number of starting bits of B to make the total number of bits be f. This undoubtedly leads to one extra memory access.

The motivations for storing constant weight codes of input element set IDs rather than set IDs themselves or other non-constant weight codes are four-folds. First, constant weight codes are generally more space-efficient than directly storing set IDs. The cardinality of distinct values of input elements $\aleph$ could be far less than the largest set ID of input elements. As the length of codes should be designed to be uniform among elements and should accommodate the largest set ID of input elements, directly storing set IDs of elements could be a waste of space. For example, expressing elements with three distinct set IDs 1, 2 and $2^{32} - 1$ needs 32 bits, while constant weight codes 001, 010, 100 with three-bit length are sufficient to present all set IDs. Second, constant weight codes simply check the Hamming weight of the query result to verify its correctness, which is more efficient and explicit than other methods. Third, compared with other methods, constant weight codes can eliminate or alleviate the impact of distribution skewness of input element set IDs, guaranteeing the performance of NBF in general cases. The distribution of element set IDs is typically skewed and unpredictable, *e.g.*, traffic flows across large networks mostly follow Zipf or "Zipf-like" distribution which is highly skewed [19]. Using non-constant weight codes could introduce heavy noise to data stored in NBF if the codes of set IDs corresponding to large numbers of elements happen to have large Hamming weights, and therefore, it leads to high classification failure rate or high false positive rate. Fourth, constant weight codes can ensure fairness for set IDs in terms of query success rate, whereas other methods do not. As noise in NBF is approximately uniformly distributed, non-constant weight codes or set IDs with relatively less Hamming weight would suffer more from noise as they have more "0" bits (we will prove "0" bits are affected by noise while "1" bits are not), and are less likely to be successfully recovered.

3.2 Query Phase

To query an element e, NBF finds its associated set ID v by the following four steps. First, it computes k hash functions $h_1(e)\%m$, ..., $h_k(e)\%m$ to determine k positions from where the data records of e start. Second, for each $1 \le i \le k$, it reads the next consecutive f bits $B[(h_i(e)\%m+j)\%m]$ $(0 \le j < f)$ and forms a bitmap B_i. Third, we conduct bitwise AND operation among all obtained k bitmaps, *i.e.*, $C(v) = B_1\& \ldots \&B_i\& \ldots \&B_k$ $(1 \le i \le k)$. Fourth, if the Hamming weight of the obtained code $C(v)$ is equal to w, then we report the corresponding v of $C(v)$, *i.e.* $C^{-1}(C(v))$, as the associated set ID of e. In the case that the Hamming weight is smaller than w, NBF reports that the element v doesn't exist in the Bloom filter and discards the query result.

There are two types of failures that can happen in the query phase of NBF: *classification failure* and *false positive*. In the case that the Hamming weight of $C(v)$ is larger than w, NBF cannot classify the query result into any predefined constant weight code, which we call it as *classification failure*. In the case that the element is not in the Bloom filter, but the Hamming weight of $C(v)$ is equal to w due to noises, NBF will falsely identify the element to a set with set ID v, which we call it as *false positive*.

3.3 Analysis

For theoretical analysis, we pay special attention to classification failure rate and false positive rate. In particular, we derive expected values of these parameters and discuss how to optimize them.

As the basis of further analysis, we begin with inferring the rate of a bit error in the query result, which is quantitatively evaluated by the equation in the following lemma.

LEMMA 1. *Given the NBF size m, number of stored elements n, code length f, code weight w and number of hash functions k, the error rate for "0" bits in a set ID code,* i.e. *the probability for any "0" bit in a set ID code changing to* 1 *in the query result, denoted by p_e is given by*

$$p_e = \left(1 - \left(1 - \frac{w}{m}\right)^{nk}\right)^k, \tag{1}$$

and the error rate for "1" bits is zero.

PROOF. Considering a "0" bit b in a recorded set ID code in B, the probability that another recorded set ID code with length f covers this bit is f/m, and the probability that the overlapped bit in this recorded set ID code taking value 1 is w/f, thus the probability that a "1" bit from another recorded set ID code overlaps with b is $f/m * w/f = w/m$. As there are $n \times k$ recorded set ID codes, the probability that b does NOT change its value is the probability that every overlapped bit takes value 0, *i.e.*, $(1 - w/m)^{nk}$. In contrast, the probability that b changes its value is $1 - (1 - \frac{w}{m})^{nk}$. Finally, a "0" bit in a set ID code changing to 1 in the query result after AND operation is conditioned on all k bits in k corresponding recorded set ID codes are "1", whose probability is $p_e = (1-(1-\frac{w}{m})^{nk})^k$. Next, it is easy to see the error rate for "1" bits is zero. This completes the proof. □

We continue to discuss the correlation between error rates of bits in a query result. For two bits b_p^i and b_q^i in a fetched bitmap B_i, their error rates are indeed statistically correlated because the "1" bits overlapped with b_p^i and b_q^i may come from the same set ID code and they are dependent of each other due to the constant weight constraint of set ID code. To see this, imagine a simple case that both b_p^i and b_q^i are covered by a set ID code $C(v)$ with code length f and Hamming weight w, then given that b_p^i is overlapped with a "1" bit in this code, the probability that b_q^i is also overlapped with a "1" bit in $C(v)$ is reduced from w/f to $(w-1)/(f-1)$, which means their error rates are correlated. However, for the corresponding two bits b_p^j and b_q^j at the same positions in another bitmap B_j that corresponds to the same element with B_i, their error rates should be almost independent of that of b_p^i and b_q^i. This is because any set ID code that covers b_p^j and b_q^j is unlikely to cover b_p^i and b_q^i due to the uniform distribution property of the outputs of the applied hash functions. Suppose the resulting bits in the query result is $b_p = b_p^1\&b_p^2 \ldots \&b_p^k$ and $b_q = b_q^1\&b_q^2 \ldots \&b_q^k$. As b_p equals 1 if and only if $b_p^i = 1$ for all $1 \le i \le k$ and the case is similar to b_q, it is clear that the correlation between error rates of any pair of bits b_p^i and b_q^i will be substantially weakened, which results in a weak correlation between b_p and b_q. For this reason, we approximately assume the error rates of bits conform to *independent and identical distribution (i.i.d.)*. We will justify it by experimental results to several theorems below based on this assumption.

Next, we analyze the classification failure rate for query phase. Note that at the last step of query phase, classification failure occurs if the Hamming weight of the query result is larger than w, or equivalently, there is one or more "0" bits are corrupted. The theorem below formally gives the expression of classification failure rate.

THEOREM 1. *Given the NBF size m, number of stored elements n, code length f, code weight w and number of hash functions k, the classification failure rate P_{cf} for NBF is given by*

$$1 - (1 - p_e)^{f-w}, \tag{2}$$

where $p_e = (1 - (1 - \frac{w}{m})^{nk})^k$.

PROOF. Suppose there are exactly j $(1 \le j \le f - w)$ "0" bits are corrupted while the other $f - w - j$ "0" bits are not. Apparently, j conforms to Binomial distribution, *i.e.*, $j \sim Binom(f - w, p_e)$ where $p_e = (1 - (1 - \frac{w}{m})^{nk})^k$ by following Lemma 1 and our *i.i.d.* assumption for error rates of bits in the query result. Thus the classification failure rate for j corrupted bits is given by $\binom{f-w}{j} p_e^j (1 - p_e)^{f-w-j}$. By considering all possible cases of j $(1 \le j \le f - w)$, we have $P_{cf} = \sum_{j=1}^{f-w} \binom{f-w}{j} p_e^j (1 - p_e)^{f-w-j}$, which is equivalent to Equation (2). This completes the proof. □

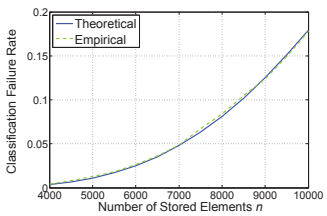

Figure 3: Comparison of Theoretical and Empirical Value of P_{cf}

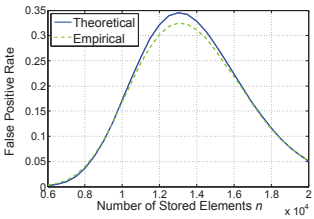

Figure 4: Comparison of Theoretical and Empirical Value of P_{fp}

As Figure 3 shows, the theoretical value of classification failure rate fits well with the empirical value when $m = 1.2 \times 10^5$, $f = 5$, $w = 2$, $k = 6$, and n increases from 4000 to 10000. The average approximation error is merely 5.3%.

In addition to classification failure rate, false positive rate is also a critical performance metric which refers to the probability with which an NBF mistakenly returns a positive answer for an absent element. The theorem below gives the formal expression of false positive rate.

THEOREM 2. *Given the NBF size m, number of stored elements n, code length f, code weight w and number of hash functions k, the false positive rate P_{fp} for NBF is given by*

$$\binom{f}{w} p_e^w (1-p_e)^{f-w}, \tag{3}$$

where $p_e = (1-(1-\frac{w}{m})^{nk})^k$.

PROOF. A false positive happens if and only if there are exactly w bits corrupted which make the obtained result undistinguishable from valid results. By Theorem 1, each bit in the final result has probability p_e to take value 1 and probability $1-p_e$ to take the opposite value. Therefore, the false positive rate is given by Equation (3). □

Figure 4 shows that the empirical value of false positive rate agrees well with the theoretical value when $m = 1.2 \times 10^5$, $f = 5$, $w = 2$, $k = 12$, and n grows from 6000 to 20000, and the average approximation error is only about 6.1%.

In what follows, we are dedicated to optimizing classification failure and false positive in terms of number of hash functions k.

THEOREM 3. *Given the fixed NBF size m, number of stored elements n, code length f, and code weight w, and the error rate for "0" bits in a set ID code p_e is sufficiently small, both of the classification failure rate P_{cf} and false positive rate P_{fp} for NBF can be minimized when*

$$k = \frac{m}{wn} \ln 2. \tag{4}$$

PROOF. For classification failure rate, by Equation (2), it is obvious that P_{cf} is minimized when p_e is minimized.

For false positive rate, taking the first-order partial derivative of $\ln P_{fp}$ with respect to p_e and following Equation (3) we have

$$\begin{aligned} \frac{\partial}{\partial p_e} \ln P_{fp} &= \frac{\partial}{\partial p_e} \ln \left[\binom{f}{w} p_e^w (1-p_e)^{f-w} \right] \\ &= \frac{w}{p_e} - \frac{f-w}{1-p_e}. \end{aligned} \tag{5}$$

By letting the above formula equal to zero, we obtain $p_e = \frac{w}{n}$. Furthermore, taking the second-order partial derivative of $\ln P_{fp}$ with respect to p_e and plugging in $p_e = \frac{w}{n}$ we have

$$\begin{aligned} \frac{\partial^2}{\partial^2 p_e} \ln P_{fp} &= -\frac{w}{p_e^2} - \frac{f-w}{(1-p_e)^2} \\ &= -\frac{w}{(w/n)^2} - \frac{f-w}{(1-w/n)^2} \\ &< 0. \end{aligned} \tag{6}$$

Thus, P_{fp} is maximized at point $p_e = \frac{w}{n}$. In contrast, in order to minimize P_{fp}, we need to minimize p_e or maximize p_e to make it being deviate from $\frac{w}{n}$ as much as possible. When p_e is sufficiently small such that $p_e < \frac{w}{n}$, minimizing p_e should be a reasonable choice to minimize P_{fp}.

Next, we study how to minimize p_e. In particular, we take the first-order partial derivative of $\ln p_e$ with respect to k and make an appropriate approximation

$$\begin{aligned} \frac{\partial}{\partial k} \ln p_e &= \frac{\partial}{\partial k} \ln (1-(1-\frac{w}{m})^{nk})^k \\ &\approx \frac{\partial}{\partial k} \ln (1-e^{-\frac{wnk}{m}})^k \\ &= \ln (1-e^{-\frac{wnk}{m}}) + \frac{wnk}{m} \frac{e^{-\frac{wnk}{m}}}{1-e^{-\frac{wnk}{m}}}. \end{aligned} \tag{7}$$

It is easy to check the above derivative is equal to zero when $k = \frac{m}{wn} \ln 2$, and further this is a global minimum. To summarize, both of P_{cf} and P_{fp} are minimized when p_e is minimized at point $k = \frac{m}{wn} \ln 2$. This completes the proof. □

Discussion: We note that there is an interesting similarity between the optimal number of hash function for our proposed NBF and that for standard Bloom filter, which is given by $k' = \frac{m}{n} \ln 2$. As standard Bloom filter writes a single 1 to each of k hash bits whereas NBF writes w 1s, the total numbers of written 1s for both filters are the same, *i.e.*, $wnk = nk' = m \ln 2$. Therefore, on average each bit in both filters has the same probability 1/2 to be 1 or 0.

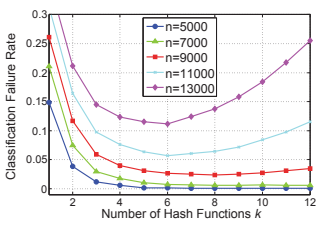

Figure 5: P_{cf} vs. k for NBF

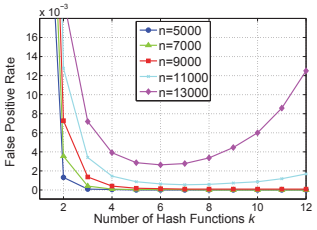

Figure 6: P_{fp} vs. k for NBF

Figures 5 and 6 show the classification failure rate P_{cf} and false positive rate P_{fp} when $m = 3 \times 10^5$, $f = 8$, $w = 3$, k increases from 1 to 12 and n increases from 5000 to 13000. Basically, all curves for P_{cf} and P_{fp} first drops until reaching a minimum, and then rises. For example, P_{cf} achieves the minimum points at $k = 14, 9, 8, 6, 6$, while P_{fp} attain its minimum at $k = 14, 10, 8, 7, 6$, when $n = 5000, 7000, 9000, 11000, 13000$ respectively. In comparison, the theoretical optimal k for $n = 5000, 7000, 9000, 11000, 13000$ are $k = 13.86, 9.90, 7.70, 6.30, 5.33$, respectively, which match well with the actual values. Note that the theoretical k is typically not an integer, while in practice k must be an integer. This corroborates with Theorem 3.

Next, we continue to optimize P_{cf} and P_{fp} in terms of f, and present theoretical findings in the theorem below.

THEOREM 4. *Both of the classification failure rate P_{cf} and the false positive rate P_{fp} for NBF decrease monotonically when the code size f decreases given that the number of hash functions k takes its optimal value,* i.e., *$k = \frac{m}{wn} \ln 2$, and the error rate for "0" bits in a set ID code p_e is sufficiently small.*

PROOF. For classification failure, plugging the expression of k in Equation (4) into Equation (2) we obtain $P_{cf} =$

$1-(1-(\frac{1}{2})^{\frac{m}{wn}\ln 2})^{f-w}$. As m, n and w are fixed, it is clear that P_{cf} decreases with an decreasing f.

For false positive, we plug the optimal value of k into Equation (3) and have

$$P_{fp} = \binom{f}{w}\left(\frac{1}{2}\right)^{\frac{m}{n}\ln 2}\left[1-\left(\frac{1}{2}\right)^{\frac{m}{wn}\ln 2}\right]^{f-w} \quad (8)$$

To search for the optimal value of f that minimizes P_{fp}, we take the first-order partial derivative of $\ln P_{fp}$ with respect to f

$$\begin{aligned}\frac{\partial}{\partial f}\ln P_{fp} &= \frac{\partial}{\partial f}\ln\left\{\binom{f}{w}\left(\frac{1}{2}\right)^{\frac{m}{n}\ln 2}\left[1-\left(\frac{1}{2}\right)^{\frac{m}{wn}\ln 2}\right]^{f-w}\right\}\\ &= \frac{\partial}{\partial f}\ln\frac{\prod_{i=0}^{w-1}(f-i)}{w!} - \ln\left[1-\left(\frac{1}{2}\right)^{\frac{m}{wn}\ln 2}\right]\\ &= \sum_{i=0}^{w-1}\frac{1}{f-i} - \ln\left[1-\left(\frac{1}{2}\right)^{\frac{m}{wn}\ln 2}\right] \quad (9)\end{aligned}$$

As $\frac{\partial^2}{\partial^2 f}\ln P_{fp} = -\sum_{i=0}^{w-1}\frac{1}{(f-i)^2} < 0$, f that makes the last formula equal to zero is a maximum. Furthermore, it is also a global maximum because $\sum_{i=0}^{w-1}\frac{1}{f-i}$ is a monotonically decreasing function with f and, therefore, there is at most one value letting the first-order partial derivative of $\ln P_{fp}$ equal to zero. Consider the case when $p_e = (\frac{1}{2})^{\frac{m}{wn}\ln 2}$ is sufficiently small and thereby the value of Formula (9) is always greater than zero, P_{fp} must decrease as f decreases. This completes the proof. □

Discussion: Despite of the result stated in Theorem 4, we should bear in mind that we can NOT arbitrarily cut down the value of f to minimize P_{cf} and P_{fp}, because we need an adequate number of codewords to accommodate all potential set IDs of input elements with cardinality $\aleph$. In other words, NBF should guarantee that $\binom{f}{w} \geq \aleph$. To this end, we can choose f and w that satisfy $w = f/2$ and thereby yield the minimum value of f under a given constraint $\binom{f}{w} \geq \aleph$. We will take this parameter setting in the later theoretical analysis in this paper.

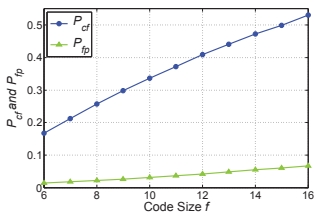

Figure 7: P_{cf} & P_{fp} **vs.** f **for NBF**

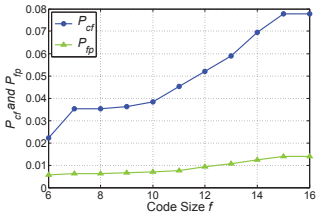

Figure 8: P_{cf} & P_{fp} **vs.** f **for NBF-E**

Figure 7 shows P_{cf} and P_{fp} when $m = 1.73\times10^6$, $n = 10^5$, $w = 3$, $k = 4$, and f increases from 6 to 16. We can see that both P_{cf} and P_{fp} decease smoothly as f decreases. This observation well supports Theorem 4.

4. ERROR CORRECTED NOISY BLOOM FILTER (NBF-E)

In this section, we incorporate asymmetric error-correcting codes into our NBF framework, which is called NBF-E. Our goal is to further suppress the classification failure and/or false positive. First of all, we present the motivations of applying asymmetric error-correcting codes. Then, we elaborate on utilizing it in the construction phase and query phase for NBF-E. Next, we calculate the classification failure rate and false positive rate of NBF-E, and study their optimization.

4.1 Optimization Using Asymmetric Error-Correcting Code

Our motivation for employing Asymmetric Error-Correcting (AEC) codes is to leverage the asymmetric error property of NBF to bring down the classification failure rate and/or false positive rate of NBF. As revealed by Lemma 1, the error rates for "0" bits in query results being corrupted are uniform and are given by $p_e = (1-(1-w/m)^{nk})^k$, but, on the contrary, the probability for a "1" bit to be corrupted is zero. On one hand, the error rates for "0" bits resembles the bit error rate of common memory-less channel models in information theory and coding theory that assume errors occur randomly and with a certain probability. This inspires us to use traditional error-correcting codes to enhance the resilience of the query results to noise. On the other hand, the one-sided error property of query results, *i.e.*, the crossover $0 \rightarrow 1$ occurs with positive probability p_e whereas the crossover $1 \rightarrow 0$ never occurs as illustrated in Figure 9, is naturally the property of binary asymmetric channel, or to be specific, Z channel [12]. There has emerged a large body of literature studying on AEC codes for asymmetric channel in recent years such as [12,20,21,26,27]. These proposed AEC codes are commonly as good as or better than Symmetric Error-Correcting (SEC) codes in terms of error-correcting ability and decoding speed. The root reason is that their search space for correcting errors is much smaller due to the one-sided error pattern of Z channel.

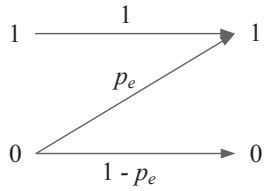

Figure 9: Binary Asymmetric Channel (Z Channel)

Notwithstanding, AEC codes have their weakness in slow encoding speed compared to SEC codes. For instance, the authors in [12] described a class of AEC codes such as Kim-Freiman codes, Constantin-Rao codes and Varshamov codes for which no simple algorithm is known. Surprisingly, our proposed constant weight codes capture the benefits of SEC codes in terms of fast encoding speed and the advantages of AEC codes in terms of strong error-correcting ability and fast decoding speed, while avoiding their major limitations. On one hand, we prove in Theorem 5 that AEC codes and SEC codes have exactly the same error-correcting ability for *present elements* that are already stored in the Bloom filter using constant weight codes, which means NBF-E can follow traditional encoding process as SEC codes do for constant weight codes. On the other hand, by the discussion for Theorems 8 and 9, the false positive rate for NBF-E using AEC codes is nearly half that of SEC codes in general cases, and,

meanwhile, its expected decoding speed is twice higher than SEC codes.

We emphasize that our disclosure of inherent asymmetric error property of our proposed Bloom filters and proposition of adopting AEC codes would shed light to future researches including not only NBF-E extensions but also other Bloom filer based schemes. For example, suppose we have a priori knowledge regarding the distribution of set IDs, then we can use non-constant weight codes for the sake of better memory efficiency (*e.g.*, assign codes with light Hamming weights to frequent set IDs like Huffman Coding does). In this scenario, AEC codes should have stronger error-correcting ability than SEC codes even for present elements [12]. Besides, the decoding algorithm for AEC codes can be applied to the E-COMB scheme proposed in [11] to accelerate its decoding speed.

4.2 Construction Phase

Basically, the construction phase for NBF-E is the same as that for NBF except for the construction of constant weight codes.

Unlike NBF that merely needs to enumerate all codewords with length f and constant weight w, NBF-E additionally requires all codewords share the same Hamming distance d between each other. Under such an extra constraint, the maximum number of codewords, which is called $A(f, d, w)$, is generally impossible to be computed directly. There has emerged a large body of works to study effective enumerative methods of constant weight codes such as Steiner systems method [2] and geometric encoding method [22]. Among these methods, the method proposed in [18] yields essentially the optimal results and has complexity scale of $O(n(\log n)^c)$ where $c \geq 2$ is some fixed constant. As enumerative methods of constant weight code is out of the focus of this paper, we simply adopt the enumerative method proposed in [18] to generate codewords.

We first present some necessary definitions and theoretical results for Hamming distance and asymmetric distance, and then propose a critical theorem for asymmetric error-correcting codes in our case.

Definition 1. *(Definition 2.6 in [12]) Given two codes* $x = (x_1, x_2, \ldots, x_n)$, $y = (y_1, y_2, \ldots, y_n) \in \{0,1\}^n$. *Let* $N(x, y) := \#\{i | x_i = 0 \text{ and } y_i = 1\}$.

(1) The Hamming distance is defined as:

$$d(x, y) := N(x, y) + N(y, x);$$

(2) The asymmetric distance is defined as:

$$\Delta(x, y) := \max\{N(x, y), N(y, x)\}.$$

Theorem 5. *Symmetric error-correcting codes and asymmetric error-correcting codes have the same error-correcting ability for present elements when using constant weight codes.*

Proof. By Lemma 2.1 in [12], we have $2\Delta(x, y) = d(x, y)$ for constant weight codes as all its codewords have the same weight. This also reveals the fact that the Hamming distance of constant weight codes $d(x, y)$ must be an even number. As a result, symmetric error-correcting codes for constant weight codes can correct up to $t_s = (d-2)/2 = (2\Delta(x, y) - 2)/2 = \Delta(x, y) - 1$ errors by Theorem 2.1.2 in [16], while asymmetric error-correcting codes can correct up to $t_a = \Delta(x, y) - 1$ errors by Theorem 2.1 in [12]. Then we have $t_s = t_a$, which means both of these two types of codes have the same error-correcting ability. □

Next, we present the Graham-Sloane lower bound, one of the most known lower bounds, for the maximum number of codewords $A(f, w, d)$ for binary constant weight codes.

Theorem 6. *(Theorem 4 in [10]) Let q be the smallest prime power (prime power is a positive integer power of a single prime number) satisfying $q \geq f$, the minimum Hamming distance of codes d is an even number, then*

$$A(f, d, w) \geq \frac{1}{q^{d/2-1}} \binom{f}{w}. \tag{10}$$

As $A(f, w, d)$ is hard to be exactly determined, one reasonable choice is to alternatively use its lower bound by requiring that the lower bound should be no less than the cardinality of potential set IDs of input elements, *i.e.* $\frac{1}{q^{d/2-1}}\binom{f}{w} \geq \aleph$.

4.3 Query Phase

Compared to NBF, the only difference of the query phase for NBF-E is the decoding procedure using AEC codes.

Despite of the existence of efficient error-correcting algorithms for constant weight codes like [7], we use the following simple decoding algorithm and show the advantages of AEC codes over SEC codes in terms of decoding speed. For any obtained query result, normally we scan over the whole codebook to single out the codeword having the minimum Hamming distance with the query result as the corrected code, just as traditional SEC codes do. Here is the difference between SEC codes and AEC codes: for SEC codes, they are unaware of asymmetric error pattern and thus attempt to correct all obtained results even if the Hamming weight of the results is less than the code size f, which means these results are indeed invalid codes. For AEC codes, they will identify and ignore all such invalid codes by checking the Hamming weight of the results before scanning the codebook for decoding.

4.4 Analysis

In this section, we first derive the expressions of classification failure rate and false positive rate for NBF-E, and then study their optimization. After that, we study the necessary condition to use NBF-E rather than NBF for delivering insights to real applications.

We present the mathematical expressions of the classification failure rate and false positive rate for NBF-E in the following two theorems.

Theorem 7. *Given the NBF-E size m, number of stored elements n, code length f, code weight w, number of hash functions k, and number of error-correcting bits t, the classification failure rate P_{cf} for NBF-E is given by*

$$\sum_{j=t+1}^{f-w} \binom{f-w}{j} p_e^j (1-p_e)^{f-w-j}, \tag{11}$$

where $p_e = (1 - (1 - \frac{w}{m})^{nk})^k$.

Proof. As the proof is similar to that to Theorem 1, we omit it here to save space. □

Theorem 8. *Given the NBF-E size m, number of stored elements n, code length f, code weight w and number of hash*

functions k, the false positive rate P_{fp} for NBF-E is given by

$$\sum_{j=w}^{w+t} \binom{f}{j} p_e^j (1-p_e)^{f-j}, \tag{12}$$

where $p_e = (1-(1-\frac{w}{m})^{nk})^k$.

PROOF. For NBF-E, due to its error-correcting capability, we consider the even worse case in which NBF-E takes all codes with weight j that satisfies $w \le j \le w+t$ as valid codes with or without corruption, which leads to false positive. Consequently, the false positive rate for NBF-E can be obtained by summing up all probability for j ($w \le j \le w+t$) number of bits are corrupted, which is given by $\binom{f}{j} p_e^j (1-p_e)^{f-j}$. Then, the result follows. □

Discussion: Strictly speaking, Equation (12) is indeed an upper bound of false positive rate because in fact not all codes with j ($w \le j \le w+t$) number of bits are wrongly taken as valid set ID codes. For example, when $j = w$, only $A(f,d,w)$ out of $\binom{f}{w}$ codes are mistakenly regarded as valid as stated in Section 4.2. Nevertheless, we approximately take it as false positive rate for simplicity of analysis. Moreover, by similar analysis in the above proof, it is easy to see that the false positive for SEC codes is $P'_{fp} = \sum_{j=w-t}^{w+t} \binom{f}{j} p_e^j (1-p_e)^{f-j}$. Considering general cases where w is supposed to be around $n/2$, then we have $\binom{f}{w-t} p_e^{w-t} (1-p_e)^{f-w+t} > \binom{f}{w+t} p_e^{w+t} (1-p_e)^{f-w-t}$, which means P'_{fp} is nearly two times of P_{fp} for AEC codes.

Next, we analyze optimization of P_{cf} and P_{fp} of NBF-E in terms of the number of hash functions k. Before that, we present a critical lemma below to assist further analysis.

LEMMA 2. *(Theorem 1 in [3]) For Binomial distribution $Binom(n,p)$, given that $p < \frac{k}{n}$, the upper bound of the lower tail of $Binom(n,p)$,* i.e. $F(k;n,p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}$, *can be given by*

$$F(k;n,p) \le \exp\left\{-n\left[\frac{k}{n}\ln\frac{k/n}{p} + (1-\frac{k}{n})\ln\frac{1-k/n}{1-p}\right]\right\}. \tag{13}$$

Though Lemma 2 offers an upper bound, [3] shows that the gap between this bound and the exact value is quite small (only 3% for the numerical example in [3]). In this paper, we use this bound rather than the lower tail of Binomial distribution for the convenience of optimization analysis by assuming that they vary at nearly the same trend, which is a much weaker assumption than assuming they have the same value. We will justify its correctness in later simulations for theoretical results based on this assumption.

The following theorem presents the optimal value of the number of hash functions for NBF-E.

THEOREM 9. *Given the fixed NBF-E size m, number of stored elements n, code length f, and code weight w, and error rate for "0" bits in a set ID code p_e is sufficiently small, both of the classification failure rate P_{cf} and false positive rate P_{fp} for NBF-E can be minimized when*

$$k = \frac{m}{wn}\ln 2. \tag{14}$$

PROOF. We first consider the classification failure rate P_{cf}. According to Theorem 7 and Lemma 2, the upper bound for the classification failure rate is given by

$$\exp\left\{-\left[(t+1)\ln\frac{t+1}{(f-w)p_e} + ((f-w)-(t+1))\ln\frac{(f-w)-(t+1)}{(f-w)(1-p_e)}\right]\right\}. \tag{15}$$

where $p_e = (1-(1-\frac{w}{m})^{nk})^k$ is the error rate for "0" bits in a set ID code. To minimize the above formula is equivalent to maximize

$$(t+1)\ln\frac{t+1}{(f-w)p_e} + ((f-w)-(t+1))\ln\frac{(f-w)-(t+1)}{(f-w)(1-p_e)}. \tag{16}$$

We take the first-order partial derivative of the above formula with respect to p_e and then have

$$\frac{\partial}{\partial p_e}\left\{(t+1)\ln\frac{t+1}{(f-w)p_e} + ((f-w)-(t+1))\ln\frac{(f-w)-(t+1)}{(f-w)(1-p_e)}\right\} = -\frac{t+1}{p_e} + \frac{f-w-t-1}{1-p_e}, \tag{17}$$

and therefore its second-order partial derivative is

$$\frac{\partial}{\partial p_e}\left\{-\frac{t+1}{p_e} + \frac{f-w-t-1}{1-p_e}\right\} = \frac{t+1}{p_e^2} + \frac{f-w-t-1}{(1-p_e)^2} > 0, \tag{18}$$

which means Formula (16) achieves its minimum at point $p_e = \frac{t+1}{f-w}$. As p_e is sufficiently small and is less than $\frac{t+1}{f-w}$, we need to minimize p_e in order to maximize Formula (16), and finally minimize the classification failure rate.

We proceed to consider the false positive rate P_{fp}. Similar to the proof to Theorem 3, we can prove that $\binom{f}{j} p_e^j (1-p_e)^{f-j}$ when $w \le j \le w+t$ should decrease as p_e decreases under the condition that p_e is sufficiently small such that $p_e < \frac{j}{n}$. Thus, it is clear that $\sum_{j=w}^{w+t} \binom{f}{j} p_e^j (1-p_e)^{f-j}$ should decrease as p_e decreases given that $p_e < \frac{w}{n}$.

To sum up, both of P_{cf} and P_{fp} can be minimized when $p_e = (1-(1-\frac{w}{m})^{nk})^k$ is minimized, which can be achieved when $k = \frac{m}{wn}\ln 2$ indicated by the proof to Theorem 3. This completes the proof. □

Discussion: By Theorem 9, we can easily reach the conclusion that every bit in NBF-E takes the equal probability to be 1 or 0. Furthermore, w is generally set to be around $n/2$. Due to this symmetry, NBF-E should have roughly the same chance to process query results with Hamming weight varies from w to $w+t$ and query results with Hamming weight from $w-t$ to $w-1$. While AEC codes skip computing the latter kind of results that are indeed false positives, SEC codes compute both of them, which means the average decoding speed of SEC is twice slower than that of AEC codes.

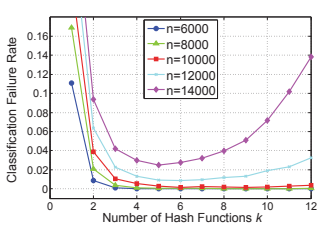

Figure 10: P_{cf} vs. k for NBF-E

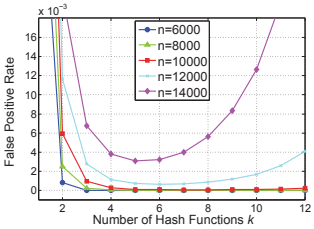

Figure 11: P_{fp} vs. k for NBF-E

Figures 10 and 11 show P_{cf} and P_{fp} when $m = 3 \times 10^5$, $f = 21$, $w = 3$, k increases from 1 to 12 and n increases from 6000 to 14000. The results well support Theorem 9. For instance, when $n = 6000, 8000, 10000, 12000, 14000$, P_{cf}

and P_{fp} achieve their minimum at $k = 11, 9, 6, 6, 5$ and at $k = 11, 8, 7, 6, 5$, respectively, while the theoretical optimal number of hash functions are $k = 11.55, 8.66, 6.93, 5.78, 4.95$, respectively.

THEOREM 10. *Both of the classification failure rate P_{cf} and the false positive rate P_{fp} for NBF-E decrease monotonically when the code size f decreases given that the number of hash functions k takes its optimal value,* i.e., $k = \frac{m}{wn} \ln 2$, *and the error rate for "0" bits in a set ID code p_e is sufficiently small.*

PROOF. We first consider the classification failure rate P_{cf}. Similar to the proof to Theorem 9, to minimize the upper bound of P_{cf}, we need to maximize

$$(t+1)\ln\frac{t+1}{(f-w)p_e} + ((f-w)-(t+1))\ln\frac{(f-w)-(t+1)}{(f-w)(1-p_e)}. \quad (19)$$

Taking the first-order partial derivative of the above formula with respect to f and considering the fact that $p_e = (1-(1-\frac{w}{m})^{nk})^k = (\frac{1}{2})^{\frac{m}{wn}}$ when $k = \frac{m}{wn}\ln 2$ has nothing to do with f, we obtain

$$\frac{\partial}{\partial f}\left\{(t+1)\ln\frac{t+1}{(f-w)p_e} + ((f-w)-(t+1))\ln\frac{(f-w)-(t+1)}{(f-w)(1-p_e)}\right\}$$
$$= \ln(1-\frac{t+1}{f-w}) - \ln(1-p_e), \quad (20)$$

and therefore its second-order partial derivative is

$$\frac{\partial}{\partial f}\left\{\ln(1-\frac{t+1}{f-w}) - \ln(1-p_e)\right\} = \frac{t+1}{f-w-t-1} > 0, \quad (21)$$

which means Formula (19) achieves its minimum at point $f = \frac{t+1}{p_e} + w$. As p_e is sufficiently small and therefore f is smaller than $\frac{t+1}{p_e} + w$, we need to minimize f in order to minimize P_{cf}.

Next we consider the false positive rate P_{fp}. Recall that we have proved in the proof to Theorem 4 that $\binom{f}{w}(\frac{1}{2})^{\frac{m}{n}\ln 2}[1-(\frac{1}{2})^{\frac{m}{wn}\ln 2}]^{f-w}$ decreases monotonically when f decreases given $k = \frac{m}{wn}\ln 2$. By similar analysis, we can prove $\binom{f}{j}(\frac{1}{2})^{\frac{mj}{wn}\ln 2}[1-(\frac{1}{2})^{\frac{m}{wn}\ln 2}]^{f-j}$ decreases monotonically as f decreases for $w \le j \le w+t$, and thereby $\sum_{j=w}^{w+t}\binom{f}{j}(\frac{1}{2})^{\frac{mj}{wn}\ln 2}[1-(\frac{1}{2})^{\frac{m}{wn}\ln 2}]^{f-j}$ decreases monotonically as f decreases as well. This completes the proof. □

Figure 8 illustrates P_{cf} and P_{fp} when $m = 1.73 \times 10^6$, $n = 10^5$, $w = 3$, $k = 4$, and f increases from 6 to 16. It can be seen that both of P_{cf} and P_{fp} decease as f decreases, which confirms Theorem 10.

Apart from parameter optimization for NBF-E, another crucial and natural question is that under what condition is it better to use NBF-E rather than NBF or vice versa. We consider a constrained case and present our theoretical findings in the following theorem.

THEOREM 11. *Given the fixed Bloom filter size m, number of stored elements n, number of hash functions k, and code weight w, and the error rate for "0" bits in a set ID code p_e is sufficiently small, the necessary (but not sufficient) condition for choosing NBF-E rather than NBF in terms of classification failure rate P_{cf} is*

$$\left(1-\left(1-\frac{w}{m}\right)^{nk}\right)^k < \frac{2(f-w)}{(f\frac{w}{w-1}-f)^2} \quad (22)$$

where f is the desirable code length for NBF, that is, the minimum integer satisfying $\binom{f}{w} \ge \aleph$ where $\aleph$ is the number of distinct set IDs of elements.

PROOF. Under the given parameter settings, NBF would choose a desirable code length f that is the minimum integer satisfying $\binom{f}{w} \ge \aleph$ where $\aleph$ is the cardinality of potential set IDs of input elements. For simplicity, we assume $\binom{f}{w} = \aleph$ for NBF. Essentially, if NBF-E is more preferable than NBF, it should be true that NBF-E with exact one error-correcting ability, namely $t = 1$, has less classification failure rate than the classification failure rate of NBF. Suppose the code length for such NBF-E is f', the above condition can be formally expressed as

$$\sum_{j=2}^{f'-w}\binom{f'-w}{j}p_e^j(1-p_e)^{f'-w-j} < 1-(1-p_e)^{f-w}, \quad (23)$$

or equivalently,

$$1-(1-p_e)^{f'-w} - (f'-w)p_e(1-p_e)^{f'-w-1} < 1-(1-p_e)^{f-w}, \quad (24)$$

where $p_e = (1-(1-\frac{w}{m})^{nk})^k$, following Theorem 1 and Theorem 7.

By rearranging Equation (24), we obtain

$$1+(f'-w-1)p_e > (1-p_e)^{-(f'-f-1)}. \quad (25)$$

According to the generalized Newton Binomial theorem [14], we have

$$(1-p_e)^{-(f'-f-1)}$$
$$= 1+(f'-f-1)p_e + \frac{1}{2}(f'-f)(f'-f-1)p_e^2 + O(p_e^3)$$
$$\approx 1+(f'-f-1)p_e + \frac{1}{2}(f'-f)(f'-f-1)p_e^2. \quad (26)$$

The last approximation holds since p_e is sufficiently small. Combing Inequality (25) and Equation (26) we have

$$p_e < \frac{2(f-w)}{(f'-f)(f'-f-1)} \approx \frac{2(f-w)}{(f'-f)^2}. \quad (27)$$

To simplify Inequality (27), we want to find an estimation of f'. By Theorem 6, f' is subject to $A(f',d,w) \ge \frac{1}{q^{d/2-1}}\binom{f'}{w}$ where q is the smallest prime power satisfying $q \ge f'$. As d should be at least $2t+2 = 4$ to ensure the error-correcting ability of NBF-E is at least 1 and q is approximately equal to f', this inequality can be also expressed as $A(f',d,w) \ge \frac{1}{f'}\binom{f'}{w}$. In practical design, we can set $\frac{1}{f'}\binom{f'}{w} = \aleph = \binom{f}{w}$ to guarantee the performance of NBF-E in the worst case. Then we have

$$f' = \frac{f'(f'-1)(f'-2)\dots(f'-w+1)}{f(f-1)(f-2)\dots(f-w+1)} > \left(\frac{f'}{f}\right)^w, \quad (28)$$

and therefore

$$f' < f^{\frac{w}{w-1}}. \quad (29)$$

Combining Inequalities (27) and (29) we have Inequality (22). Furthermore, as Inequality (22) is reached through a number of relaxations, it is indeed a necessary but not sufficient condition for selection. This completes the proof. □

Discussion: This result is consistent with our intuition that when the error rate for "0" bits $p_e = (1-(1-\frac{w}{m})^{nk})^k$ is quite small, the bit error rarely happens and most bits in the Bloom filter are 0, then applying NBF-E can effectively correct such error and reduce the classification failure rate,

while not introducing much noise to data in NBF. In contrast, the situation will be reversed when p_e becomes large because the limited error correcting ability of AEC codes is of little use to severe noise while the additionally introduced noise by AEC codes will make the situation even worse. In addition, as Inequality (22) is a necessary but not sufficient condition, the right side of this inequality is essentially a *lower bound* of the threshold for selection between NBF and NBF-E, and can approximately serve as a threshold in practice. Finally, we note that Theorem 11 may not suit the general case where f, k and w can be arbitrarily set.

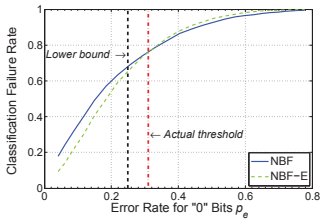

Figure 12: P_{cf} vs. f for NBF and NBF-E

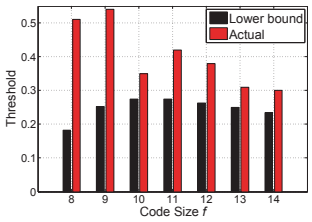

Figure 13: Lower Bound vs. Actual Value of Threshold for Selection

Figure 12 illustrates how P_{cf} changes when $m = 5 \times 10^4$, $f = 13$ for NBF and $f = 19$ for NBF-E, $w = 7$, $k = 3$, and P_e rises from about 0.04 to 0.78 for NBF and NBF-E. The results are shown in the solid blue curve and dotted green curve, respectively. We can observe P_{cf} for NBF-E is first superior, and then becomes inferior to that for NBF as P_e increases, which validates our discussion above. Furthermore, these two curves cross at point $P_e = 0.31$, as marked by the vertical red dot-dash line. This is the actual threshold value for selection between NBF and NBF-E. We also plot our theoretical lower bound of threshold $P_e = 0.25$ in the vertical black dash line in Figure 12. The lower bound of threshold $P_e = 0.25$ shown in the vertical black dash line in Figure 12 is unsurprisingly smaller than the actual threshold. The difference of P_{cf} for NBF and NBF-E at the lower bound of P_e is very small, which means using the lower bound for selection between NBF and NBF-E is reasonable.

Moreover, Figure 13 shows the lower bound and actual threshold values when $m = 5 \times 10^4$, $w = 7$, $k = 3$, and f varies from 8 to 14 as $\aleph$ increases. It can be seen that our lower bound for the threshold for selection precisely holds, and on average, the theoretical bound is equal to 58.2% of the actual threshold value.

5. EVALUATION

In this section, we conduct experiments to validate our NBF and NBF-E schemes and compare them to state-of-the-art solutions for multi-set membership testing.

5.1 Experimental Setup

We briefly describe the dataset we use in the experiments, and describe comparison algorithms as well as related parameter settings.

To evaluate the performance of NBF/NBF-E and other comparison algorithms, we deployed a traffic capturing system on a 10Gbps link of a backbone router to collect trace data. The traffic capturing system contains two sub-systems each of which is equipped with a 10G network card and uses netmap to capture packets. Especially, only 5-tuple flow IDs of packets consisting of source IP, source port, destination IP, destination port, and protocol type are recorded because of the high read/write overhead to capture the entire high-speed traffic. Each 5-tuple flow ID is stored as a 13-byte string, and used as an element in the experiments. The set ID of the set that the flow is belonged to is artificially generated. Overall, we collected up to 10 million flow IDs wherein 8 million flow IDs are distinct.

We use four algorithms, *i.e.*, COMB [11], Summary Cache (SC) [8], kBF [23], and IBF [9] for comparison purposes. To optimize query speed for COMB, SC, IBF and NBF/NBF-E, we use the following simple yet effective accelerating technique that is widely adopted in implementing standard Bloom filter or its variants: fetch k hashed bits (or bitmaps for NBF/NBF-E) one by one, and check if the value of current bit (or value of intermediate query result after performing AND operation for all obtained bitmaps for NBF/NBF-E) is 0; if yes, terminate immediately and return negative answer. Note that this technique can be refined specifically for NBF/NBF-E by checking whether the Hamming weight of the current result is smaller than a threshold (w for NBF/NBF-E) rather than checking whether it is 0. The reason we don't use it is to make a fair comparison between NBF/NBF-E and other schemes.

For computing platform, we used a standard off-the-shelf desktop computer equipped with an Intel(R) Core i7-3520 CPU @2.90GHz and 8GB RAM running Windows 10 to do our experiments. Throughout the experiments, we use the following parameter settings unless otherwise stated. The memory space allocated for any algorithm is $m = 2.16 \times 10^6$ bits, the number of stored elements $n = 10^5$, the number of hash functions is $k = 4$, the code length $f = 7$ for NBF, and $f = 15$ for NBF-E, the constant Hamming weight of codes in NBF or NBF-E is $w = 3$, and the number of distinct set IDs is $\aleph = 35$.

5.2 Correctness Rate

Our results show that our schemes, especially NBF-E, have nearly the best correction rates among all algorithms when changing k and $\aleph$, which is 8.7 *times larger than kBF or IBF.* The correctness rate here means the ratio of the number of correctly answered query elements to the total number of all query elements. As Figures 14 and 15 illustrate, NBF always has similar correctness rate with COMB, and NBF-E outperforms NBF or COMB by about 13%. Furthermore, it can be observed that SC has the best correctness rate. This is not surprising since SC allocates one standard Bloom filter to each set, and each Bloom filter absolutely returns a positive answer if the query element belongs to its associated set. The reason why SC cannot achieve 100% correctness rate at some points in Figures 14 and 15 is due to query failure, which means there are more than two Bloom filters corresponding to at least two sets return a positive answer because of false positive of standard Bloom filter, and therefore SC cannot determine which one the query element belongs to. Nevertheless, SC suffers from high memory access overhead and lower querying processing speed, and doesn't scale well with the number of distinct set IDs $\aleph$ as will be detailed later.

5.3 False Positive

Our results show that the false positive rate for

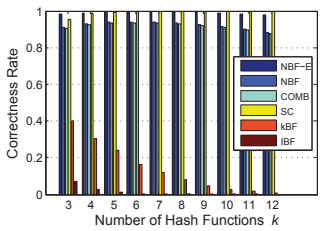

Figure 14: Correctness Rate vs. k

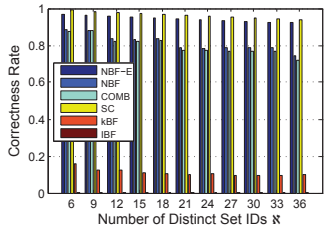

Figure 15: Correctness Rate vs. ℵ

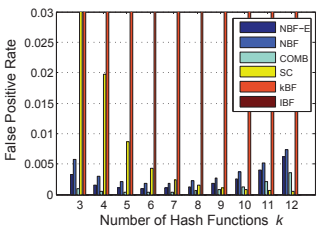

Figure 16: False Positive Rate vs. k

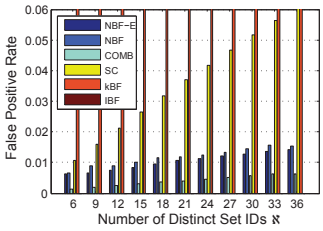

Figure 17: False Positive Rate vs. ℵ

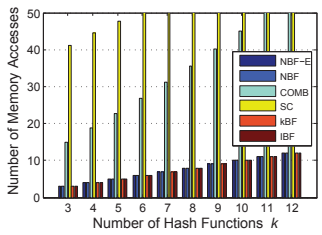

Figure 18: Memory Access vs. k when querying present elements

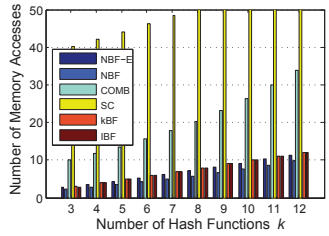

Figure 19: Memory Access vs. k when querying absent elements

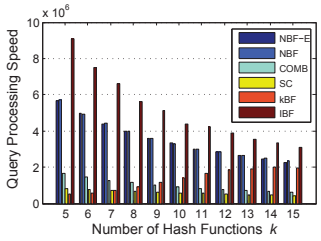

Figure 20: Querying Processing Speed vs. k when querying present elements

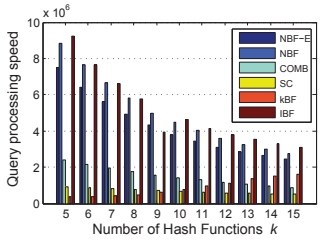

Figure 21: Querying Processing Speed vs. k when querying absent elements

NBF/NBF-E is about 6 *times less than kBF when changing* k *and* ℵ. From Figures 16 and 17, we observe that COMB generally exhibits better false positive rate than NBF/NBF-E as it uses k different hash functions to verify each bit in the query result while NBF/NBF-E uses k hash functions to verify all bits. However, this comes at the cost of much higher memory access overhead and query processing speed as will be described in Sections 5.4 and 5.5. The false positive rate for NBF/NBF-E is twice that of COMB and is smaller than 1%, which is practically acceptable. Moreover, the false positive rate for SC grows at a much faster speed than NBF/NBF-E and COMB as ℵ increases. The reason is that the assigned memory for each standard Bloom filter in SC shrinks rapidly as ℵ increases since the aggregate memory for all Bloom filters in SC in fixed, and, therefore, the false positive rate of each filter, as well as that of SC, rises dramatically. Note that the false positive rate for IBF is zero since it stores pairs of an element and its associated set ID, and returns the set ID only when the query element matches the associated element [9].

5.4 Memory Accesses

Our results show that NBF/NBF-E has a slightly smaller number of memory accesses compared to kBF and IBF, and uses only about 3.7 *and* 7.7 *times smaller memory accesses compared to COMB and SC respectively.* Figures 18 and 19 show the number of memory accesses for all six algorithms for present elements and absent elements respectively, when the number of hash functions k increases from 3 to 12. Here, present elements refer to those elements stored in the Bloom filters of these schemes while absent elements do not. It can be seen that all six algorithms have fewer memory accesses for false positives than that for present elements. This is because when using the accelerating technique described above, all these algorithms are capable to identify those false positives before carrying out all possible memory accesses, which are indispensable for present elements. We also observe that NBF has slightly fewer memory accesses than NBF-E for false positives. The reason is that NBF has higher opportunity to be terminated early than NBF-E because it has fewer "1" bits in its encoding result and the intermediate query result is more likely to become 0.

5.5 Query Processing Speed

Our results show that NBF (NBF-E) has about 3.2, 5.9 *and* 5.8 *times (*3.3, 6.4 *and* 6.5 *times) faster query processing speed compared to COMB, SC and kBF, when changing* k *and* ℵ. Figures 20 and 21 demonstrate the query processing speed, *i.e.* number of processed queries per second, for present elements and absent elements respectively, when the number of hash functions k rises from 5 to 15. We can see that all six algorithms have faster query processing speed for absent elements than that for present elements. The reason is the same as that of memory accesses, *i.e.*, the accelerating technique speeds up the querying process for absent elements. Moreover, due to the same reason why NBF outperforms NBF-E in terms of memory accesses, the query processing speed of NBF also exceeds that of NBF-E by nearly 16% for absent elements. As also shown in the two figures, IBF has the fastest query processing speed. This is because most cells in IBF in this scenario are mixed by two or more set IDs and cannot be decoded, IBF can quickly discover this fact by checking the count field in these cells and return "not found", which is much efficient than other algorithms.

6. CONCLUSION

The key contribution of this paper is to propose Noisy Bloom Filter (NBF) and Error Corrected Noisy Bloom Filter (NBF-E) for multi-set membership testing. The key advantages of NBF and NBF-E over the prior art are high space efficiency and high query processing speed. The key technical depth of this paper is in the analytical modeling of NBF and NBF-E, optimizing system parameters, finding the

minimum classification failure rate and false positive rate, and criteria of selection between NBF and NBF-E. We validated our analytical models through simulations using real world network traces. Our experimental results show that NBF and NBF-E significantly advance state-of-the-art solution regarding multi-set membership testing with 3.7 times smaller memory accesses and 3.3 times faster query processing speed.

Acknowledgment

The work is partly supported by the National Natural Science Foundation of China under Grant Numbers 61502229, 61472184, 61373129, and 61321491, the Huawei Innovation Research Program (HIRP), and the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program. Alex X. Liu is also affiliated with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA.

7. REFERENCES

[1] https://software.intel.com/en-us/articles/data-alignment-when-migrating-to-64-bit-intel-architecture.

[2] E. Agrell, A. Vardy, and K. Zeger. Upper bounds for constant-weight codes. *IEEE Transactions on Information Theory*, 46(7):2373–2395, 2000.

[3] R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of mathematical biology*, 51(1):125–131, 1989.

[4] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese. Beyond bloom filters: from approximate membership checks to approximate state machines. *ACM SIGCOMM Computer Communication Review*, 36(4):315–326, 2006.

[5] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 30–39, 2004.

[6] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese. What's the difference?: efficient set reconciliation without prior context. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 218–229. ACM, 2011.

[7] T. Etzion and A. Vardy. A new construction for constant weight codes. In *IEEE International Symposium on Information Theory and its Applications (ISITA)*, pages 338–342, 2014.

[8] 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.

[9] M. T. Goodrich and M. Mitzenmacher. Invertible bloom lookup tables. In *the 49th IEEE Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799, 2011.

[10] R. L. Graham and N. Sloane. Lower bounds for constant weight codes. *IEEE Transactions on Information Theory*, 26(1):37–43, 1980.

[11] F. Hao, M. Kodialam, T. Lakshman, and H. Song. Fast dynamic multiple-set membership testing using combinatorial bloom filters. *IEEE/ACM Transactions on Networking*, 20(1):295–304, 2012.

[12] T. Kløve. *Error correcting codes for the asymmetric channel*. Technical Report. Department of Pure Mathematics, University of Bergen, 1981.

[13] M. Lee, N. Duffield, and R. R. Kompella. Maple: A scalable architecture for maintaining packet latency measurements. In *Proceedings of the ACM conference on Internet measurement conference*, pages 101–114, 2012.

[14] C.-s. Liu. The essence of the generalized newton binomial theorem. *Communications in Nonlinear Science and Numerical Simulation*, 15(10):2766–2768, 2010.

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

[16] I. P. Naydenova. Error detection and correction for symmetric and asymmetric channels. 2007.

[17] Y. Qiao, T. Li, and S. Chen. Fast bloom filters and their generalization. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):93–103, 2014.

[18] B. Ryabko. Fast enumeration of combinatorial objects. In *Discrete Mathematics and Applications*, 1998.

[19] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, 2004.

[20] L. G. Tallini and B. Bose. Reed-muller codes, elementary symmetric functions and asymmetric error correction. In *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 1051–1055. IEEE, 2011.

[21] L. G. Tallini and B. Bose. On L_1 metric asymmetric/unidirectional error control codes, constrained weight codes and σ-codes. In *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 694–698, 2013.

[22] C. Tian, V. Vaishampayan, N. Sloane, et al. A coding algorithm for constant weight vectors: a geometric approach based on dissections. *IEEE Transactions on Information Theory*, 55(3):1051–1060, 2009.

[23] S. Xiong, Y. Yao, Q. Cao, and T. He. kBF: A Bloom Filter for key-value storage with an application on approximate state machines. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 1150–1158. 2014.

[24] M. K. Yoon, J. Son, and S.-H. Shin. Bloom tree: A search tree based on bloom filters for multiple-set membership testing. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 1429–1437. 2014.

[25] M. Yu, A. Fabrikant, and J. Rexford. Buffalo: Bloom filter forwarding architecture for large organizations. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, pages 313–324. ACM, 2009.

[26] J. Zhang and F.-W. Fu. Constructions for binary codes correcting asymmetric errors from function fields. In *Theory and Applications of Models of Computation*, pages 284–294. Springer, 2012.

[27] H. Zhou, A. Jiang, and J. Bruck. Nonuniform codes for correcting asymmetric errors in data storage. *IEEE Transactions on Information Theory*, 59(5):2988–3002, 2013.