# Difference Bloom Filter: a Probabilistic Structure for Multi-set Membership Query

Dongsheng Yang*, Deyu Tian*, Junzhi Gong*, Siang Gao*, Tong Yang*†, Xiaoming Li*
*Department of Computer Secience, Peking University, China
†Collaborative Innovation Center of High Performance Computing, NUDT, Changsha, China, 410073

***Abstract*—Given $v$ sets and an incoming item $e$, multi-set membership query is to report which set contains item $e$. Multi-set membership query is a fundamental problem in computer systems and applications. All existing data structures cannot achieve small memory usage, fast query speed and high accuracy at the same time. In this paper, we propose a novel probabilistic data structure named Difference Bloom Filter (DBF) for fast multi-set membership query, which not only is more accurate than the state-of-the-art, but has a faster query speed. There are two key design principles for DBF. The first one is to make the representation of the membership of elements exclusive by writing different number of 1s and 0s in the same filter, and the second one is to use the slow but cheap DRAM memory to improve the accuracy of the filter on the fast but expensive SRAM memory. Experimental results show that in terms of accuracy, DBF has a great advantage compared to state-of-the-art, being hundreds of times more accurate than the state-of-the-art vBF and ShBF. Furthermore, we have made the source code of our DBF available at our homepage [1] and GitHub [2].**

## I. Introduction

### A. Background and Motivation

There are $v$ sets $E_1$, $E_2 \dots E_v$, each two of which have no intersection. Given an item $e$, which set does item $e$ belong to? This is called multi-set query problem. Multi-set query with fast speed is the fundamental issue in various fields of computer networks, such as network packets processing [3]–[5], network traffic measurement [6], deep packet inspection [7], and more [8].

For example, in data centers, multi-set query mechanism determines the performance of the core device – the data center switch. To achieve fast forwarding speed, each switch keeps a MAC address table, which associates a destination MAC address with an outgoing port. The number of entries of the MAC address table is tens of or hundreds of thousands in practice. One straightforward solution is to use a hash table to store and query the MAC table. However, due to the large memory usage, the hash table is forced to be stored in the slow memory (DRAM). Moreover, the worst query performance of hash table is unbounded. Therefore, the query speed of hash table can hardly catch up with the line speed. To address this problem, several Bloom filter based algorithms have been proposed, such as the well known vBF [4] and the very recent work ShBF [9]. Bloom filters are memory efficient enough to be stored in fast memory (SRAM). Compared with DRAM, SRAM is much faster, more expensive, and limited in size. Bloom filters based algorithms can achieve high query speed at the cost of a certain error rate. Since handling errors need additional overhead, the error rate can affect the performance of the Bloom filter based algorithm to a large extent. Therefore, it is highly demanding to develop a new scheme for multi-set membership query to achieve a much lower error rate with fast query speed.

Corresponding author: Tong Yang (Email: yang.tong@pku.edu.cn). This work was done by Dongsheng Yang, Deyu Tian, Junzhi Gong, and Siang Gao under the guidance of their mentor: Tong Yang.

### B. Limitations of Prior Art

As mentioned above, the hash table often needs too much memory to be stored in fast but small SRAM. Meanwhile, the Bloom filter based algorithms are not accurate enough. To better understand prior art, it is worth introducing the Bloom filter and the so far most space efficient algorithm, vBF [4], in details.

The Bloom filter (BF) [10] is a data structure for recording elements and reporting whether an element is in a set or not. A BF consists of $m$ bits with $k$ hash functions. When inserting an item $e$, BF first maps $e$ to $k$ bits by computing the $k$ hash functions, and then sets the $k$ bits to 1. When querying an item $e$, BF checks the $k$ bits that $e$ is mapped to. If all of them are 1, BF reports true. Otherwise, BF reports false. Given $v$ sets, the vBF algorithm [4] builds $v$ Bloom filters, one Bloom filter per set. When querying an element $e$, vBF performs query operation on all the $v$ Bloom filters. If $\text{BF}_i$ reports true, it then reports that $e$ is in set $i$. Otherwise, it reports that $e$ does not belong to set $i$. It can be proved that among all Bloom filter based algorithms, vBF can achieve the maximum possible information entropy, which is $-0.5 \times \log_2(0.5) \times 2 = 1$ when the probability that any bit in the Bloom filter is 1 is 0.5. Therefore, the accuracy of Bloom filter based algorithms cannot be improved with the limited SRAM memory.

That seems to be a barrier that cannot be transcended before we realize that the accuracy could be improved by taking advantage of the auxiliary information written in DRAM. If a DRAM data structure is used in the insertion process, the Bloom filters can be guided to record the elements in a better order. On the other hand, the DRAM data structure is *transparent* for queries, if DRAM memory is never accessed during the query process. Therefore, the query speed does not decrease though the DRAM data structure is introduced. The state-of-the-art solutions pay no attention to DRAM, which limits their performances.

### C. Proposed Difference Bloom Filters

In this paper, we design a novel probabilistic structure named Difference Bloom Filter (DBF) for the sake of small error rate and fast query speed. DBF consists of a SRAM filter and a DRAM chaining hash table. The SRAM filter is a $m$-bit array with $k$ independent hash functions. In the insertion process, elements in set $i$ are mapped to $k$ bits of the filter, of which arbitrary $k-i+1$ bits are set to 1, and other $i-1$ bits are set to 0. We call this $< i, k >$ constraint. If an incoming element is *conflicted* with other existed elements on one of the mapped $k$ bits, DBF uses the `dual-flip` strategy proposed in this paper to seek for conformity between the conflicted elements to make the bit shared. *The essence of dual-flip is to change a series of mapping bits of the filters, so as to make every inserted elements satisfy the $< i, k >$ constraint.* Notice that the dual-flip operations are assisted by the DRAM table. In the query process, DBF checks the $k$ bits mapped by the queried element $e$. If exactly $k-i+1$ of them are 1, $e$ is reported to be in set $i$. In the deletion process, DBF locates the $k$ bits mapped by $e$, and for each of them, DBF decides whether to reset it or not with the assistance of the DRAM table.

The *first design principle* of DBF is to make the representation of the membership of elements exclusive in the same filter. Therefore, the query result for an element is definite, and any element, if inserted successfully, will never be reported to be in other sets. The *second design principle* is to use the slow but cheap DRAM memory to improve the accuracy of the filter on the fast but expensive SRAM memory. During an insertion, the slow DRAM table is accessed to assist the dual-flip operation. However, DBF does not access DRAM during the query process, so the DRAM table is transparent for query and does not influence the query speed. In a nutshell, with the same SRAM memory, DBF has a much smaller error rate and faster query speed than the state-of-the-art at the cost of slower update speed.

### D. Key Contributions

The key contributions of this paper lie in following aspects.

1) First, we propose a novel filter, named Difference Bloom filter (DBF), which is used for fast multi-set membership query.
2) Second, we carried out theoretical analyses and extensive experiments, and results show that our DBF algorithm significantly outperforms the state-of-the-art algorithms in terms of accuracy.
3) Third, we release the source code of our DBF algorithm and all other related algorithms at our homepage [1] and GitHub [2].

## II. Related Work

The algorithms for fast multi-set membership query include the Summary Cache [11], perfect hashing [12], kBF [13], Bloomtree [14], Bloomier [15], Coded BF [6], Sparsely Coded Filters [16], Combinatorial BF [17], and iSet [18]. It is worth noting that those solutions are mostly based on Bloom filters.

The coded BF and the Bloomier are faster than vBF. They both convert the set id of the element to a binary string. The coded BF builds one BF for each bit of the binary string and inserts $e$ into the BFs corresponding to the bits with value 1. In contrast, for each bit, the Bloomier builds two BFs and inserts $e$ into one of the two BFs according to whether the bit of the binary string of $e$ is 1 or 0. These two algorithms need a significantly larger space than vBF.

Another solution is proposed by Fang *et al.* called Combinatorial BF. It uses a single Bloom filter and uses multiple sets of hash functions to encode the set id. It improves the performance of the data structure by using constant weight error correcting codes for encoding the set id. However, the Combinatorial BF is not memory efficient and fast enough.

A recent study proposed a new algorithm called the Shifting Bloom Filter (ShBF) [9]. Instead of building $v$ different filters for $v$ sets, it uses the location offset to record elements with different values in only one filter. To be specific, ShBF first hashes $e$ to $k$ bits. We call them $k$ bases to make things clear. Then instead of setting those $k$ bases to 1, for the elements in set $i$, it sets the $i-1^{th}$ bits from the right of every bases to 1. ShBF is the fastest algorithm and is as memory efficient as vBF, so it is the best solution so far.

## III. The Difference Bloom Filter

For a better understanding of DBF, we first introduce how DBF works for two sets. For more sets, the process is similar, and will be introduced later.

### A. Problem Formulation

Suppose there are two disjoint sets $E_1$ and $E_2$. We define the union of them to be $E$, and the collection of elements out of $E$ is denoted as $\overline{E}$. An element in $E_1$ is denoted as $e_1$, and an element in $E_2$ is denoted as $e_2$. The mission of DBF is to keep track of $E$ and handle membership query. In other words, DBF needs to report which set the queried element belongs to, $E_1$, $E_2$ or $\overline{E}$.

### B. The Structure of DBF

Before going into the details of our algorithm, we first describe the structure of our DBF. As shown in figure 1, DBF has two parts, a filter on SRAM and a chaining-hash table on DRAM. The filter is an array with $m$ bits, and the table consists of $m$ cells corresponding to the $m$ bits of the filter.

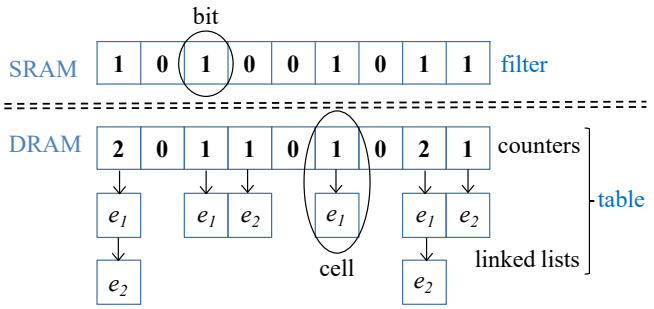


Fig. 1. Structure of DBF

Each cell is composed of a counter to count the number of elements in this cell, and a linked list which stores the information of the elements, including the key and the membership of the elements. DBF is associated with $k$ uniformly distributed independent hash functions $h_i(.)$ $(1 \leqslant h_i(.) \leqslant w)$.

### C. Building DBF for Static Sets

In this section, we first introduce how the query works in DBF, and then discuss the construction of DBF for two static sets $E_1$ and $E_2$.

*1) Query:* During the query process, DBF accesses only the fast SRAM memory. Suppose we query for an element $e$, which is mapped to $k$ positions by the $k$ hash functions $h_i(e)$ $(1 \leqslant i \leqslant k)$. The corresponding $k$ bits in the SRAM filter, $A[h_i(e)]$ $(1 \leqslant i \leqslant k)$, are checked. There are three possible results: 1) All these $k$ bits are 1, then it reports that $e$ belongs to $E_1$. 2) Only $k-1$ bits are 1, then it reports that $e$ belongs to $E_2$. 3) Less than $k-1$ bits are 1, then it reports that $e$ does not belong to these two sets, or we can say it belongs to $\overline{E}$.

*2) Construction:* The construction of DBF are divided into two steps, and we propose a strategy called dual-flip to deal with collisions.

The first step is to insert every element into DBF without distinguishing between $e_1$ and $e_2$. To be specific, every element is mapped to $k$ cells of the DRAM table, which are denoted as $B[h_i(e)]$ $(1 \leqslant i \leqslant k)$. For each of the $k$ cells, its counter is increased by 1, and the element is inserted into the linked list of the cell. Then, the corresponding bit in the SRAM filter, $A[h_i(e_1)]$, is set to 1. After this step, all $e_1$s are inserted successfully, but all $e_2$s are in wrong status.

The second step is to *settle* every $e_2$. To settle an $e_2$ means to turn exactly one bit of its $k$ mapped bits from 1 to 0, which will make its query result correct. It's simple to settle an $e_2$ that is not sharing all its $k$ cells with others. Among those $k$ cells, DBF finds out a cell whose counter is one, and turn the corresponding bit of the filter from 1 to 0. However, if an element $e_2$ is sharing all its $k$ bits with others, flipping (changing the bit from 1 to 0 in this case) any one of them will lead to conflicts. If we do not handle these conflicts, DBF will mistake some $e_1$s for $e_2$s and vice versa.

To address this issue, we propose a `dual-flip` strategy, which flips two bits of an existed element concurrently. Figure 2 shows an example of dual-flip. Dual-flip is adapted when DBF finds out that the counters of the $k$ cells that the inserted $e_2$ is mapped to are all larger than one. First, among those $k$ cells, DBF finds all cells in whose linked list all elements belong to $E_2$, denoted as $B[h_{find}(e_2)]$s. Then, for each element $e_2'$ in a $B[h_{find}(e_2)]$, we call it flippable if it satisfies one of the following two conditions. 1) $e_2'$ has not been settled. 2) $e_2'$ has already been settled, but it is the only element mapped to its 0 bit, denoted as $A[h_{find'}(e_2')]$. If all $e_2'$s sharing a $B[h_{find}(e_2)]$ are flippable, we flip $A[h_{find}(e_2)]$ and $A[h_{find'}(e_2')]$s synchronously, and the dual-flip of $B[h_{find}(e_2)]$ succeeds. The settling of $e_2$ will fail if the dual-flip efforts for all $B[h_{find}(e_2)]$s fail.

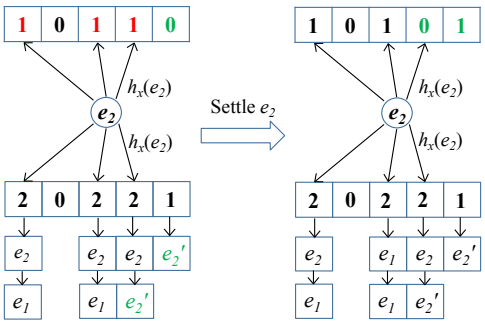


Fig. 2. Dual-flip strategy: This is a part of DBF. There is an $e_2$ shares all $k$ bits with others, so it cannot write a 0 directly. Notice that it shares the $4^{th}$ bit with $e_2'$, whose 0 bit is in the $5^{th}$ bit solely. Therefore, flipping the $4^{th}$ bit and the $5^{th}$ bit concurrently produces no side-effect.

### D. Dynamic Update of DBF

In this section, first we discuss the mechanics of dynamic insertion, then we present the details of deletion.

*1) Dynamic Insertion:* Similar to the construction process, the dynamic insertion also aims to set all $k$ bits of $e_1$ to 1, and to set $k-1$ bits of $e_2$ to 1 and the other one to 0. However, more conflicts will occur during this process. We divide them into two categories and discuss them respectively.

a) An inserted 1 conflicts with an existed 0: This will happen when a newly inserted $e_1$ conflicts with the 0 bit of an existing $e_2$, or a newly inserted $e_2$ conflicts with the 0 bit of two other existing $e_2$s. In this situation, we try to move the 0 away using the dual-flip strategy shown in Figure 3.

Specifically, if $e_1$ conflicts with a 0 at $A[h_x(e_1)]$, every element in the linked list of $B[h_x(e_1)]$ must be in $E_2$. We denote each of them as $e_{2i}$. For each $e_{2i}$, among the other $k-1$ cells it is mapped to, DBF tries to find a cell whose counter is 1, which is denoted as $B[h_{find}(e_{2i})]$. If such a cell is found for every $e_{2i}$, DBF sets $A[h_x(e_1)]$ to 1 and every $A[h_{find}(e_{2i})]$ to 0. Otherwise, DBF still sets $A[h_x(e_1)]$ to 1. In this way, it is ensured that every $e_1$ can be inserted correctly, at the cost of introducing errors of $e_2$s.

b) An inserted 0 is mapped to $k$ 1s: If all the $k$ bits that an incoming $e_2$ is mapped to have already been set to 1, one of these $k$ bits needs to be flipped from 1 to 0. The dual-flip

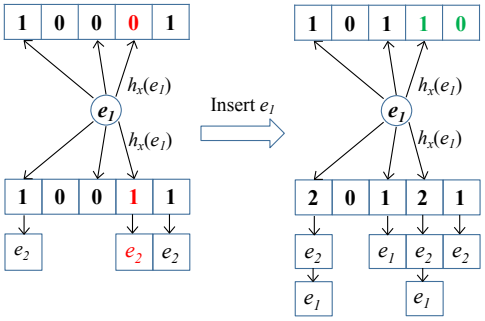


Fig. 3. Dual-flip strategy: This is a part of DBF. The inserted element $e_1$ conflicts with an $e_2$ on the $4^{th}$ bit. The $5^{th}$ cell holds this $e_2$ solely, so flipping the $4^{th}$ bit and the $5^{th}$ bit concurrently produces no side-effect.

strategy can be applied in the same way with the construction process.

*2) Deletion:* To perform a deletion, DBF first modifies the DRAM table, then updates the SRAM filter according to the status of the table. The details are presented below.

First, it calculates the values of the $k$ hash functions $h_i(e)$ $(1 \leqslant i \leqslant k)$, and finds the $k$ cells $B[h_i(e)]$ $(1 \leqslant i \leqslant k)$ in the DRAM table. Second, it deletes the entries of $e$ in the linked lists of these $k$ cells, and decreases the counters of the $k$ cells by 1. Third, if the counter of a cell $B[h_x(e)]$ has been decreased to 0, the corresponding bit in the SRAM filter, which is $A[h_x(e)]$ in this case, will be set to 0.

### *E. Periodically Refresh of DBF*

Although our DBF supports dynamic updates, it will have a better performance if it is rebuilt periodically. With the assistance of the DRAM table, the reconstruction of DBF can be done *in situ*. First, every $e_2$ in DBF is recorded in a queue. Second, every bit corresponding to a non-empty cell is set to 1. Third, every $e_2$ in the queue is settled in the same way with the construction process.

### *F. DBF for Multi-set Membership Query*

The DBF for 2-set can be extended naturally to handle multi-set membership query. We denote the application of DBF to $v$-set as DBF$_v$. To insert an element in $E_i$, DBF$_v$ maps it to $k$ bits of the filter, and try to set $k-i+1$ bits of them to 1 and $i-1$ bits to 0. If conflicts occur during the insertion, the dual-flip strategy is applied to resolve them. During the construction, the set with larger id is settled earlier, which can decrease the overall error rate. For the query of an element $e$, DBF$_v$ checks the number of 1s among the $k$ bits it is mapped to. It reports $e$ belongs to $E_i$ if there are exact $k-i+1$ $(1 \leqslant i \leqslant v)$ 1s, and reports $e$ belongs to $\overline{E}$ if there are less than $k-v+1$ 1s. To delete an element $e$, it is deleted in the table first. If the counter of a cell is decreased to 0, the corresponding bit in the filter is reset to 0.

## IV. Analysis

In this section, we will analyze the performance of DBF for 2-set membership query theoretically. The DBF algorithm may suffer the following types of errors:

1) $f_1$: $e_1$ is reported to be in $E_2$.
2) $f_2$: $e_2$ is reported to be in $E_1$.
3) $f_3$: $\overline{e}$ is reported to be in $E_1$.
4) $f_4$: $\overline{e}$ is reported to be in $E_2$.

First we present some conclusions of the Bloom filter proved in [19]. If we know the total number of inserted elements $n$ and the length of BF $m$, there is an equation to calculate the optimal value of the number of hash functions $k$, and we use that equation to adjust the parameters of DBF.

$$k = ln2 \times \frac{m}{n} \tag{1}$$

According to the insertion strategy of DBF, if a conflict occurs, $e_1$ is guaranteed to be inserted successfully. As a result, all the $k$ bits mapped by $e_1$ are set to 1, and $e_1$ will never be reported to be in $E_2$.

$$P(f_1) = 0 \tag{2}$$

$f_2$ comes from insertion failures. To analyze the probability of insertion failures, first we should know how many cells are mapped by $e_1$ and how many are mapped by $e_2$. For a certain cell, the probability that it is **not** mapped by $e_1$ is $\left(1-\frac{1}{m}\right)^{k|E_1|}$. Note that $k$ is determined by $ln2 \times \frac{m}{|E_1|+|E_2|}$, so the expression above can be simplified as $2^{-\frac{|E_1|}{|E_1|+|E_2|}}$. For an arbitrary cell, the probability that it is mapped by $e_1$ is defined as $I_1$. Similarly, that for $e_2$ is defined as $I_2$.

$$I_1 = 1 - 2^{-\frac{|E_1|}{|E_1|+|E_2|}} \tag{3}$$

$$I_2 = 1 - 2^{-\frac{|E_2|}{|E_1|+|E_2|}} \tag{4}$$

In the building process, an insertion failure of $e_2$ takes place in these two situations:

1) All cells that $e_2$ is mapped to are occupied by $e_1$s. The probability is $I_1^k$.
2) Some of the cells that $e_2$ is mapped to are occupied by $e_1$s, and other cells are being occupied by $e_2'$s which fail the dual-flip operation. For an arbitrary cell, the probability that it holds another $e_2'$ is $I_2^2$, and the probability that the 0 bit of this $e_2'$ is shared by two or more elements is also about $I_2^2$.

Taking both situations into account, we deduce $P(f_2)$ for DBF using dynamic insertion as follows:

$$P(f_2) \approx \left(I_1 + I_2^4\right)^k \tag{5}$$

In order to deduce $P(f_3)$ and $P(f_4)$, first we should estimate the number of 1s in the SRAM filter after the building process. 1 is used to determine the optimal value of $k$ for DBF, so there are about $\frac{m}{2}$ 1s in the filter before settling. Most of $e_2$s can be settled successfully, so there are about $\frac{m}{2} - |E_2|$ 1s after settling.

$f_3$ happens when all the $k$ bits that the query element is mapped to are 1.

$$P(f_3) \approx \left(\frac{m-2|E_2|}{2m}\right)^k \tag{6}$$

$f_4$ happens when arbitrary $k-1$ of the $k$ bits that the query element is mapped to are 1, and the other bit is 0.

$$P(f_4) \approx k\left(\frac{m-2|E_2|}{2m}\right)^{k-1}\left(\frac{m+2|E_2|}{2m}\right) \tag{7}$$

The performance of DBF is relative to the ratio between $E_1$ and $E_2$. All of the four kinds of errors tend to decrease when the percentage of $E_1$ decreases. Given the amount of $E_1$, $E_2$ and $\overline{E}$, we can calculate the overall error rate of DBF (denoted as $P(f)$).

$$P(f) = \frac{P(f_2)|E_2| + (P(f_3)+P(f_4))|\overline{E}|}{|E_1|+|E_2|+|\overline{E}|} \tag{8}$$

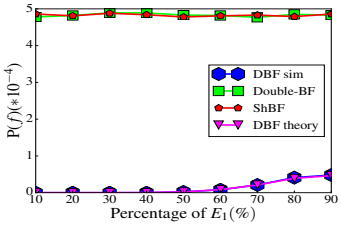


Fig. 4. Error rate vs. percentage of $E_1$

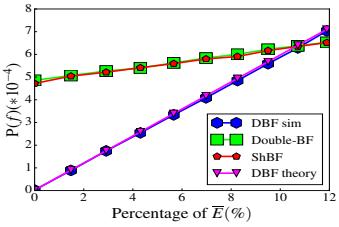


Fig. 5. Error rate vs. percentage of $\overline{E}$

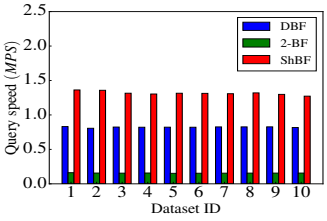


Fig. 6. Query Speed

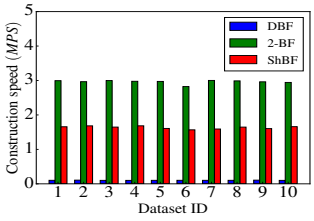


Fig. 7. Construction Speed

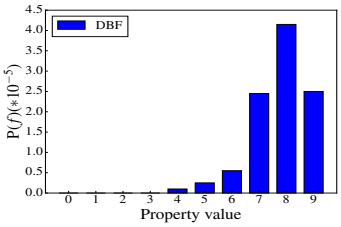


Fig. 8. Distribution of errors of $\text{DBF}_{10}$

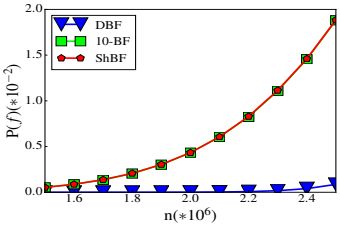


Fig. 9. Error rate vs. n

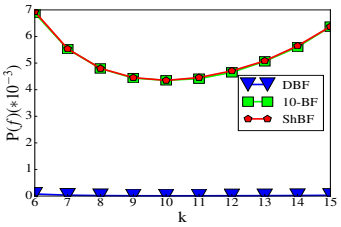


Fig. 10. Error rate vs. k

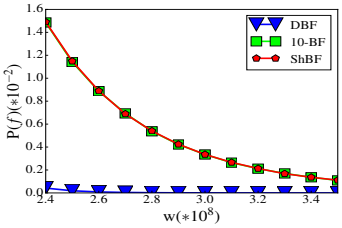


Fig. 11. Error rate vs. m

## V. Experimental Results

In this section, we present the experimental results of DBF, vBF and ShBF for 2-set membership query and multi-set membership query.

### A. Experimental Setup

All of the experiments are running on a machine with 12-core CPUs (24 threads, Intel Xeon CPU E5-2620 @2 GHz) and 62 GB DRAM memory. The datasets used in the experiments are generated to simulate MAC addresses and their destination ports. In all experiments, the SRAM usage of DBF, vBF and ShBF are all the same, which is $m$ bits, while the DRAM usage has no limitation.

### B. 2-set Membership query

We implement DBF, 2-BF (vBF for 2-set) and ShBF for 2-set membership query to verify the equation 8.

*1) Overall Error Rates with Changing Ratio Between $E_1$ and $E_2$: With the percentage of $E_1$ varies from 90% to 10%, our experimental and theoretical results show that DBF is $10 \sim 10^7$ times more accurate than 2-BF and ShBF.* In this experiment, $k$ is set to 10, $n$ is set to $20M$, and $m$ is calculated by the optimal equation 1, which is $289M$. The memory usage of the two bloom filters of 2-BF are adjusted dynamically according to the percentage of $E_1$ and $E_2$. To test the overall error rates of these three structures, every inserted element is queried once, and no element in $\overline{E}$ is queried. Figure 4 shows that as the percentage of $E_1$ increases, the error rate of DBF increases. The overall error rate of DBF varies from 0 to $4.8 \times 10^{-5}$, which matches well with the theoretical values calculated by equation 8. When the proportion of $E_1$ is 10%, the experimental value is 0, and the theoretical value is $1 \times 10^{-11}$ On the contrary, the overall error rate of 2-BF and ShBF are staying around $4.8 \times 10^{-4}$.

*2) Overall error rates with Changing Ratio Between $E$ and $\overline{E}$: The experimental results show that DBF has the best performance when the proportion of the queries of $\overline{E}$ is less than 11%.* The experimental conditions remain the same. $k$ is 10, $n$ is $20M$, and $m$ is $289M$. The ratio between $E_1$ and $E_2$ is fixed to $1:1$. In addition to all the inserted elements, different number of elements in $\overline{E}$ are queried, which account for at most 12% of the total queried elements. Experimental results of Figure 5 show that the overall error rate of DBF increases from $2.4 \times 10^{-6}$ to $7.1 \times 10^{-4}$ with the increasing of the proportion of $\overline{E}$. By contrast, that of 2-BF and ShBF increases from $4.8 \times 10^{-4}$ to $6.5 \times 10^{-4}$. When the percentage of $\overline{E}$ is less than 11%, DBF has an advantage over the other two structures. From Figure 5, it can also be seen that the simulation values of $P(f_{DBF})$ satisfy the theoretical value calculated by equation 7 well.

### C. Multi-set membership query

In this section, MAC addresses are grouped into 10 sets denoted by $E_i$ $(1 \leqslant i \leqslant 10)$ according to their ports. The $E_i$s are of almost the same size. We implement $\text{DBF}_{10}$ with the algorithm of $\text{DBF}_v$, ShBF with offsets from 0 to 9, and 10-BF with 10 individual BFs. Next, We will test the performance of the three structures with various parameters.

*1) Query Speed and Construction Speed: The experimental results show that the query speed of $\text{DBF}_{10}$ is 5 times faster than that of the 10-BF algorithm.* In this experiment, we set $n$ to $20M$, $m$ to $289M$ for the three structures. The $k$ of 10-BF and ShBF is 10, which is their optimal value. And the $k$ of $\text{DBF}_{10}$ is 20, which is the practical best value for $\text{DBF}_{10}$. In the query process, we first fetch all the $k$ bits, and then judge which set the element is in. We use ten different datasets to repeat this experiment for ten times. As shown in Figure 6, the query speed of $\text{DBF}_{10}$ keeps at $0.82MPS$ when dataset changes, while that of ShBF is $1.3MPS$ and that of 10-BF is $0.15MPS$. In conclusion, though $\text{DBF}_{10}$ performs not so

well as the fastest existing algorithm, ShBF, it is significantly faster than the widely used algorithm, vBF.

Due to the accesses to the DRAM table, the construction and updating speed is the drawback of $\text{DBF}_{10}$. As shown in Figure 7, the speed of construction of $\text{DBF}_{10}$ is $0.1MPS$, while that of ShBF is about $1.6MPS$ and that of 10-BF achieves about $3.0MPS$. However, the construction and update speed is not a concerned problem for relatively static sets, which is the case of the MAC forwarding tables.

*2) Error rate of Elements in different sets: The experimental results show that $\text{DBF}_{10}$ tends to perform better for elements in the set with smaller id.* The experimental parameters are the same with the speed experiment, and each set is of the same size. Figure 8 shows that the error rates increase when the set id increases. Elements in set $1 \sim 4$ suffer no faults, or we can say the error rate is lower than $5 \times 10^{-7}$. Meanwhile, elements in set 9 have the highest error rate, $4.15 \times 10^{-5}$. The error rate of set 10 is not the highest, because it is the first settled set in the construction process.

*3) Overall error rate with Changing $n$: The experimental results show that $\text{DBF}_{10}$ is $24 \sim 5 \times 10^3$ times more accurate than 10-BF and ShBF when $n$ changes.* In this experiment, we set $m$ to $289M$, $k$ of 10-BF and ShBF to 10, and $k$ of $\text{DBF}_{10}$ to 20. Besides, $n$ varies from $15M$ to $25M$. Figure 9 shows that DBF performs the best and has a larger advantage when the demand for accuracy grows. The overall error rate of $\text{DBF}_{10}$ increases from $1.0 \times 10^{-7}$ to $8.6 \times 10^{-4}$, while that of 10-BF increases from $5.3 \times 10^{-4}$ to $1.9 \times 10^{-2}$ and ShBF is similar.

*4) Overall error rate with Changing $k$: The experimental results show that $\text{DBF}_{10}$ is 430 times more accurate than 10-BF and ShBF when the parameters of them are all set to their optimal value.* In this experiment, we set $m$ to $289M$ and $n$ to $20M$. $k$ of 10-BF and ShBF varies from 6 to 15, and $k$ of $\text{DBF}_{10}$ varies from 16 to 25, which is 10 larger than the former. The x-axis refers to $k$ of 10-BF and ShBF. Figure 10 shows that all of the three structures' error rates first go up and then go down. The overall error rate of $\text{DBF}_{10}$ reaches the minimum value $1.0 \times 10^{-5}$ when its $k$ is 20, while at the same time that of 10-BF and ShBF reach the minimum value $4.3 \times 10^{-3}$ when $k$ is 10. $\text{DBF}_{10}$ outperforms the other two structures no matter how $k$ changes.

*5) Overall error rate with Changing $m$: The experimental results show that $\text{DBF}_{10}$ is $30 \sim 2 \times 10^4$ times more accurate than 10-BF and ShBF when $m$ changes.* In this experiment, we set $n$ to $20M$, $k$ of 10-BF and ShBF to 10, and $k$ of $\text{DBF}_{10}$ to 20. Besides. We change $m$ from $240M$ to $350M$. Figure 11 shows that DBF always performs the best and has a greater advantage when the memory footprint becomes larger. The overall error rate of $\text{DBF}_{10}$ increases from $2 \times 10^{-7}$ to $4.2 \times 10^{-4}$, while that of 10-BF and ShBF increases from $1.1 \times 10^{-3}$ to $1.5 \times 10^{-2}$.

## VI. Conclusion

Multi-set query is an important issue in various fields of computer science. In this paper, we propose a novel probabilistic filter named `Difference Bloom Filter (DBF)` for fast multi-set membership query, which not only is more accurate than the state-of-the-art, but has a faster query speed. The key operation of DBF is the dual-flip. The essence of dual-flip is to change a series of mapping bits of the filters, so as to make every inserted elements satisfy the $< i, k >$ constraint. Theoretical analyses and extensive experimental results show that in terms of accuracy, DBF has a great advantage compared to state-of-the-art, being hundreds of times more accurate than vBF and ShBF. We believe that our DBF can be applied to many more fields with multi-set query problem.

## Acknowledgment

This work is partially supported by Primary Research & Development Plan of China (2016YFB1000304), National Basic Research Program of China (2014CB340400), NSFC (61232004, 61672061), and CAS-NDST Lab, ICT, CAS, 100190, Beijing, China, Special Fund for strategic pilot technology Chinese Academy of Sciences (XDA06040602).

## References

[1] "Our Homepage." http://net.pku.edu.cn/~yangtong/.
[2] "Source code of DBF." https://github.com/yangdsh/DBF.
[3] W. Bux, W. E. Denzel, T. Engbersen, A. Herkersdorf, and R. P. Luijten, "Technologies and building blocks for fast packet forwarding," *IEEE Communications Magazine*, vol. 39, no. 1, pp. 70–77, 2001.
[4] M. Yu, A. Fabrikant, and J. Rexford, "Buffalo: bloom filter forwarding architecture for large organizations." in *ACM Conference on Emerging NETWORKING Experiments and Technology, CONEXT 2009, Rome, Italy, December*, 2009, pp. 313–324.
[5] T. Yang, G. Xie, Y. Li, Q. Fu, A. X. Liu, Q. Li, and L. Mathy, "Guarantee ip lookup performance with fib explosion," *ACM SIGCOMM Conference*, vol. 44, no. 4, pp. 39–50, 2014.
[6] F. Chang, W. C. Feng, and K. Li, "Approximate caches for packet classification," vol. 4, pp. 2196–2207 vol.4, 2004.
[7] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel bloom filters," in *High PERFORMANCE Interconnects, 2003. Proceedings. Symposium on*, 2004, pp. 52–61.
[8] F. Hao, M. Kodialam, T. Lakshman, and H. Song, "Fast dynamic multiple-set membership testing using combinatorial bloom filters," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 1, pp. 295–304, 2012.
[9] T. Yang, A. X. Liu, M. Shahzad, and et al., "A shifting bloom filter framework for set queries," *Proceedings of the VLDB Endowment*, vol. 9, no. 5, pp. 408–419, 2016.
[10] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of The ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[11] L. Fan, P. Cao, J. M. Almeida, and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEEACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.
[12] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. M. A. D. Heide, H. Rohnert, and R. E. Tarjan, "Dynamic perfect hashing: upper and lower bounds," *Siam Journal on Computing*, vol. 23, no. 4, pp. 524–531, 1988.
[13] S. Xiong, Y. Yao, Q. Cao, and T. He, "kbf: a bloom filter for key-value storage with an application on approximate state machines," 2014.
[14] M. Yoon, J. Son, and S. Shin, "Bloom tree: A search tree based on bloom filters for multiple-set membership testing," 2014.
[15] D. Charles and K. Chellapilla, "Bloomier filters: A second look," 2008.
[16] Y. Lu, B. Prabhakar, and F. Bonomi, "Bloom filters: Design innovations and novel applications," no. 1, pp. 201–206, 2005.
[17] F. Hao, M. Kodialam, T. V. Lakshman, and H. Song, "Fast multiset membership testing using combinatorial bloom filters," 2009.
[18] Y. Qiao, S. Chen, Z. Mo, and M. Yoon, "When bloom filters are no longer compact: Multi-set membership lookup for network applications," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2016.
[19] M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2003.