An Improved Less Hashing Bloom Filter

Kuai Yu\textsuperscript{1,a}, Deyu Qi\textsuperscript{2,b*}, Changjian Zhang\textsuperscript{3}

\textsuperscript{1}School of Computer Science and Engineering South China University of Technology Guangzhou, China
\textsuperscript{2}School of Computer Science and Engineering South China University of Technology Guangzhou, China
\textsuperscript{3}School of Computer Science and Engineering South China University of Technology Guangzhou, China
\textsuperscript{a}travoryk@gmail.com, \textsuperscript{b*}Corresponding author: csa@scut.edu.cn

Abstract—Bloom filter is a useful data structure, which is often used in the membership query with allowing errors. However, high computational cost of the hash functions limits the performance of the Bloom filter. In this paper, we propose a new Bloom filter based on a single hash function named No-partition Single-hashing Bloom filter (NPSHBF). Compared with the Standard Bloom filter (SBF), we theoretically prove that the false positive probability of NPSHBF is approximately equal to SBF. At the same time, we theoretically prove that the processes of modulo are independent to each other, which greatly improves the querying performance of the Bloom filter. After theoretical verification, we can see from a series of experimental results that the false positive probability of NPSHBF is consistent with the theoretical speculation, and the querying efficiency and generating efficiency of NPSHBF are much higher than SBF.

1. INTRODUCTION
Nowadays, there are a lot of data screening and qualification requirements in network applications. The Bloom filter is widely used in these applications, but there are some disadvantages of using the Standard Bloom filters such as high requirements of the hash functions (independence and good randomness) and limited storage space, which can lead to insufficient computing performance. Therefore, how to reduce the consumption of computation and the false positive probability of Bloom filter has become a major focus of the study.

2. BACKGROUND AND MOTIVATION
This chapter will cover the background and motivation of this article. We also discuss the introduction of related work, such as Standard Bloom Filter and its variants, then explain the motivation of this article.

2.1. Related work
The Standard Bloom filter (SBF) \cite{2} is proposed by Bloom. SBF regards the hash area as m individual addressable bits. It is assumed that all bits in the hash area are first set to 0. Next, each message in the membership set to be stored is hash coded by k hash functions into k distinct bit addresses, say \( a_1, a_2, \ldots, a_k \), which are the indexes in the hash area. Then, all k bits addressed by \( a_i \) through \( a_k \) are set to 1. Finally, an SBF is established.
To query a new message, a sequence of k bit addresses, say \(b_2, ..., b_k\), is generated in the same manner as for storing a message. If all bits are 1, the new message will be accepted. If any of these bits is 0, the new message will be rejected. However, it does not guarantee that the new message is in the membership even all bits are 1. If the new message is not in the membership, but accepted by the Bloom filter, we will get a false positive result. We could analyze the false positive probability of the SBF.

Let \(p_0\) denote the probability that the address value of the hash area remains zero after \(n\) messages are hashed to the address value through \(k\) hash functions.

\[
p_0 = (1 - \frac{1}{m})^m
\]  

Let \(p_1\) denote false positive probability, which is the probability that \(n\) messages will be accepted when a new message is queried after they are inserted into the standard Bloom filter.

\[
p_1 = (1 - p_0)^k = (1 - (1 - \frac{1}{m})^m)^k = (1 - e^{-\frac{kn}{m}})^k
\]  

Fixed the value of \(n, m, k\), when \(k = (m/n)\ln 2\), the minimum value of can be taken, that is, at this time, the false positive probability of the SBF can reach the theoretical minimum.

Based on SBF, a lot of improved Bloom filters are proposed to improve the performance of SBF. The following will be classified according to the different emphases of optimization and improvement.

2.1.1. Counting Bloom filters (CBF): To solve the problem that SBF cannot delete elements, the idea of counting the Bloom filter [33] emerged. It provides options that can be deleted and are widely used by improving CBF [5] [34]. This method improves the changeability of SBF and also optimizes the number of hash functions to reduce the false positive probability of CBF when the size of CBF and the number of elements is fixed. But CBF needs to consume more storage space and hash functions with the same high requirements as SBF.

2.1.2. Dynamic Bloom filter: SBF only focuses on static sets, which cannot meet the requirements in some dynamic changing situations. Therefore, the improved dynamic Bloom filter [30] can expand the Bloom filter [25] and their improved Bloom filter [16] can realize the size of the dynamic extended hash field.

2.1.3. Partitions Bloom filters: One kind of partition is to group the hash functions. The total number of hash functions remains the same, and they are combined in different groups. Through the greedy algorithm, the group with the minimum filling rate of the hash field is obtained [4]. The algorithm reduces the false positive probability of SBF, but it needs more computing resources to get the optimal grouping, and still requires more hash functions. There are similar zoned Bloom filters and their applications [8].

2.1.4. Multiple Bloom filter: In general, it is an integrated Bloom filter. More than one Bloom filter structure is used to form a bloom filter group to reduce the false positive probability of the Bloom filter, such as double count Bloom filter [6], similar idea as [18]. Although this method can reduce the false positive probability, it needs more storage space and computing resources.

2.1.5. Novel structure: The performance of Cuckoo filter (CF) [31] is better than that of SBF when a large number of items are stored and the probability of false-positive is small, and the structure supports dynamic addition and deletion of items. Rdbf [22] proposes a novel algorithm, which has
faster filtering speed, lower false-positive, and less space than SBF. Both Cuckoo filter and Rdbf are alternative algorithms of the bloom filter, and the underlying logic is not the idea of Bloom filter.

Different from above studies, we focus on reducing the consumption of computation. In this aspect, we collectively called Less hash Bloom filters, several previous studies have attempted to achieve this goal. Kirsh and Mitzenmacher proposed that two hash functions \( h_1(x) \) and \( h_2(x) \) can be used as seed hash functions to construct new hash functions in the following ways: \( g_i(x) = h_i(x) + ih_2(x) \) [1], which we call Less Hashing Bloom filter(LBF). LBF reduces the number of hash functions with high requirements. Two hash functions with high requirements are used to obtain n hash functions with random distribution, but their independence is not guaranteed. Song et al. proposed using O (log k) hash functions as seed hash to produce k hash values, but still did not analyze the direct independence of hash values [3]. Magnus Skjegstad also proposed a Bloom filter using only one hash function, but multiple Bloom filters need to be connected in series to form a larger Bloom filter [9]. Besides, Lu et al. put forward one hash Bloom filters (OHBF) [10], which also belong to partition Bloom filters. OHBF only uses one hash function with high requirements, and make mutually prime partitions, but the process of proving the independence of mutually prime partitions is not rigorous.

2.2. Motivation
Focus on SBF, the idea of this paper is to reduce the consumption of hashing operations, we propose the NPSHBF that reduce number of hash functions to only one with no partition. Our work improving LBF mainly in three aspects. Firstly, NPSHBF is an alternative to SBF and can be more widely used than SBF. Secondly, because the process of modulo in NPSHBF is different from OHBF, we strictly proved the independence of modulo coprime process in theory and insure the false positive probability of NPSHBF is close to SBF. Thirdly, when generating (initializing and storing) the Bloom filters, we choose no partition for reducing the amount of calculation.

3. DESIGN AND THEORETICAL ANALYSIS OF NPSHBF
In this part, we will introduce the structure and design of NPSHBF. For better understanding, we will first introduce the structure of SBF. Second, we will describe the design of NPSHBF in detail. Then we will prove that mappings in the modulo process of NPSHBF is mutually independence. Finally, we will analyze and formulate the false positive probability of NPSHBF.

3.1. Design of NPSHBF
In SBF, as shown in Figure 1, the elements are mapped to the hash field by k hash functions, i.e., from \( U \rightarrow M \), where \( U \) is the space of elements, and \( M \) is the space of SBF. In NPSHBF, we divide this process into two mapping stages, \( U \rightarrow W \rightarrow M \), where \( U \) is the space of elements, \( W \) is the space of single hashing field, \( M \) is the space of NPSHBF. \( U \rightarrow W \) mapping process is \( f(x) \) shown in Figure 2, \( W \rightarrow M \) mapping process is \( f(x) \mod m_i \) shown in Figure 2, and the each of \( m_i \) must be smaller than the Bloom filter size \( m \).

The structure of NPSHBF is shown in Figure 2. SBF obtains k bit addresses through k hash functions, then all k bits are set to 1, while NPSHBF obtains a hash code through a hash function, and then modulo \( k \) coprime integers for k bit addresses, and finally k bits are set to 1. For example, let \( m_i \) represent coprime integers. Suppose \( i = 3 \), \( m_1 = 83 \), \( m_2 = 89 \) and \( m_3 = 97 \). It is assumed that hash value of \( h(x) = 5439 \). As \( h(x) \mod m_1 = 15 \), \( h(x) \mod m_2 = 64 \) and \( h(x) \mod m_3 = 61 \), then the corresponding 15th, 61th, 64th bits should be set to 1.

It can be seen from the above structure description that NPSHBF is not difficult to establish. We replace \( k \cdot 1 \) hash functions with modulo mapping, because the ALU of CPU has optimized the module execution unit, the calculation of modulo operation is tiny. NPSHBF reduces the number of hash functions to \( 1 / k \) of SBF, and does not need partition stage of OHBF. However, we need to meet the following conditions to make the false positive probability of NPSHBF close to that of SBF:
• the values generated during the modulo taking phase are independent of each other. To ensure the processes of modulo are independent, we need to modulo coprime integers.
• The false positive probability of NPSHBF is close to that of SBF when the coprime integers are close to the Bloom filter size m.
• The single hash function has good randomness in the partition range. This is proved in the following experimental results.

3.2. Independence of modulo
Before the proof, we use some notations from Number Theory, as shown in Table I.

The independence of modulo in NPSHBF can be written by \( f_i(x) = h(x) \mod m_i \), which means \( f_i(x)(1 \leq i \leq k) \) should be independent, further we need to prove the theorem 1.

| Items                        | Domains                  |
|------------------------------|--------------------------|
| \((a, b) = 1\)              | \(a, b\) are coprime integers |
| \(x \equiv a \mod m\)       | \(x \mod m = a\)            |
| \(a \mid b\)                | \(a\) divides \(b\)            |

Theorem 1. If \( h(x) \) is uniformly distributed in interval \([0, m_1 m_2 \ldots m_k - 1]\), If \( m_i \) are mutual primes, then \( f_i(x) \) is independent of each other, where \( 1 \leq i \leq k \), \( f_i(x) = h(x) \mod m_i \).

In order to prove theorem 1, we first need to introduce and prove two lemmas.
Lemma 1. Chinese Remainder Theorem (CRT). Let \( m_1, m_2, \ldots, m_k \) be pairwise positive coprime integers, and \( a_1, a_2, \ldots, a_k \) be positive integers. Modulo \( M = \prod_{i=1}^{n} m_i \), then the Congruence Equations set (3) has a unique solution \( x = (\sum_{i=1}^{n} a_i M_i y_i) \), where \( M_i = \frac{M}{m_i} \), \( y_i = (M_i^{-1}) \mod m_i \), \( 1 \leq i \leq n \).

\[
\begin{align*}
    x &\equiv a_1 \pmod{d_1} \\
    x &\equiv a_2 \pmod{d_2} \\
    \vdots \\
    x &\equiv a_n \pmod{d_n}
\end{align*}
\]

(3)

Proof of Lemma 1. Two steps to proof, firstly, prove the Equations set (3) has a solution, Secondly, prove the solution of the Equations set (3) is unique.

Since \( m_1, m_2, \ldots, m_k \) are coprime integers, we have \( (m_i, m_j) = 1 \) and \( M = m_i \ast M_j \), so that \( (m_i, M_j) = 1 \).

When \( j \neq k \), then \( m_i \mid M_j \), thus \( M_j M_j^{-1} = 0 \pmod{m_i} \).

\[
X = M_1 M_1^{-1} a_1 + M_2 M_2^{-1} a_2 + \ldots + M_k M_k^{-1} a_n \pmod{M}
\]

Let

\[
X = \begin{cases} 
X = 1 \times a_1 + 0 \times a_2 + \ldots + 0 \times a_n \equiv a_1 \pmod{d_1} \\
X = 0 \times a_1 + 1 \times a_2 + \ldots + 0 \times a_n \equiv a_2 \pmod{d_2} \\
\vdots \\
X = 0 \times a_1 + 0 \times a_2 + \ldots + 1 \times a_n \equiv a_n \pmod{d_n}
\end{cases}
\]

(4)

Equations set (4) means \( x = X \mod M \) is the solution of the Equations set (3). Then prove the uniqueness of the Equations set (3), we proof by contradiction.

Suppose \( x = Y \mod M \) is another solution of Equations set (3), then we get \( X = a_i \pmod{m_i} \), \( Y = a_i \pmod{m_i} \), where \( 1 \leq i \leq n \), thus \( X = Y \pmod{m_i} \). Since \( m_i \) are coprime integers, then \( X = Y \pmod{M} \), i.e., \( X = Y \) with the modulo \( M \).

Lemma 2. Let \( N \) be a set of non-negative integers uniformly distributed in the interval \( [0, abc - 1] \), where \( a, b, c \in \mathbb{Z}^+ \). Let \( X = (N \mod c) \) and \( Y = (Z \mod b) \). If \( c, b \) satisfy \( (c, b) = 1 \), then \( X, Y \) are independent random variables.

Proof of Lemma 2. From the probability theorem of random variable independence, we know that if we want to prove that \( X \) and \( Y \) are independent random variables, we must prove that \( P(X = p, Y = q) \) is equal to \( P(X = p) \ast P(X = q) \).

Since \( (c, b) = 1 \), thus \( u, v \in \mathbb{Z}^+ \), \( cu + bv = 1 \), let \( m = cu + bv \), then:

\[
P(X = p, Y = q) = P(N = p \mod c, N = q \mod b) \tag{5}
\]

\[
P(N = p \mod c, N = q \mod b) = P(N = m \mod cb) \tag{6}
\]

\[
P(N = m \mod cb) = P(N = m + i \ast cb)(0 \leq i \leq 1) \tag{7}
\]

\[
P(N = m + i \ast cb) = 1/cb \tag{8}
\]

(5) is can be deduced by definition, (6) is deduced by Lemma 1, (7)(8) is easy to understand.
The following (9)(10) are similar to above.

\[ P(X = p) = P(N = p \mod a) = 1/c \tag{9} \]
\[ P(X = q) = P(N = q \mod b) = 1/b \tag{10} \]

Finally, we prove equation (5) is true and we conclude \( X, Y \) are independent random variables.

Then we can prove the Theorem 1.

Proof of Theorem 1. If \((i, j) = 1, 1 \leq i < j \leq k\), let \( a = m_1m_2...m_k \mod m_i m_j \), since \( m_i \) are mutual primes, and \( h(x) \) is uniformly distributed in interval \([0, am_i m_j - 1]\), form Lemma 2, we know \( h(x) \mod m_i \) and \( h(x) \mod m_j \) are independent random variables, thus \( f_i(x) \) is independent of each other where \( 1 \leq i \leq k \).

So far, we have completed the first part of the proof. In the theorem, the value range of \( h(x) \) is \( X_{m_i} \), but in practical applications, we cannot generally take such accuracy, so let the value range of \( h(x) \) be \( H \) and \( H \geq X_{m_i} \) is enough.

3.3. False positive probability of NPSHBF

The false positive of NPSHBF may occur in two different processes. First, it may occur during the hash mapping process, which means hash collision. Second, it may occur in the modulo process. Let \( B \) represents the false positive occurs in hash mapping process. If no collision happens in the hashing, but a collision happens in the modulo process, which will also cause the false positive. In summary, the false positive probability of NPSHBF is:

\[ P(F) = P(B) + P(F \mid \neg B) P(\neg B) \tag{11} \]

\( P(F) \) represents the false positive probability of NPSHBF, and \( P(B) \) represents the probability of hash collision in the hash process. If the collision happens in this process, then the false positive must occur in NPSHBF. The false positive probability of hashing process is:

\[ P(B) = 1 - (1 - 1/H)^m \tag{12} \]

Where \( H \) is sufficiently large and much larger than the size \( m \) of NPSHBF (\( H >> m \)), the value of \( P(B) \) is close to zero and can be ignored. Then we analyze the probability when no collision happens in the hash process, but the false positive occurs. We could analyze the false positive probability of each modulo part:

\[ P_i = 1 - (1 - 1/m_i)^m \tag{13} \]

From theorem 1, we know \( P_i \) is mutually independent, thus:

\[ P(F \mid \neg B) P(\neg B) = \prod_{i=1}^{k} P_i = \prod_{i=1}^{k} 1 - (1 - 1/m_i)^m \tag{14} \]

Since \((1-(1 - \frac{1}{x})^x)\) decreases monotonically with respect to \( x \), thus:
\[(1-(1-1/m_{\text{max}})^n)^k \leq \prod_{i=1}^{k} P_i \leq (1-(1-1/m_{\text{min}})^n)^k \quad \text{(15)}\]

Where \( m_{\text{max}} = \max_i \{m_i\} \), \( m_{\text{min}} = \min_i \{m_i\} \).

Let \( m_{\text{max}} = m \):

\[(1-(1-1/m_{\text{max}})^n)^k = (1-(1-1/m)^n)^k \quad \text{(16)}\]

From equations (15) and (16), we can see that if \( m_{\text{max}} = m_{\text{min}} = m \), the false positive probability of NPSHBF is equal to SBF. However, in application, \( m_{\text{max}} \) and \( m_{\text{min}} \) will not be equal. We can make the \( m_{\text{max}}, m_{\text{min}} \) and \( m \) as close as possible, the false positive probability of NPSHBF also close to SBF. An optimal choice of \( m_1, ..., m_k \) is given as follows. We first choose \( m_k \) to be the largest prime number satisfies \( m_k \leq m \). Then, let \( m_i \) be the largest prime number satisfies \( m_i \leq m_{\text{min}} + 1, i=1, ..., k-1 \).

Algorithm 1 describe the method to find \( m_1, ..., m_k \) and Table II shows some examples using Algorithm 1.

After that, we can know the optimal false positive probability of OHBF is:

\[P_2 = (1-(1-m/k)^n)^k \quad \text{(17)}\]

If \( k > 1 \), we know Then we get:

\[P_1 \leq (1-(1-1/m_{\text{min}})^n)^k \leq (1-(1-1/m_{\text{min}})^n)^k = P_2 \quad \text{(18)}\]

So, we have proved that the false positive probability of NPSHBF is lower that of OHBF.

**Algorithm 1** Determine the value of coprime integers

**Input:** \( m, k \)

**Output:** \( m_1, ..., m_k \)

1: \( \text{temp} \leftarrow m \)
2: for \( i \) \( \text{from} \) 1 \( \text{to} \) \( k \) do
3: \( \text{if} \ \text{temp} \ \text{is prime integer} \) \( \text{then} \)
4: \( \quad m_i \leftarrow \text{temp} \ \text{and} \ i \leftarrow i + 1 \)
5: \( \quad \text{else} \ \text{temp} \leftarrow \text{temp} - 1 \)
6: end for

**TABLE II. EXAMPLES OF USING ALGORITHM 1 (k=5)**

| \( m \) | Value of coprime integers |
|-------|--------------------------|
| 10000 | 9973, 9967, 9949, 9941, 9931 |
| 20000 | 19997, 19993, 19991, 19979, 19973 |
| 40000 | 39989, 39983, 39979, 39971, 39953 |
| 80000 | 79999, 79997, 79987, 79979, 79973 |
| 16000 | 159979, 159977, 159937, 159931, 159911 |
| 32000 | 319993, 319981, 319973, 319967, 319937 |
| 64000 | 639997, 639983, 639959, 639949, 639941 |

It can be seen that if we make \( m_{\text{max}} \) and \( m_{\text{min}} \) close to \( m \), the theoretical false positive probability of NPSHBF is close to SBF and lower than that of OHBF, although the theoretical false positive probability of Bloom filter is an approximate calculation, the practical false positive probabilities are shown in next section.
4. EXPERIMENTS AND RESULTS

Generally speaking, the performance of Bloom filters can be evaluated in three different aspects: the false positive probability, querying efficiency and generating efficiency. In fact, practical false positive probability means the probability of being accepted by the Bloom filter after querying, querying efficiency can be represented by the time needed for querying n times, generating efficiency can be represented by the time needed for initializing and storing all data.

The m is the size of Bloom filter, k is the number of hash functions or modulo functions, n is the number of storing, p is the number of generating Bloom filters, q is the number of querying collection.

Then we focus on these three aspects for the following experiments, and there are some explanations before our experiments:

- The three Bloom filter structures we compare are SBF, OHBF, and NPSHBF. Because NPSHBF is improved based on SBF. Among the various Bloom filters mentioned in Chapter 2, NPSHBF belongs to the category of Less hash Bloom filters, OHBF also belongs to this category and is a well-performed one in the category so that it is included in the comparison. While other variants of Bloom filters such as CBF, Multiple Bloom filter, Multiple Bloom filter are superimposed with NPSHBF, and the underlying ideas of new structures such as Cuckoo filter and Rdbf are not same as the Bloom filter, so they are not included in the comparison.

- For better interpretation, we use tables for comparison of practical false positive probabilities of three Bloom filters.

- Use larger amounts of data and less regular data as data sources (about ten million different strings), some for storing, others for querying.

### TABLE III. PRACTICAL FALSE POSITIVE PROBABILITIES COMPARISON ON q=10000, k=6, n=10000

| m    | SBF sim  | OHBF sim | NPSHBF sim | NPSHBF gap | OHBF gap |
|------|----------|----------|------------|------------|----------|
| 10000| 6.106E-03| 8.357E-03| 8.454E-03  | 3.238E-02  | 3.042E-02|
| 15000| 1.024E-03| 1.291E-03| 1.266E-03  | 8.094E-02  | 8.463E-02|
| 20000| 3.311E-04| 3.210E-04| 3.070E-04  | 1.255E-01  | 2.884E-01|
| 25000| 9.131E-05| 9.600E-05| 9.500E-05  | 7.208E-02  | 2.920E-01|
| 30000| 3.010E-05| 3.200E-05| 3.000E-05  | 8.694E-02  | 3.606E-01|
| 35000| 1.547E-05| 1.600E-05| 1.400E-05  | 8.094E-02  | 8.385E-02|
| 40000| 1.439E-05| 1.000E-05| 7.000E-06  | 3.290E-02  | 1.559E-01|
| 45000| 2.540E-06| 7.000E-06| 4.000E-06  | 3.771E-02  | 1.463E-02|

### TABLE IV. PRACTICAL FALSE POSITIVE PROBABILITIES COMPARISON ON q=100000, k=6, n=10000

| m    | SBF sim  | OHBF sim | NPSHBF sim | NPSHBF gap | OHBF gap |
|------|----------|----------|------------|------------|----------|
| 10000| 9.561E-01| 9.852E-01| 9.871E-01  | 2.777E-01  | 3.687E-01|
| 15000| 8.301E-01| 9.004E-01| 8.973E-01  | 1.912E-01  | 2.607E-01|
| 20000| 6.539E-01| 8.425E-01| 7.360E-01  | 7.850E-02  | 3.050E-02|
| 25000| 5.162E-01| 6.669E-01| 5.534E-01  | 3.886E-02  | 5.139E-02|
| 30000| 3.758E-01| 5.113E-01| 4.084E-01  | 3.333E-03  | 6.312E-02|
| 35000| 2.911E-01| 3.155E-01| 3.146E-01  | 1.050E-01  | 3.426E-02|
| 40000| 2.142E-01| 2.476E-01| 2.212E-01  | 1.056E+00  | 3.051E-01|
| 45000| 1.637E-01| 1.613E-01| 1.576E-01  | 3.650E-01  | 1.756E+00|
4.1. False positive probabilities

As shown in Table III, the ‘sim’ represents simulation, the ‘NPSHBF gap’ column is calculated by $\frac{|NPSHBF_{sim} - SBF_{sim}|}{SBF_{sim}}$, we set $n = 10000$ and $k = 6$, and $q=100000$ in the parameters. Then we change the value of $m$ to test the false positive probability of each Bloom filter. The ‘gap’ represents how close it is to false positive probability of SBF, From the two tables, we can see that practical positive probabilities of NPSHBF are very close to SBF (as we explained in Section 4), and practical positive probabilities of NPSHBF is slightly lower than that of OHBF which consistent with our theoretical derivation. At the same time, the ‘NPSHBF gap’ is lower than ‘OHBF gap’ which means the practical false positive probabilities of NPSHBF is closer to SBF than that of OHBF. Additionally, as shown in table IV, the practical gap between NPSHBF and OHBF is more obvious when $m$ is not long enough.

4.2. Querying efficiency

In this subsection, we evaluate the querying rates of three Bloom filters. NPSHBF choose the md5 hash function (which is used in OHBF) for a fair comparison, in Figure1, Figure 2 and Figure4, the number of querying collection $q$ is set to 1000000 and the number of generating Bloom filters $p$ is set to 1.

Figure 3. Querying time of three Bloom filters with $n=10000$, the size $m$ varying in each subfigure, (a) $k=3$, (b) $k=6$.

Figure 4. Querying time of three Bloom filters with $m=100000$, the $n$ varying in each subfigure, (a) $k=3$, (b) $k=6$.

Figure 5. Querying time of three Bloom filters with $m=100000$, $n=15000$, the $q$ varying in each subfigure, (a) $k=3$, (b) $k=6$.

Figure 3 and Figure 4 show the querying time comparison result when $m$, $n$ varies. We can see from the four subfigures that the querying time of NPSHBF is much lower than that of SBF and almost same as OHBF. As the $k$ increases, the gap of querying time between OHBF and NPSHBF increases slowly, but the querying time gap between NPSHBF and SBF increases quickly. The reason is that OHBF and NPSHBF reduced $k-1$ hash functions of SBF. Figure 3 shows the querying time results, the querying time of SBF increases quickly when $q$ increases, but the querying time of OHBF and NPSHBF also increases as $q$ increases, but the rate increase is much slower compared with SBF.
4.3. generating efficiency
In this subsection, we evaluate the generating efficiency of three Bloom filters. Generating time includes initializing time and storing time. The size m is set to 100000.

Figure 6. Generating time of three Bloom filters with m=100000, the n varying in each subfigure, (a) k=3, (b) k=6.

Figure 7. Generating time of three Bloom filters with m=100000, n=15000, the p varying in each subfigure, (a) k=3, (b) k=6.

Figure 6 shows the generating time comparison result when n varies, Figure 7 shows the generating comparison result when p varies, we can see from four figures that NPSHBF takes less time for generating. The reason is OHBF and NPSHBF both use a single hash function, but NPSHBF reduces computational cost of partitions. As more Bloom filters are generated, the generating time gap between NPSHBF and OHBF increases.

5. CONCLUSIONS
The Bloom filters have been used in many applications. Because the optimization of Bloom filters has not been completed, we propose the NPSHBF based on a single hash function without partitions. NPSHBF reduces multiple hash functions of SBF to a single hash function, then uses k mutually independent modulo operations to replace k hash operations. This article verifies the feasibility of NPSHBF in theory and compare it with SBF and OHBF in experiments. The results show that the false positive probability of NPSHBF is close to SBF and lower than OHBF, the querying efficiency of NPSHBF is much higher than SBF and almost same as OHBF, and the generating efficiency of NPSHBF is higher than SBF and OHBF. In addition, we can easily extend NPSHBF to other Bloom filter variants, such as Counting Bloom filters, Dual Counting Bloom Filter, and Dynamic Bloom Filters.

REFERENCES
[1] Kirsch A, Mitzenmacher M. Less hashing, same performance: Building a better Bloom filter[J]. Random Structures and Algorithms, 2008, 33(2):187-218.
[2] Bloom, Burton H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7):422-426.
[3] Iqbal A A, Ott M, Seneviratne A. Simplistic Hashing for building a Better Bloom Filter on Randomized Data[C]// The 13th International Conference on Network-Based Information Systems, NBiS 2010, Takayama, Gifu, Japan, 14-16 September 2010. DBLP, 2010.
[4] Hao F, Kodialam M, Lakshman T V. building high accuracy bloom filters using partitioned hashing[C]// ACM, 2007:277-287.
[5] Kleyko D, Rahimi A, Gayler R W, et al. Autoscaling Bloom Filter: Controlling Trade-off Between True and false-positives[J]. Neural Computing and Applications, 2019:1-10.

[6] Dodig, Sruk I, Cafuta V, et al. Reducing false rate packet recognition using Dual Counting Bloom Filter[J]. Telecommunication Systems, 2018.

[7] Grandi F. On the analysis of Bloom filters[J]. Information Processing Letters, 2018, 129:35-39.

[8] Jiang M, Zhao C, Mo Z, et al. An improved algorithm based on Bloom filter and its application in bar code recognition and processing[J]. EURASIP Journal on Image and Video Processing, 2018, 2018(1).

[9] Magnus Skjegstad and Torleiv Maseng. 2011. Low complexity set reconciliation using Bloom filters. In Proceedings of the 7th ACM ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing (FOMC’11). Association for Computing Machinery, New York, NY, USA, 33–41. DOI:https://doi.org/10.1145/1998476.1998483

[10] Lu J, Yang T, Member, et al. Low Computational Cost Bloom Filters[J]. 2018.

[11] Sanchez-Macian A, Reviriego P, Maestro J A, et al. Single Event Transient Tolerant Bloom Filter Implementations[J]. IEEE Transactions on Computers, 2017, 66(10):1831-1836.

[12] Yunxue, Yan, Lei, et al. A dynamic integrity verification scheme of cloud storage data based on lattice and Bloom filter[J]. Journal of information security and applications, 2018.

[13] Zhang R, Liu J, Huang T, et al. Adaptive Compression Trie based Bloom Filter: Request Filter for NDN Content Store[J]. IEEE Access, 2017, PP(99):1-1.

[14] Meghana V, Suresh M, Sandhya S, et al. SoC implementation of network intrusion detection using counting bloom filter[C]// IEEE International Conference on Recent Trends in Electronics. IEEE, 2017.

[15] Basat R B, Einziger G, Friedman R, et al. Poster abstract: A sliding counting bloom filter[C]// Computer Communications Workshops. IEEE, 2017.

[16] Patgiri R, Nayak S, Borgohain S K. scaleBF: A High Scalable Membership Filter using 3D Bloom Filter[J]. International Journal of Advanced Computer Science and Applications, 2019, 9(12).

[17] Lai Y L, Goi B M, Chai T Y. Alignment-free indexing-first-one hashing with bloom filter integration[C]// 2017 IEEE International Conference on Intelligence and Security Informatics (ISI). IEEE, 2017.

[18] Yang T, Yin B, Li H, et al. Rectangular Hash Table: Bloom Filter and Bitmap Assisted Hash Table with High Speed[C]// 2017 IEEE International Conference on Big Data. IEEE, 2017.

[19] Pandeeswari P, Murugeswari S. A partition based bloom filter for fastest data search[C]// International Conference on Advanced Communication Control & Computing Technologies. IEEE, 2017.

[20] Mun J H, Lee J, Lim H. A new Bloom filter structure for identifying true positiveness of a Bloom filter[C]// 2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR). IEEE, 2017.

[21] Basat R B, Einziger G, Friedman R, et al. Poster abstract: A sliding counting bloom filter[C]// Computer Communications Workshops. IEEE, 2017.

[22] Patgiri R, Nayak S, Borgohain S K. rDBF: A r-Dimensional Bloom Filter for massive scale membership query[J]. Journal of Network and Computer Applications, 2019.

[23] Lee, Jungwon & Lim, Hyesook. (2019). A circled Bloom filter for the membership identification of multiple sets. 1-3. 10.23919/ELINFOCOM.2019.8706480.

[24] Varshney T, Verma K. Rectifying flow of duplicacy using Bloom-filter[C]// 2017 International Conference on Computer, Communications and Electronics (Comptelix). IEEE, 2017.

[25] Paulo Sérgio Almeida, Baquero C, Pregui?A N, et al. Scalable Bloom Filters[J]. Information Processing Letters, 2007, 101(6):255-261.

[26] Zengin S, Ece Güran Schmidt. A Fast and Accurate Hardware String Matching Module with Bloom Filters[M]. IEEE Press, 2017.
[27] Yamaguchi F, Nishi H. Hardware-based hash functions for network applications[C]// 2013 19th IEEE International Conference on Networks (ICON). IEEE, 2013.

[28] Einziger G, Friedman R. TinySet--An Access Efficient Self Adjusting Bloom Filter Construction[C]// IEEE, 2017:1-13.

[29] Christen P, Ranbaduge T, Vatsalan D, et al. Precise and Fast Cryptanalysis for Bloom Filter Based Privacy-Preserving Record Linkage[J]. IEEE Transactions on Knowledge and Data Engineering, 2018:1-1.

[30] Guo D, Wu J, Chen H, et al. The Dynamic Bloom Filters[J]. IEEE Transactions on Knowledge and Data Engineering, 2010, 22(1):120-133.

[31] Fan B, Andersen D G, Kaminsky M, et al. Cuckoo Filter: Practically Better Than Bloom[C]// Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies. ACM, 2014.

[32] Rottenstreich O, Keslassy I. The Bloom Paradox: When Not to Use a Bloom Filter[J]. IEEE/ACM Transactions on Networking, 2015, 23(3):703-716.

[33] Fan L, Cao P, Almeida J, et al. Summary cache: a scalable wide-area Web cache sharing protocol[J]. IEEE/ACM Transactions on Networking, 2000, 8(3):281-293.

[34] YANG Changsong, TAO Xiaoling, ZHAO Feng, WANG Yong. Secure Data Transfer and Deletion from Counting Bloom Filter in Cloud Computing[J]. Chinese Journal of Electronics, 2020, 29(02):273-280.

[35] Lim H, Lee J, Yim C. Complement Bloom Filter for Identifying True Positiveness of a Bloom Filter[J]. IEEE Communications Letters, 2015, 19(11):1905-1908.

[36] Chang, Du-Chung & Chen, Chien & Thanavel, Mahadevan. (2017). Dynamic reordering bloom filter. 288-291. 10.1109/APNOMS.2017.8094131.

[37] Patgiri, Ripon. (2019). HFil: A High Accuracy Bloom Filter. 2169-2174. 10.1109/HPCC/SmartCity/DSS.2019.00300. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].