IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 22 December 2023, accepted 7 March 2024, date of publication 13 March 2024, date of current version 21 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3377222

RESEARCH ARTICLE

Decoding Errors in Difference-Invertible Bloom Filters: Analysis and Resolution

EUNJI CHOI[1], (Student Member, IEEE), JUNGWON LEE[2], (Member, IEEE), CHANGHOON YIM[3], (Senior Member, IEEE), AND HYESOOK LIM[1], (Senior Member, IEEE)

[1]Department of Electronic and Electrical Engineering, Ewha Womans University, Seoul 03760, South Korea
[2]Department of Convergence and Open Sharing System, Seoul National University, Seoul 08826, South Korea
[3]Department of Computer Science and Engineering, Konkuk University, Seoul 05029, South Korea

Corresponding authors: Jungwon Lee (jungwon0736@snu.ac.kr) and Hyesook Lim (hlim@ewha.ac.kr)

The work of Jungwon Lee was supported by the National Research Foundation of Korea funded by the Ministry of Education under Grant NRF-2021R1I1A1A01042218. The work of Changhoon Yim was supported by the National Research Foundation of Korea under Grant NRF-2023R1A2C1004868. The work of Hyesook Lim was supported by the National Research Foundation of Korea under Grant NRF-2023R1A2C1002611.

ABSTRACT An invertible Bloom filter (IBF) is a useful data structure for various network applications because the difference IBF (d-IBF) of two IBFs programmed by two separate sets effectively identifies distinct elements unique to each set. d-IBF eliminates common elements, and unique elements are listed through a decoding process that utilizes pure cells, each of which stores a single element in a cell. However, the definition of pure cells used for decoding an IBF is insufficient to decode a d-IBF. Composite cells in a d-IBF can also satisfy the pure cell conditions defined for an IBF, and decoding composite cells adversely affects d-IBF performance. This study mathematically analyzes the probability of decoding errors in a d-IBF and proposes a new decoding method to resolve these errors. Experimental results confirm that the proposed decoding method successfully detects and resolves the decoding errors. This enables accurate identification of the difference between the two sets without generating any incorrect elements, even with a small IBF of $m = 2d$ regardless of set sizes, where m is the number of cells in the IBF and d is the size of the difference.

INDEX TERMS Bloom filter, invertible bloom filter, IBF, difference-IBF, set reconciliation.

I. INTRODUCTION

Bloom filters (BFs) [1] and their variants [2], [3], [4], [5] are widely used in network applications [6], [7], such as set reconciliation [8], [9], [10], cloud computing and storage [11], [12], [13], [14], named data networking [15], [16], [17], [18], [19], [20], network security [21], [22], [23], [24], P2P networks [25], IP address lookup [26], [27], [28], [29], sensor networks [30], [31], and network monitoring and measurement [32], [33]. Although a standard BF is typically used to provide a membership query, its variants have additional functionalities. For example, counting BF (CBF) [4], [34] enables element deletion by providing a counter per cell. Ternary BF (TBF) [9], [35] avoids *false negatives* caused by the deletion of overflow cells in a CBF and reduces the *false positive* rate.

One of the most interesting variants of a BF is the invertible-BF (IBF) [2], [8], [9], [10], [19]. A useful characteristic of an IBF is its ability to provide a difference operation for two IBFs and decode the remaining elements. In a difference-IBF (d-IBF) of two IBFs, each of which is programmed using a set of elements, common elements are removed and distinct elements included in each set are retained. By decoding the remaining elements, distinct elements that were exclusively included in each set were identified. Hence, the union of two sets can be effectively obtained by decoding a d-IBF; this process is called set reconciliation [8], [9], [10], [36]. Set reconciliation has been used in various applications that require distributed databases to contain the same set of elements.

The associate editor coordinating the review of this manuscript and approving it for publication was Abderrezak Rachedi.

However, the decoding method shown in [2] and [10] has the following problems:

- The definition of pure cells for IBF decoding is necessary but not sufficient for d-IBF decoding.
- If the same procedure used to decode an IBF is applied to decode a d-IBF, some of the distinct elements will not be decoded, and incorrect elements can be erroneously decoded.

To reduce the possibility of decoding errors, a large number of cells with a large number of bits to each cell should be allocated to an IBF despite the small difference between the two sets [19]. Even with a large number of IBF cells, decoding errors cannot be completely avoided if the conventional decoding method used for IBF decoding is applied to d-IBF decoding. This paper proposes a new and efficient method for decoding a d-IBF. The contributions of this study can be summarized as follows.

- The probability of decoding errors is mathematically analyzed when the conventional decoding method used for IBF decoding [2], [10] was applied for d-IBF decoding.
- An additional pure cell condition is proposed that should be considered for d-IBF decoding to avoid errors.
- However, because the additional pure cell condition is not clearly identified from the cell values, the decoding errors that occurred during d-IBF decoding are classified.
- Solutions to these errors are provided by proposing a new method to decode d-IBF.
- The experimental results demonstrate that the proposed decoding method can accurately identify every element included in a d-IBF without generating incorrect elements. Moreover, the proposed method achieves this using a small number of IBF cells, which is proportional to the total difference between the two sets, regardless of the set sizes.
- Source codes programmed with Verilog (used to enable a performance comparison between the proposed method and the conventional method) were made public for use by other researchers.

The remainder of this paper is organized as follows. Section II describes a BF, an IBF, and a d-IBF as related work. Section III provides a theoretical analysis of d-IBF decoding errors. Section IV classifies the errors in d-IBF decoding and describes the proposed decoding method to resolve the errors. Section V presents the results of performance evaluation. Finally, Section VI concludes the study.

II. RELATED WORKS

A. BLOOM FILTER

A Bloom filter [1] is a probabilistic data structure that determines whether an element belongs to a set. An array of m bits, $B[m]$, is used to store set $S = \{x_1, x_2, \cdots, x_n\}$ of n elements. All initial values of the array (before inserting any elements) were zero. Using k hash functions, $h_1, h_2, \cdots, h_k$, each element of S maps to the k cells of the BF. The insertion of element x is done to set the value of hashed cells to 1, i.e., $B[h_i(x)] = 1$ for $i = 1, 2, \cdots, k$.

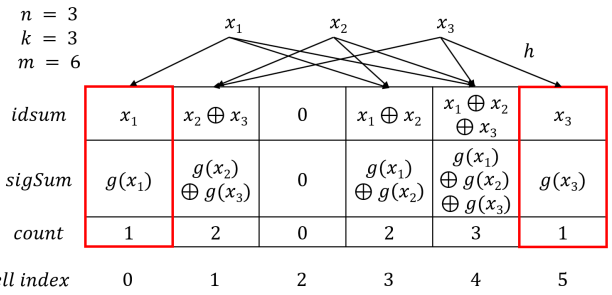


FIGURE 1. Invertible Bloom filter (IBF).

When querying an input y, if $B[h_i(y)] = 1$ for $i = 1, 2, \cdots, k$, then input y is a member of the set, which represents a *positive* result. However, if any $B[h_i(y)] = 0$, input y is not a member of the set, which represents a *negative* result. A *negative* result of the BF result is always valid. However, because of hash collisions, BFs can produce *false positives* that return *positive* results when querying elements that have not been inserted. The *false positive* rate f is related to the size of the BF as follows [5].

$$f = (1 - (1 - \frac{1}{m})^{kn})^k \tag{1}$$

B. INVERTIBLE BLOOM FILTER

An invertible Bloom filter [2], [8], [9], [10], [19] provides a useful operation to recover inserted elements in addition to insertion, deletion, and query operations. The process of recovering inserted elements is called *decoding*. Fig. 1 shows an example of IBF with six cells programmed by a set of three elements, x_1, x_2, and x_3. Each IBF cell has three fields: *idSum*, *sigSum*, and *count*. *idSum* stores the sum of inserted elements, *sigSum* stores the sum of the signatures of inserted elements, and *count* stores the number of elements stored in a cell. In this context, the sum used for element insertion or deletion was obtained using exclusive-OR (XOR) operations instead of additions. The XOR operation preserves the number of bits after summation. Algorithm 1 describes the insertion and deletion operations for an element; to insert (delete) an element into (from) a cell, the element is XORed with the *idSum* value of the cell, and the signature of the element is XORed with the *sigSum* value of the cell. The *count* value is incremented by one for each insertion and decremented by one for each deletion. In this algorithm, $g(x)$ is a function for obtaining *sigSum* of element x.

The decoding process for recovering the inserted elements depends on the pure cells. A pure cell, which is a cell with a single stored element, satisfies two conditions: $count = 1$ and $g(idSum) = sigSum$. In Fig. 1, the cells marked with red boxes represent the pure cells. Algorithm 2 shows the decoding process for IBF. *idSum* of a pure cell represents the inserted element. By examining every cell in an IBF, *pure_cell_list*, which is composed of the indices of pure cells, is obtained. The first step in the decoding process is to add the *idSum* of

Algorithm 1 Insertion and Deletion of an Element

```
Function IBF_insert (x)
    for i = 0 to k − 1 do
        IBF[h_i(x)].idSum = IBF[h_i(x)].idSum ⊕ x
        IBF[h_i(x)].sigSum = IBF[h_i(x)].sigSum ⊕
          g(x)
        IBF[h_i(x)].count = IBF[h_i(x)].count + 1
    end
end
Function IBF_delete (x)
    for i = 0 to k − 1 do
        IBF[h_i(x)].idSum = IBF[h_i(x)].idSum ⊕ x
        IBF[h_i(x)].sigSum = IBF[h_i(x)].sigSum ⊕
          g(x)
        IBF[h_i(x)].count = IBF[h_i(x)].count − 1
    end
end
```

Algorithm 2 Decoding Process in an IBF

```
Function IBF_decoding (pure_cell_list)
    while !pure_cell_list.empty do
        idSum_p ← IBF[pure_cell_list.front].idSum;
        decode_list.end ← idSum_p;
        IBF_delete (idSum_p);
        pure_cell_list.end ←
          IBF[new_pure_cell].index;
    end
end
```

a pure cell to the decode list and then to delete the element in the pure cell from the IBF. If an element in a pure cell is deleted, new pure cells can be generated if the other accessed cells had two stored elements. Newly generated pure cells are added to the *pure_cell_list*, and deletions are then repeated for the next pure cell until the *pure_cell_list* becomes empty.

Figure 2 shows an example of the IBF decoding process of the IBF in Fig. 1. In Fig. 2(a), cells in indexes 0 and 5 satisfy pure cell conditions; hence, indexes 0 and 5 are added to the *pure_cell_list*. x_1 is added to *decode_list* in Fig. 2(b). For the first index of *pure_cell_list*, Fig. 2(c) shows the deletion of x_1. From the cells in indexes 0, 3, and 4 of the IBF, x_1 was deleted by applying Algorithm 1. The cell with index 3 became a new pure cell by deleting x_1, and hence, index 3 was added to *pure_cell_list*. The same process was repeated for the first index in the *pure_cell_list*, which was 5. The decoding process is terminated when the *pure_cell_list* is empty. The decoding process is considered successful if every cell of the IBF became zero. Because the decoding process of an IBF relies solely on pure cells, decoding fails if there are no pure cells before all the elements have been decoded.

The appropriate number of hash functions used to build an IBF can be experimentally obtained [10]. When a small number of hash functions is used, there is a higher possibility

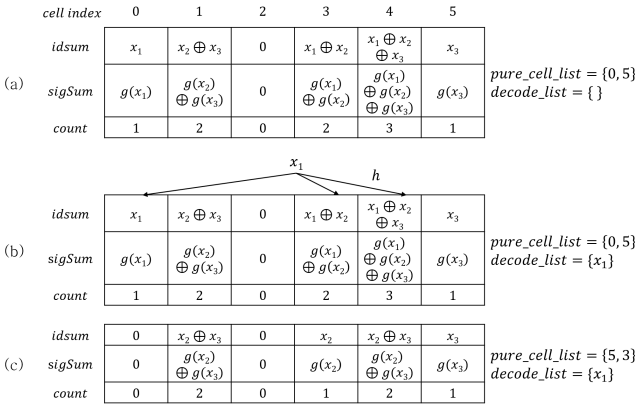


FIGURE 2. IBF decoding for x_1 at index 0.

of generating a larger number of pure cells initially. However, a few new pure cells may have been created during the decoding process. However, when a large number of hash functions are used, the probability of decoding failure increases because of the inadequate number of initial pure cells. The optimal number of hash functions for a d-IBF is typically three or four according to [10].

C. DIFFERENCE-INVERTIBLE BLOOM FILTER

A difference-IBF (d-IBF) makes an IBF more useful in many applications. In particular, for set reconciliation, d-IBF is effectively used to obtain the union of two sets [9], [10], [19], [36]. A d-IBF is obtained by subtraction of each cell, as shown in Algorithm 3. If IBF_1 and IBF_2 programmed by two sets S_1 and S_2, respectively, have the same size and use the same hash functions h_i for $i = 1, 2, \cdots, k$, common elements are programmed into the same cells in both IBFs. Hence, every common element is eliminated by subtraction, and only the distinct elements (which are unique elements included in either of the sets) are retained in the d-IBF.

Algorithm 3 Obtaining a d-IBF

```
Function IBF_subtract (IBF_1, IBF_2)
    for i = 0 to m − 1 do
        d-IBF[i].idSum = IBF_1[i].idSum ⊕
          IBF_2[i].idSum
        d-IBF[i].sigSum = IBF_1[i].sigSum
          ⊕IBF_2[i].sigSum
        d-IBF[i].count = IBF_1[i].count − IBF_2[i].count
    end
end
```

Using a d-IBF is particularly effective when the number of common elements in the two sets is considerably larger than the number of distinct elements. Because common elements are removed in a d-IBF, the number of cells in an IBF, m, can be set in proportion to d regardless of the set size, where d is the sum of the numbers of distinct elements included in each set. Therefore, $m = \alpha d$, and $\alpha \geq 1$.

TABLE 1. Terms and notations.

Term	Description
m	number of cells in an IBF or a d-IBF
h_i	hash functions to map an element to IBF or d-IBF cells
g	hash function to obtain *sigSum*
S_1	Set of elements in IBF_1
S_2	Set of elements in IBF_2
D_1	Set of distinct elements in S_1
D_2	Set of distinct elements in S_2
$\hat{D}_1$	Set of candidate elements for D_1
$\hat{D}_2$	Set of candidate elements for D_2
d	sum of the numbers of distinct elements of S_1 and S_2, $d = \lvert D_1\rvert + \lvert D_2\rvert$

Table 1 summarizes the terms and notations used in this study. Algorithm 4 shows the deletion operation of an element from a d-IBF. Because the count value could be 1 or -1, upon deleting an element from a cell in a d-IBF, the count value of the pure cell should be subtracted from the count value of the cell, unlike Algorithm 1.

Algorithm 5 outlines the conventional decoding process for d-IBF [10]. This process is similar to the IBF decoding process, except for the handling of count values. For each pure cell, the *idSum* value of the pure cell is added to the decoded list of D_1 or D_2 depending on the count value of the pure cell, and then the deletion operation described in Algorithm 4 is performed. Assuming that hash indexes for an element are distinct, if the count value of a cell is 1, the *idSum* of the pure cell is an element belonging to set D_1; if the count value is -1, it is an element belonging to set D_2.

When *pure_cell_list* becomes empty, the d-IBF needs to be examined whether it is empty. If all the cell values of the d-IBF are zero, this implies that every distinct element included in either of the sets is successfully decoded. Otherwise, the decoding process fails.

Algorithm 4 Element Deletion From a d-IBF

```
Function d-IBF_delete (x, c)
    for i = 0 to k − 1 do
        d-IBF[h_i(x)].idSum = d-IBF[h_i(x)].idSum ⊕ x
        d-IBF[h_i(x)].sigSum = d-IBF[h_i(x)].sigSum ⊕
          g(x)
        d-IBF[h_i(x)].count = d-IBF[h_i(x)].count − c
    end
end
```

III. THEORETICAL ANALYSIS OF D-IBF DECODING ERRORS

The conventional decoding method [10] shown in Algorithm 5, which relies on cells satisfying two pure cell conditions, can cause decoding errors for a d-IBF. For example, if a cell in IBF_1 has two unique elements and the corresponding cell in IBF_2 has one unique element, the count value of the corresponding d-IBF cell is one, but the *idSum* of the cell is the composite value of three distinct elements. If $g(idSum)$ of the cell is not equal to *sigSum* of the cell, the cell is identified as a non-pure cell. However, if they

Algorithm 5 Conventional Decoding in a d-IBF

```
Function d-IBF_decoding (d-IBF)
    while !pure_cell_list.empty do
        idSum_p ← d-IBF[pure_cell_list.front].idSum;
        count_p ← d-IBF[pure_cell_list.front].count;
        if count_p == 1 then
            D_1.end ← idSum_p;
        end
        else if count_p == -1 then
            D_2.end ← idSum_p;
        end
        d-IBF_delete (idSum_p, count_p);
        pure_cell_list.end ←
          d-IBF[new_pure_cell].index;
    end
    if d-IBF.empty then
        return Success;
    end
    else
        return Fail;
    end
end
```

are the same by chance, the decoding process using this cell corrupts the d-IBF and causes errors. In [10], a large number of bits were allocated to the *sigSum* field to reduce this type of error. However, because every cell has a *sigSum* field, allocating a large number of bits to *sigSum* requires a large memory to implement an IBF. This is more beneficial when an IBF has a space-efficient data structure and fits into a small amount of memory. Moreover, as will be shown through performance evaluation, a large number of bits allocated to the *sigSum* field does not effectively reduce this type of error. Hence, our motivation is to resolve decoding errors without requiring a large number of bits for the *sigSum* field. This section analyzes the probability of decoding errors when the composite cells are used for d-IBF decoding.

Based on the notation in Table 1, S_1 is composed of disjoint sets D_1 and $S_1 \cap S_2$, i.e., $S_1 = D_1 \cup (S_1 \cap S_2)$ and $D_1 \cap (S_1 \cap S_2) = \emptyset$. Similarly, S_2 consists of disjoint sets D_2 and $S_1 \cap S_2$, i.e., $S_2 = D_2 \cup (S_1 \cap S_2)$ and $D_2 \cap (S_1 \cap S_2) = \emptyset$. Let n_1, n_2, and n_3 represent the numbers of elements in D_1, D_2, and $S_1 \cap S_2$, respectively. Let N be the total number of elements in $S_1 \cup S_2$. Then $N = n_1 + n_2 + n_3$. The sets of distinct elements can be represented as $D_1 = \{x_1, x_2, \cdots, x_{n_1}\}$ and $D_2 = \{y_1, y_2, \cdots, y_{n_2}\}$. Let b be the number of bits used to represent element x in $S_1 \cup S_2$. Then, x can be stored as a number in $0, 1, \cdots, 2^b - 1$. Let $H = \{h_1, h_2, \cdots, h_k\}$ be the set of hash functions, where k is the number of hash functions. Let $A = \{0, 1, 2, \cdots, m-1\}$ be the set of indices (addresses) in a d-IBF, where m is the number of cells in the d-IBF.

Let d-IBF[i].*idSum* be the *idSum* value at index i of the d-IBF. Let d-IBF[i].*sigSum* be the *sigSum* for the index i of the d-IBF. Let $g(x)$ be the signature of x.

A cell with index i is defined as a genuine pure cell in a d-IBF if it satisfies the following conditions:

1) $|$ d-IBF[i].count $| = 1$
2) g(d-IBF[i].*idSum*) = d-IBF[i].*sigSum*
3) d-IBF[i].*idSum* $\in D_1 \cup D_2$

Compared with the pure cell requirement of an IBF, an additional requirement (Condition 3) was added with a d-IBF. Because Condition 3 cannot be directly identified from the cell values, if a cell satisfies Conditions 1 and 2, it is referred to as a candidate pure cell.

We then derived the probability that a cell in a d-IBF is a candidate pure cell. Let $O_{1,j}$ be the event that a cell in IBF_1 is set j times using k hash functions with n_1 elements in D_1. Similarly, let $O_{2,j}$ be the event that a cell in IBF_2 is set j times using k hash functions with n_2 elements in D_2. Events $O_{1,i}$ and $O_{2,j}$ for any i, j are independent. Let C_j be the event in which the count of a cell in d-IBF is j. Then, the events for Condition 1, C_1 and C_{-1}, can be formulated as

$$C_1 = (O_{1,1} \cap O_{2,0}) \cup (O_{1,2} \cap O_{2,1}) \cup \cdots \cup (O_{1,n_1} \cap O_{2,n_1-1}) \tag{2}$$

$$C_{-1} = (O_{1,0} \cap O_{2,1}) \cup (O_{1,1} \cap O_{2,2}) \cup \cdots \cup (O_{1,n_2-1} \cap O_{2,n_2}) \tag{3}$$

Events $O_{1,j}$ and $O_{2,j-1}$ are independent, and events $O_{1,j-1}$ and $O_{2,j}$ are independent too. Hence

$$P(O_{1,j} \cap O_{2,j-1}) = P(O_{1,j})P(O_{2,j-1}) \tag{4}$$

$$P(O_{1,j-1} \cap O_{2,j}) = P(O_{1,j-1})P(O_{2,j}) \tag{5}$$

where $P(O_i)$ represent the probability of event O_i.

$P(O_{1,j})$ and $P(O_{2,j})$ can be calculated as

$$P(O_{1,j}) = \binom{kn_1}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{kn_1-j} \tag{6}$$

$$P(O_{2,j}) = \binom{kn_2}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{kn_2-j} \tag{7}$$

From (4), (6), and (7),

$$\begin{aligned} P(O_{1,1} \cap O_{2,0}) &= P(O_{1,1})P(O_{2,0}) \\ &= \binom{kn_1}{1} \left(\frac{1}{m}\right)^1 \left(1 - \frac{1}{m}\right)^{kn_1-1} \\ &\quad \cdot \binom{kn_2}{0} \left(\frac{1}{m}\right)^0 \left(1 - \frac{1}{m}\right)^{kn_2} \\ &= kn_1 \left(\frac{1}{m}\right) \left(1 - \frac{1}{m}\right)^{kn_1+kn_2-1} \end{aligned} \tag{8}$$

$$\begin{aligned} P(O_{1,2} \cap O_{2,1}) &= P(O_{1,2})P(O_{2,1}) \\ &= \binom{kn_1}{2} \left(\frac{1}{m}\right)^2 \left(1 - \frac{1}{m}\right)^{kn_1-2} \\ &\quad \cdot \binom{kn_2}{1} \left(\frac{1}{m}\right)^1 \left(1 - \frac{1}{m}\right)^{kn_2-1} \\ &= \frac{k^2 n_1 n_2 (kn_1 - 1)}{2} \left(\frac{1}{m}\right)^3 \left(1 - \frac{1}{m}\right)^{kn_1+kn_2-3} \end{aligned} \tag{9}$$

Similarly, from Eqs.(5), (6), and (7),

$$P(O_{1,0} \cap O_{2,1}) = kn_2 \left(\frac{1}{m}\right) \left(1 - \frac{1}{m}\right)^{kn_1+kn_2-1} \tag{10}$$

$$\begin{aligned} P(O_{1,1} \cap O_{2,2}) &= \frac{k^2 n_1 n_2 (kn_2 - 1)}{2} \left(\frac{1}{m}\right)^3 \left(1 - \frac{1}{m}\right)^{kn_1+kn_2-3} \end{aligned} \tag{11}$$

Events $(O_{1,j} \cap O_{2,j-1})$ and $(O_{1,j-1} \cap O_{2,j})$ are disjoint. Using (2) and (4),

$$\begin{aligned} P(C_1) &= P(O_{1,1} \cap O_{2,0}) + P(O_{1,2} \cap O_{2,1}) \\ &\quad + \cdots + P(O_{1,n_1} \cap O_{2,n_1-1}) \\ &= \sum_{j=1}^{n_1} P(O_{1,j} \cap O_{2,j-1}) \\ &= \sum_{j=1}^{n_1} P(O_{1,j}) P(O_{2,j-1}) \end{aligned} \tag{12}$$

From (6), (7), and (12),

$$\begin{aligned} P(C_1) &= \sum_{j=1}^{n_1} \binom{kn_1}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{kn_1-j} \\ &\quad \cdot \binom{kn_2}{j-1} \left(\frac{1}{m}\right)^{j-1} \left(1 - \frac{1}{m}\right)^{kn_2-j+1} \\ &= \sum_{j=1}^{n_1} \binom{kn_1}{j} \binom{kn_2}{j-1} \left(\frac{1}{m}\right)^{2j-1} \left(1 - \frac{1}{m}\right)^{kn_1+kn_2-2j+1} \end{aligned} \tag{13}$$

From (3) and (5),

$$\begin{aligned} P(C_{-1}) &= P(O_{1,0} \cap O_{2,1}) + P(O_{1,1} \cap O_{2,2}) \\ &\quad + \cdots + P(O_{1,n_2-1} \cap O_{2,n_2}) \\ &= \sum_{j=1}^{n_2} P(O_{1,j-1} \cap O_{2,j}) \\ &= \sum_{j=1}^{n_2} P(O_{1,j-1}) P(O_{2,j}) \end{aligned} \tag{14}$$

From (6), (7), and (14),

$$\begin{aligned} P(C_{-1}) &= \sum_{j=1}^{n_2} \binom{kn_1}{j-1} \left(\frac{1}{m}\right)^{j-1} \left(1 - \frac{1}{m}\right)^{kn_1-j+1} \\ &\quad \cdot \binom{kn_2}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{kn_2-j} \\ &= \sum_{j=1}^{n_2} \binom{kn_1}{j-1} \binom{kn_2}{j} \left(\frac{1}{m}\right)^{2j-1} \left(1 - \frac{1}{m}\right)^{kn_1+kn_2-2j+1} \end{aligned} \tag{15}$$

Let $\tilde{C}_1$ be the event that a cell in a d-IBF satisfies Condition 1 of the pure cell, and p_1 be the probability of $\tilde{C}_1$. We obtain

$$p_1 = P(\tilde{C}_1) = P(C_1 \cup C_{-1}) = P(C_1) + P(C_{-1}) \tag{16}$$

Probability $P(\tilde{C}_1)$ can be calculated using (13), (15), and (16), which is not represented here because of space limitations. The probability that no cell in the d-IBF satisfies Condition 1 can be obtained using $(1-p_1)^m$. This is the probability that d-IBF decoding ceased.

Let O_p be the event that *idSum* of a cell in a d-IBF is set by only one element of $D_1 \cup D_2$, which satisfies Condition 3 of a pure cell. If a cell has an event O_p, then it is a genuine pure cell, that is, it satisfies all the conditions of a pure cell. Event O_p is a subset of $C_1 \cup C_{-1}$ in (2) and (3), yielding:

$$O_p = (O_{1,1} \cap O_{2,0}) \cup (O_{1,0} \cap O_{2,1}) \tag{17}$$

From (8), (10), and (17),

$$\begin{aligned} P(O_p) &= P(O_{1,1} \cap O_{2,0}) + P(O_{1,0} \cap O_{2,1}) \\ &= kn_1 \left(\frac{1}{m}\right)\left(1-\frac{1}{m}\right)^{kn_1+kn_2-1} \\ &\quad + kn_2 \left(\frac{1}{m}\right)\left(1-\frac{1}{m}\right)^{kn_1+kn_2-1} \\ &= k(n_1+n_2)\left(\frac{1}{m}\right)\left(1-\frac{1}{m}\right)^{kn_1+kn_2-1} \end{aligned} \tag{18}$$

Let $C_{1,e}$ be the event that the absolute value of the count is one and $g(idSum)$ is equal to *sigSum* in a cell of a d-IBF. In other words, a cell with event $C_{1,e}$ would satisfy Conditions 1 and 2. Let $C_{1,ne}$ be the event that the absolute value of the count is 1 and the $g(idSum)$ is not equal to *sigSum* in a cell of a d-IBF. A cell with event $C_{1,e}$ satisfies Condition 2, whereas a cell with event $C_{1,ne}$ does not satisfy Condition 2. Events $C_{1,e}$ and $C_{1,ne}$ are two disjoint (partition) events of $\tilde{C}_1$, i.e., $\tilde{C}_1 = C_{1,e} \cup C_{1,ne}$ and $C_{1,e} \cap C_{1,ne} = \emptyset$. Event $\tilde{C}_1$ can also be decomposed into two disjoint events (O_p and $\tilde{C}_1 - O_p$), i.e., $\tilde{C}_1 = O_p \cup (\tilde{C}_1 - O_p) = O_p \cup (\tilde{C}_1 \cap O_p^c)$, where O_p^c represent the complement event of O_p.

The cell with event O_p satisfies conditions of $C_{1,e}$. There might exist cells with events $C_{1,e}$ which do not satisfy the event condition of O_p. This case can occur when the absolute value of the count is 1 (Condition 1), the *idSum* of a cell is the result of XOR operations with multiple elements, and $g(idSum)$ is equal to the *sigSum* of the cell (Condition 2). Hence $O_p \subset C_{1,e} \subset \tilde{C}_1$. When *idSum* of a cell is the result of XOR operations with multiple elements, this cell is a composite cell, which is a non-pure cell.

We derive the probability of Condition 2 given Condition 1, which can be formulated as

$$P(C_{1,e}|\tilde{C}_1) = P(C_{1,e}|O_p)P(O_p|\tilde{C}_1) + P(C_{1,e}|O_p^c)P(O_p^c|\tilde{C}_1) \tag{19}$$

Since $O_p \subset C_{1,e}$, $O_p \cap C_{1,e} = O_p$. Hence,

$$P(C_{1,e}|O_p) = \frac{P(C_{1,e} \cap O_p)}{P(O_p)} = \frac{P(O_p)}{P(O_p)} = 1 \tag{20}$$

Since $O_p \subset C_1 \cup C_{-1}$ and $O_p \subset \tilde{C}_1$,

$$P(O_p|\tilde{C}_1) = \frac{P(O_p \cap \tilde{C}_1)}{P(\tilde{C}_1)} = \frac{P(O_p)}{P(\tilde{C}_1)} \tag{21}$$

$$P(O_p^c|\tilde{C}_1) = \frac{P(O_p^c \cap \tilde{C}_1)}{P(\tilde{C}_1)} \tag{22}$$

From (2), (3), (16), and (17), we obtain the following:

$$\begin{aligned} O_p^c \cap \tilde{C}_1 = &(O_{1,2} \cap O_{2,1}) \cup \cdots \cup (O_{1,n_1} \cap O_{2,n_1-1}) \\ &\cup (O_{1,1} \cap O_{2,2}) \cup \cdots \cup (O_{1,n_2-1} \cap O_{2,n_2}) \end{aligned} \tag{23}$$

From (6), (7), and (23),

$$P(O_p^c \cap \tilde{C}_1) = \sum_{j=2}^{n_1} P(O_{1,j})P(O_{2,j-1}) + \sum_{j=2}^{n_2} P(O_{1,j-1})P(O_{2,j}) \tag{24}$$

Let s be the number of bits allocated to the signature. Because there are 2^s possible cases with a uniform distribution of the *sigSum* field in a cell, we can write $P(C_{1,e}|O_p^c) = 1/2^s$. From (19), (20), (21), (22), and (24), we formulate

$$\begin{aligned} P(C_{1,e}|\tilde{C}_1) = &\frac{P(O_p)}{P(\tilde{C}_1)} + \frac{1}{2^s P(\tilde{C}_1)}\Big(\sum_{j=2}^{n_1} P(O_{1,j})P(O_{2,j-1}) \\ &+ \sum_{j=2}^{n_2} P(O_{1,j-1})P(O_{2,j})\Big) \end{aligned} \tag{25}$$

Since $C_{1,e}$ and $C_{1,ne}$ are partition events of $\tilde{C}_1$, $P(C_{1,ne}|\tilde{C}_1)$ can be obtained as

$$P(C_{1,ne}|\tilde{C}_1) = 1 - P(C_{1,e}|\tilde{C}_1) \tag{26}$$

In (25), $P(O_p)$ can be calculated using (18), and $P(\tilde{C}_1)$ can be calculated using (13), (15), and (16).

We consider the probability of Condition 2 of a composite cell given Condition 1, which can be formulated as

$$P(C_{1,e} - O_p|\tilde{C}_1) = P(C_{1,e}|\tilde{C}_1) - P(O_p|\tilde{C}_1) \tag{27}$$

Let p_{de} be the probability of the d-IBF decoding error using composite cells. As there are k trials by k hash functions for one element in d-IBF, p_{de} can be calculated as follows:

$$p_{de} = k \cdot P(C_{1,e} - O_p|\tilde{C}_1) \tag{28}$$

From (21), (25), (27), and (28), the probability of the d-IBF decoding error using composite cells for decoding p_{de}, is obtained as follows:

$$\begin{aligned} p_{de} = &\frac{k}{2^s p_1}\Big(\sum_{j=2}^{n_1} P(O_{1,j})P(O_{2,j-1}) \\ &+ \sum_{j=2}^{n_2} P(O_{1,j-1})P(O_{2,j})\Big) \end{aligned} \tag{29}$$

IV. PROPOSED DECODING METHOD

To resolve d-IBF decoding errors, an additional step is required to determine whether a cell satisfies Condition 3 (the *idSum* of a cell is an element included in the union of D_1 and D_2). If the *idSum* of a candidate pure cell belongs to neither D_1 nor D_2, then the cell is not a pure cell.

This section classifies the d-IBF decoding errors described in Section III into two types and then proposes a decoding method to resolve these errors.

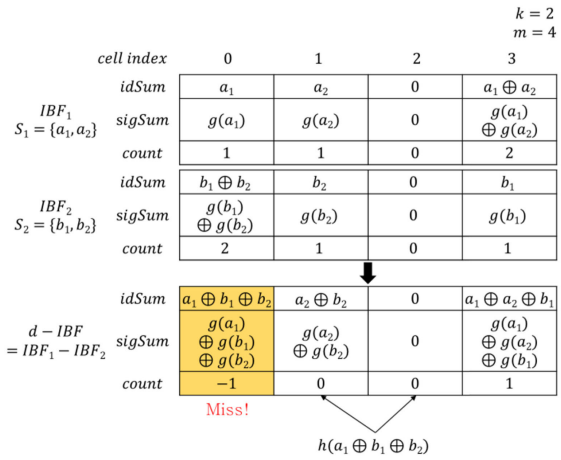


FIGURE 3. T_1: Case that a candidate pure cell is identified non-pure from hashing indexes of *idSum* since the cell index is not included in the hashing indexes of *idSum*.

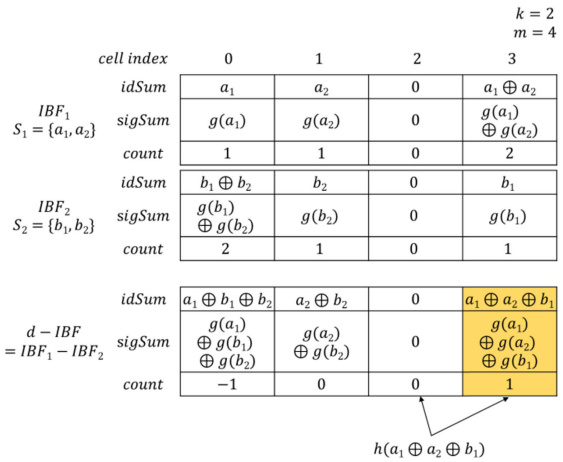


FIGURE 4. T_2: Case that a candidate pure cell is not identified non-pure from hashing indexes of *idSum* since the cell index is accidentally equal to one of hash indexes of *idSum*.

A. T_1: NON-PURE CELL, IDENTIFIED FROM HASHING INDEXES

The first type, T_1, is the case where a candidate pure cell that satisfies Conditions 1 and 2 is identified as a non-pure cell during the decoding process. For example, Fig. 3 shows the T_1 case. The cell at index 0 of the d-IBF has $|count| = 1$, and if by chance the following holds true $g(a_1 \oplus b_1 \oplus b_2) = g(a_1) \oplus g(b_1) \oplus g(b_2)$, the cell is a candidate pure cell and will be used for decoding. Assuming that the hash indexes of $a_1 \oplus b_1 \oplus b_2$ do not include index 0 as shown in the figure, the decoding process using the *idSum* of this cell will result in the insertion of $a_1 \oplus b_1 \oplus b_2$ to the second and the third cells, without deleting any element. Hence, the decoding process performed for a cell with a T_1 case will result in corrupting the d-IBF by inserting a composite element. Moreover, the insertion of a dummy element into the d-IBF results in a reduced number of pure cells and a higher probability of decoding failures.

T_1 error can be easily resolved because the hash indices of a pure cell do not include the index of the cell itself (index 0 in Fig. 3). In the case of T_1, the deletion process using this cell should be aborted to avoid erroneous decoding. It is noteworthy that the hash indexes of a candidate pure cell, $h_i(idSum)$, serve as signatures in T_1 cases similar to the *sigSum* field. Algorithm 6 shows the decoding process for resolving T_1 errors.

Algorithm 6 T_1 Error Resolving

```
Function T1_resolve(pureCellIndex, h[])
    if pureCellIndex ∉ h[] then
        pureList_erase(pureCellIndex);
        return 1;
    end
    else
        return 0;
    end
end
```

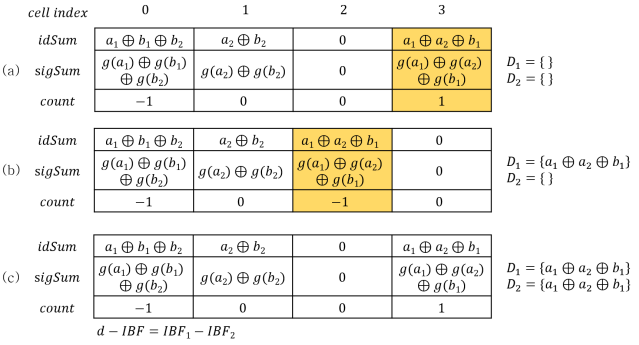


FIGURE 5. Decoding process when a T_2 error occurs.

B. T_2: NON-PURE CELL, BUT NOT IDENTIFIED BY HASHING INDEXES

The second type, T_2, is the case where a candidate pure cell is not identified as a non-pure cell because one of the hashing indices of the *idSum* of the cell is (accidentally) equal to its index. For example, Fig. 4 shows the case in which a T_2 error occurs in the cell at index 3 of d-IBF. If $g(a_1 \oplus a_2 \oplus b_1) = g(a_1) \oplus g(a_2) \oplus g(b_1)$ and $h_i(a_1 \oplus a_2 \oplus b_1) = \{2, 3\}$ by accident, it cannot be determined whether this cell is genuine. As shown through the simulation in Section V, T_2 errors occasionally occur because the size of an IBF is proportional to the number of distinct elements and not to the total number of elements in the union of the two sets.

Figure 5 shows the decoding process for the d-IBF in Fig. 4. For the cell at index 3, *idSum*, which is $a_1 \oplus a_2 \oplus b_1$, is added to D_1 (since $count = 1$) and deleted from hashed cells 2 and 3. Figure 5(b) shows the resulting d-IBF after deleting $a_1 \oplus a_2 \oplus b_1$ from the d-IBF. Note that the decoding process, performed for a non-pure cell with a T_2 case, is the same as inserting an element into the d-IBF.

The cell at index 2 in Fig. 5(b) is also not a genuine pure cell, which has an opposite count value, and the decoding

process performed for the cell results in the d-IBF in Fig. 5(c), which is the same as that in Fig. 5(a).

An important observation regarding the T_2 error is shown in Fig. 5. As shown in Fig. 5(c), the *idSum* of a composite cell with a T_2 error is added to the lists of both D_1 and D_2 after performing the decoding process for the cell twice. In the step shown in Fig. 5(b), because the *idSum* of a pure cell with $count = -1$ has already been included in the list of D_1, the cell can be regarded as a non-pure cell with a T_2 error. This is because *idSum* with $count = -1$ should be in D_2 and not in D_1. Hence, the decoding process performed for Fig. 5(a) should be recovered by deleting $a_1 \oplus a_2 \oplus b_1$ from the d-IBF and decoded lists.

Algorithm 7 shows the procedure for resolving a T_2 error. For a given candidate pure cell and its hash indices, if $count = 1$ and *idSum* is already included in D_2, this means that the candidate pure cell is not genuine (because *idSum* with $count = 1$ is an element of set S_1). Likewise, if $count = -1$ and *idSum* is included in D_1, the candidate pure cell is not a genuine pure cell (because *idSum* with $count = -1$ is an element of set S_2). For the non-pure cell, *idSum* should be deleted from the d-IBF.

Algorithm 7 returns one if a T_2 error has occurred and is resolved, and returns zero, otherwise.

Algorithm 7 T_2 Error Resolving

```
Function T2_resolve(pure_cell_list.front)
    s ← d-IBF[pure_cell_list.front].idSum;
    c ← d-IBF[pure_cell_list.front].count;
    if c = 1 && s ∈ D2 then
        pure_cell_list.remove(pure_cell_list.front);
        remove s from D2;
        d-IBF_delete(s, c);
        pure_cell_list.end ←
          d-IBF[new_pure_cell].index;
        return 1;
    end
    else if c = −1 && s ∈ D1 then
        pure_cell_list.remove(pure_cell_list.front);
        remove s from D1;
        d-IBF_delete(s, c);
        pure_cell_list.end ←
          d-IBF[new_pure_cell].index;
        return 1;
    end
    else
        return 0;
    end
end
```

Algorithm 8 shows the decoding process of the d-IBF proposed in this study. The decoding process continues only when neither a T_1 nor a T_2 error occurs. After every element in the *pure_cell_list* is decoded, if all the cells in the d-IBF have a zero value, the decoding process returns *Success*; otherwise, it returns *Fail*.

Algorithm 8 Proposed Decoding Algorithm

```
Function d-IBF_decoding(d-IBF)
    while !pure_cell_list.empty do
        type1 ← T1_resolve(pure_cell_list.front);
        type2 ← T2_resolve(pure_cell_list.front);
        if !(type1 || type2) then
            idSum_p ←
            d-IBF[pure_cell_list.front].idSum;
            count_p ←
            d-IBF[pure_cell_list.front].count;
            if count_p == 1 then
                D1.end ← idSum_p;
            end
            else if count_p == −1 then
                D2.end ← idSum_p;
            end
            d-IBF_delete( idSum_p, count_p );
            pure_cell_list.end ←
              d-IBF[new_pure_cell].index;
        end
    end
    if d-IBF.empty then
        return Success;
    end
    else
        return Fail;
    end
end
```

V. PERFORMANCE EVALUATION

This section compares the decoding performance of the proposed d-IBF decoding algorithm (Algorithm 8) with that of conventional decoding (Algorithm 5). Simulations were conducted with Verilog using ModelSim (Prime Lite Edition, software version 18.1) provided by Intel [37], and source codes were made available for public use [38]. Non-duplicated random numbers for each set were generated using a linear feedback shift register (LFSR). Three hash indexes are obtained for each element from a 32-bit cyclic redundancy check (CRC-32) generator to program and decode IBFs because multiple hash indexes of any length can be easily obtained from CRC-32 [27]. While the simulation performed in [10] has the constraint that the hash indices of an element need to be distinct, we do not have such a constraint in our simulation to cover more general cases. In other words, two or three hash indices of an element can be identical in this simulation.

Three experiments have been performed using different set sizes: ExpA, ExpB, and ExpC. Table 2 shows the number of elements included in each set and the total number of distinct elements, d. Table 2 also shows the number of bits allocated to each field. Simulations performed in [10] used 32 bits for each *idSum* and *sigSum* to reduce erroneous cases where *sigSum* of a composite cell is equal to $g(idSum)$. In our experiments, we provided two simulations for conventional

TABLE 2. Simulation setup.

	S_1	S_2	$S_1 \cap S_2$	d	Conventional decoding 1			Conventional decoding 2			Proposed decoding		
					idSum	*sigSum*	*count*	*idSum*	*sigSum*	*count*	*idSum*	*sigSum*	*count*
ExpA	10,000	10,020	9,990	40	32	32	16	14	4	16	14	4	16
ExpB	100,000	100,000	95,000	10,000	32	32	16	17	4	16	17	4	16
ExpC	1,000,000	1,000,000	999,500	1,000	32	32	16	20	4	16	20	4	16

*First 4 columns: number of elements, *Remaining columns: number of bits

TABLE 3. Decoding results.

		α			
		1	1.5	2	2.5
ExpA	Conventional decoding 1	×	×	×	×
	Conventional decoding 2	×	×	×	×
	Proposed decoding method	×	○	○	○
ExpB	Conventional decoding 1	×	×	×	×
	Conventional decoding 2	×	×	×	×
	Proposed decoding method	×	×	○	○
ExpC	Conventional decoding 1	×	×	×	×
	Conventional decoding 1	×	×	×	×
	Proposed decoding method	×	×	○	○

× : Fail (Because there is no pure cell, decoding stops before d-IBF becomes empty.)
○ : Success (Every distinct element is decoded and the resulting d-IBF is empty.)

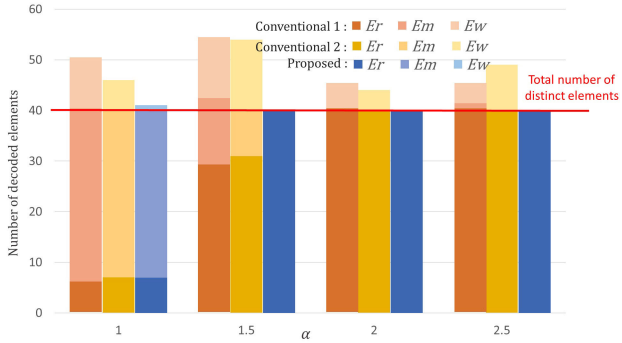


FIGURE 6. Decoding result comparison for ExpA.

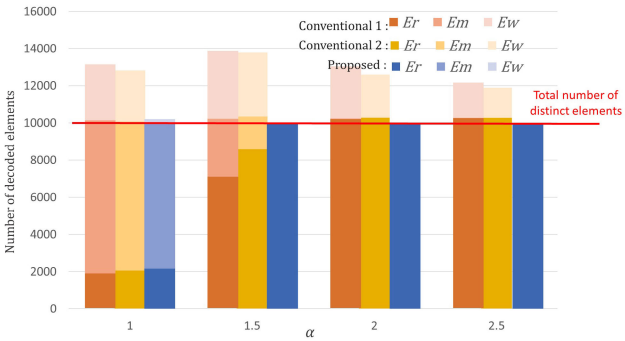


FIGURE 7. Decoding result comparison for ExpB.

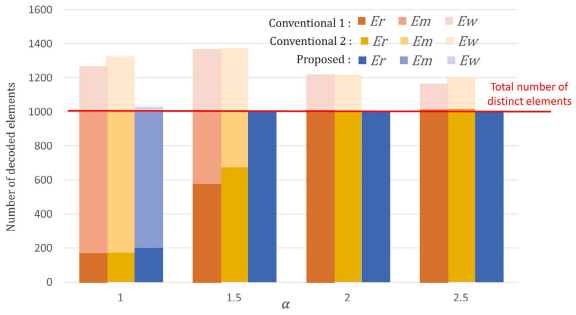


FIGURE 8. Decoding result comparison for ExpC.

decoding methods: **Conventional 1**, allocating 32 bits to each of the *idSum* and *sigSum* fields as in [10], and **Conventional 2**, allocating the minimum number of bits to *idSum* and *sigSum* fields to reduce the memory requirement. The proposed decoding method uses the same number of bits as in Conventional 2.

Note that three hash indices are used for each element, and the different combinations of hash indices of an element can be used to differentiate one element from the others. Hence, we believe that it is unnecessary to allocate a large number of bits to the *sigSum* field. Therefore, four bits were allocated to the *sigSum* field for the Conventional 2 decoding method and our proposed decoding method, as shown in Table 2. The number of bits allocated to the *count* field was 16 bits for all cases to avoid overflow. The number of cells in an IBF or d-IBF is $m = \alpha d$, where α is a size factor. For each size factor, the decoding performance of our proposed method was compared with that of the conventional methods.

Table 3 presents a comparison of the decoding results based on α. Successful decoding must satisfy two conditions: every distinct element is decoded, and after the completion of the decoding process, the d-IBF is empty, which means that every cell of the d-IBF has zero values [10]. For conventional decoding methods, all decoding processes are aborted and failed because there are no candidate pure cells before the d-IBF becomes empty for every α value. By contrast, the decoding process of the proposed method is successful for $\alpha = 1.5$, 2, and 2.5 cases in ExpA and for $\alpha = 2$ and 2.5 cases in ExpB and ExpC.

Table 4 compares the detailed numbers after decoding. E_d is the number of decoded elements, E_r denotes the number of right(correct) distinct elements among the decoded elements, E_w is the number of wrong elements among the decoded elements, and E_m denotes the number of missing distinct elements that are not decoded.

Figures 6-8 compare the decoding results graphically in terms of E_r, E_m, and E_w, related to the total number of distinct elements d. Even though the two conventional methods decode many more elements than d, the number of correct elements (E_r) is less than d, meaning that there are a number

TABLE 4. Number of decoded elements.

		α															
		1				1.5				2				2.5			
		E_d	E_r	E_w	E_m	E_d	E_r	E_w	E_m	E_d	E_r	E_w	E_m	E_d	E_r	E_w	E_m
ExpA	Conventional 1	16	6	10	34	41	29	12	13	45	40	5	0	44	40	4	1
	Conventional 2	13	7	6	33	45	31	14	9	44	40	4	0	49	40	9	0
	Proposed	8	7	1	33	40	40	0	0	40	40	0	0	40	40	0	0
ExpB	Conventional 1	4,848	1,849	2,999	8,184	10,653	7,023	3,630	3,082	12,887	10,088	2,799	36	12,037	10,146	1,891	8
	Conventional 2	4,800	2,054	2,746	8,020	12,024	8,572	3,452	1,759	12,589	10,263	2,326	19	11,888	10,277	1,611	0
	Proposed	2,361	2,171	190	7,829	10,000	9,999	1	1	10,000	10,000	0	0	10,000	10,000	0	0
ExpC	Conventional 1	436	173	263	827	929	576	353	433	1,215	1,007	208	1	1,159	1,010	149	2
	Conventional 2	494	173	321	831	1,041	676	365	333	1,213	1,002	211	4	1,203	1,016	187	3
	Proposed	231	202	29	798	1,001	999	2	1	1,000	1,000	0	0	1,000	1,000	0	0

E_d: number of decoded elements, E_r: number of right elements among decoded elements
E_w: number of wrong elements among decoded elements, E_m: number of missing elements (not decoded)

TABLE 5. Number of T_1 or T_2 occurrences.

		α											
		1			1.5			2			2.5		
		$n(S_{ne})$	$n(T_1)$	$n(T_2)$	$n(S_{ne})$	$n(T_1)$	$n(T_2)$	$n(S_{ne})$	$n(T_1)$	$n(T_2)$	$n(S_{ne})$	$n(T_1)$	$n(T_2)$
ExpA	Conventional 1	0	9	-	3	14	-	2	7	-	0	5	-
	Conventional 2	1	7	-	1	14	-	0	4	-	1	10	-
	Proposed	0	6	0	0	10	0	0	3	0	0	4	0
ExpB	Conventional 1	237	3049	-	313	3646	-	256	2702	-	187	1850	-
	Conventional 2	255	3094	-	306	3785	-	241	2528	-	183	1816	-
	Proposed	0	2988	3	0	2568	222	0	1570	143	0	1186	91
ExpC	Conventional 1	26	259	-	32	342	-	20	181	-	11	118	-
	Conventional 2	23	320	-	33	369	-	20	203	-	18	183	-
	Proposed	0	305	2	0	312	42	0	134	17	0	122	18

$n(S_{ne})$: number of non-pure cases detected by signature comparison
Conventional $n(T_1)$: number of detected T_1 cases
Proposed $n(T_1)$: number of avoided T_1 cases, $n(T_2)$: number of detected and resolved T_2 cases

of non-decoded elements, as shown in E_m and a number of incorrectly decoded elements, as shown in E_w for every case. On the other hand, our proposed algorithm decodes exactly d correct elements if the decoding is successful ($\alpha = 1.5$, 2, and 2.5 in ExpA and $\alpha = 2$ and 2.5 in ExpB and ExpC.)

When comparing the two conventional decoding cases, there was not much difference in the decoding results. Hence, it is not necessary to allocate a large number of bits to *idSum* or *sigSum*. As shown in Figures 6-8, the conventional methods have incorrect elements incorrectly decoded as distinct elements for every experiment. This implies that elements that are not included in any of the sets have been mistakenly decoded as distinct elements (owing to T_1 or T_2 errors). Even when every distinct element is decoded, E_w is not equal to zero; hence, the resulting d-IBF after the completion of the decoding process is not empty. In the cases of $\alpha = 2$ and $\alpha = 2.5$ in ExpB and ExpC of the conventional decoding methods, E_r is larger than d. This implies that distinct elements are duplicated during decoding.

In contrast, when the proposed decoding method is successful, every distinct element is decoded accurately, and there are no incorrectly decoded elements. Decoding is perfect in that every distinct element is decoded correctly, as indicated by E_r, and no incorrect element is decoded, as indicated by E_w. In other words, with the proposed decoding method, decoding completion confirms that every distinct element is identified without any incorrectly decoded elements. Consequently, every cell of the resulting d-IBF (after decoding was completed) had zero values.

These results imply that resolving T_1 and T_2 errors has a significant impact on the overall decoding performance. Furthermore, because element deletion is performed through a XOR operation (as shown in Algorithm 1), deleting an un-inserted element is the same as inserting an element. Hence, deleting a composite cell with a T_1 or T_2 error is equivalent to inserting an element. As more elements are inserted into a fixed-length d-IBF because of T_1 or T_2 errors, the probability of the occurrence of candidate pure cells decreases. Because the proposed decoding method avoids erroneous deletion, it is more likely to generate new candidate pure cells during decoding. As a result, the decoding process can be successful even for a d-IBF with a small number of cells and a small number of bits in each field.

Table 5 lists the number of T_1 and T_2 occurrences according to α. The S_{ne} column shows the number of cases where a cell with $|count| = 1$ is identified non-pure because the *sigSum* field of the cell is not equal to $g(idSum)$. There is little difference between the two conventional decoding cases, even though Conventional 1 uses significantly higher number of bits than Conventional 2. $n(T_1)$ and $n(T_2)$ columns represent the number of T_1 and T_2 occurrences, respectively. As shown, $n(T_1)$ is larger than S_{ne} for every case, which implies that identifying composite cells using T_1 cases is much more effective than using the *sigSum* field. T_1 cases are clearly identified if the hash indices generated by the *idSum* value do not include their own cell index. However, because T_2 cases are not identified during decoding, the number of T_2 cases is not shown when conventional decoding methods are used.

Once T_1 or T_2 errors occur, the decoding process of conventional decoding progresses differently from that of our proposed decoding method. The number of T_1 errors was much smaller with our method. When our proposed decoding method was used, no T_2 error occurred for every α value of ExpA. It is noteworthy that our algorithm does not perform decoding if T_1 occurs. For T_2 occurrences

TABLE 6. Decoding time (ns).

		α			
		1	1.5	2	2.5
ExpA	Conventional decoding 1	6,435	17,255	18,125	17,615
	Conventional decoding 2	3,065	10,615	9,245	12,315
	Proposed decoding	3,065	10,335	9,095	9,135
ExpB	Conventional decoding 1	2,168,605	5,265,205	6,164,795	5,384,715
	Conventional decoding 2	1,440,745	4,968,875	3,622,825	3,454,395
	Proposed decoding	1,494,745	3,394,135	3,074,145	2,917,735
ExpC	Conventional decoding 1	199,285	411,185	525,965	489,225
	Conventional decoding 2	144,605	334,595	401,135	365,685
	Proposed decoding	153,945	460,755	336,745	344,905

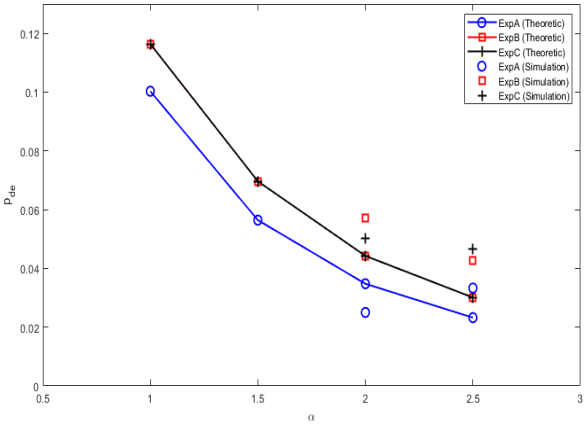


FIGURE 9. Theoretical probability (p_{de}) (Eq. (29)) and experimental results of the d-IBF decoding error because of composite cells satisfying Conditions 1 and 2. ExpA: $n_1 = 10$, $n_2 = 30$; ExpB: $n_1 = 5000$, $n_2 = 5000$; ExpC: $n_1 = 500$, $n_2 = 500$.

(because there is no way to avoid decoding), the proposed algorithm performs decoding for T_2 occurrences. However, our proposed algorithm detects and resolves errors by invalidating the decoded elements generated by T_2 errors.

Figure 9 shows the probability of decoding errors generated by the composite cells satisfying pure cell conditions 1 and 2 shown in Section III. The theoretical results are calculated using (29). The simulation results were compared with the theoretical results when the decoding process was completed without error. Hence, the simulation results of the proposed method for cases with $\alpha = 2$ and $\alpha = 2.5$ are shown, and they are calculated using $(n(T_1)+n(T_2))/kd$. The simulation results and theoretical analysis results exhibit similar trends.

Table 6 compares the decoding times, assuming a clock period of 10 ns, which is equal to a clock frequency of 100MHz. The conventional decoding algorithm is faster for small α values with decoding failures because the decoding process is aborted if no pure cells remain. However, the proposed method is faster than the conventional method, in which decoding is completed without errors. The proposed method includes the time required to resolve both types of errors, whereas the conventional method includes the time required to decode the wrong elements, which is larger than the time required to resolve T_1 or T_2 errors.

VI. CONCLUSION

For two given sets, each programmed into an invertible Bloom filter (IBF), a difference-IBF (d-IBF) is employed to remove common elements and detect unique elements exclusively present in each set, using a decoding process. The d-IBF is especially effective when the number of common elements is significantly larger than the number of distinct elements. This is because the size of the IBF can be set proportional to the sum of the distinct elements of the two sets and not to the entire number of elements. However, if the same decoding algorithm as that used for IBF decoding is used for d-IBF decoding, incorrect elements (which do not belong to either of the sets) are decoded. This occurs because of the errors during decoding. This study asserts that an extra pure-cell condition is necessary for d-IBF decoding. Two types of errors were identified that occurred during the conventional decoding process of a d-IBF and an efficient decoding method was proposed that addressed these errors. The simulation results show that the proposed method can decode every distinct element without incorrectly decoded elements, even with a small sized IBF.

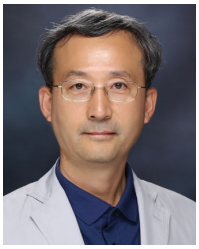
REFERENCES

[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.
[2] M. T. Goodrich and M. Mitzenmacher, "Invertible Bloom lookup tables," in *Proc. 49th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2011, pp. 792–799.
[3] J. Qian, Q. Zhu, and H. Chen, "Multi-granularity locality-sensitive Bloom filter," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3500–3514, Dec. 2015.
[4] W. Liu, Z. Xu, J. Tian, and Y. Zhang, "Towards in-network compact representation: Mergeable counting Bloom filter vis cuckoo scheduling," *IEEE Access*, vol. 9, pp. 55329–55339, 2021.
[5] H. Byun and H. Lim, "Learned FBF: Learning-based functional Bloom filter for key–value storage," *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1928–1938, Aug. 2022.
[6] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, Jan. 2004.
[7] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of Bloom filters for distributed systems," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 1, pp. 131–155, 1st Quart., 2012.
[8] D. Eppstein and M. T. Goodrich, "Straggler identification in round-trip data streams via Newton's identities and invertible Bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 2, pp. 297–306, Feb. 2011.
[9] S. Lee, H. Byun, and H. Lim, "Set reconciliation using ternary and invertible Bloom filters," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 11, pp. 11885–11898, Nov. 2023.
[10] D. Eppstein, M. Goodrich, F. Uyeda, and G. Varghese, "What's the difference? Efficient set reconciliation without prior context," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 218–229, Oct. 2011.

[11] C. Yang, Y. Liu, X. Tao, and F. Zhao, "Publicly verifiable and efficient fine-grained data deletion scheme in cloud computing," *IEEE Access*, vol. 8, pp. 99393–99403, 2020.
[12] C. Guo, R. Zhuang, C.-C. Chang, and Q. Yuan, "Dynamic multi-keyword ranked search based on Bloom filter over encrypted cloud data," *IEEE Access*, vol. 7, pp. 35826–35837, 2019.
[13] L. Ferretti, M. Marchetti, M. Andreolini, and M. Colajanni, "A symmetric cryptographic scheme for data integrity verification in cloud databases," *Inf. Sci.*, vol. 422, pp. 497–515, Jan. 2018.
[14] J. Jeong, J. W. J. Joo, Y. Lee, and Y. Son, "Secure cloud storage service using Bloom filters for the Internet of Things," *IEEE Access*, vol. 7, pp. 60897–60907, 2019.
[15] R. Zhang, J. Liu, T. Huang, T. Pan, and L. Wu, "Adaptive compression trie based Bloom filter: Request filter for NDN content store," *IEEE Access*, vol. 5, pp. 23647–23656, 2017.
[16] J. Lee, M. Shim, and H. Lim, "Name prefix matching using Bloom filter pre-searching for content centric network," *J. Netw. Comput. Appl.*, vol. 65, pp. 36–47, Apr. 2016.
[17] J. H. Mun and H. Lim, "Cache sharing using Bloom filters in named data networking," *J. Netw. Comput. Appl.*, vol. 90, pp. 74–82, Jul. 2017.
[18] J. Lee, H. Byun, and H. Lim, "Dual-load Bloom filter: Application for name lookup," *Comput. Commun.*, vol. 151, pp. 1–9, Feb. 2020.
[19] W. Fu, H. B. Abraham, and P. Crowley, "Synchronizing namespaces with invertible Bloom filters," in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst. (ANCS)*, May 2015, pp. 123–134.
[20] S. Jang, H. Byun, and H. Lim, "Dynamically allocated Bloom filter-based PIT architectures," *IEEE Access*, vol. 10, pp. 28165–28179, 2022.
[21] I. A. Khan, D. Pi, Z. U. Khan, Y. Hussain, and A. Nawaz, "HML-IDS: A hybrid-multilevel anomaly prediction approach for intrusion detection in SCADA systems," *IEEE Access*, vol. 7, pp. 89507–89521, 2019.
[22] C. E. Rothenberg, P. Jokela, P. Nikander, M. Sarela, and J. Ylitalo, "Self-routing denial-of-service resistant capabilities using in-packet Bloom filters," in *Proc. Eur. Conf. Comput. Netw. Defense*, 2009, pp. 46–51.
[23] A. Sánchez-Macián, P. Reviriego, J. A. Maestro, and S. Liu, "Single event transient tolerant Bloom filter implementations," *IEEE Trans. Comput.*, vol. 66, no. 10, pp. 1831–1836, Oct. 2017.
[24] P. K. Vairam, P. Kumar, C. Rebeiro, and V. Kamakoti, "FadingBF: A Bloom filter with consistent guarantees for online applications," *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 40–52, Jan. 2022.
[25] J. Risson and T. Moors, "Survey of research towards robust peer-to-peer networks: Search methods," *Comput. Netw.*, vol. 50, no. 17, pp. 3485–3521, Dec. 2006.
[26] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using Bloom filters," in *Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun.*, Aug. 2003, pp. 201–212.
[27] H. Lim, K. Lim, N. Lee, and K.-H. Park, "On adding Bloom filters to longest prefix matching algorithms," *IEEE Trans. Comput.*, vol. 63, no. 2, pp. 411–423, Feb. 2014.
[28] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended Bloom filter: An aid to network processing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 181–192, Oct. 2005.
[29] J. H. Mun and H. Lim, "New approach for efficient IP address lookup using a Bloom filter in trie-based algorithms," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1558–1565, May 2016.
[30] E. Tong, W. Niu, G. Li, D. Tang, L. Chang, Z. Shi, and S. Ci, "Bloom filter -based workflow management to enable QoS guarantee in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 39, pp. 38–51, Mar. 2014.
[31] G. Li, L. Guo, X. Gao, and M. Liao, "Bloom filter based processing algorithms for the multi-dimensional event query in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 37, pp. 323–333, Jan. 2014.
[32] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: A novel counter architecture for per-flow measurement," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 1, pp. 121–132, Jun. 2008.
[33] A. Kumar, J. Xu, and J. Wang, "Space-code Bloom filter for efficient per-flow traffic measurement," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2327–2339, Dec. 2006.
[34] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.
[35] H. Lim, J. Lee, H. Byun, and C. Yim, "Ternary Bloom filter replacing counting Bloom filter," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 278–281, Feb. 2017.
[36] S. Schildt, J. Morgenroth, and L. Wolf, "Efficient false positive free set synchronization using an extended Bloom filter approach," *Comput. Commun.*, vol. 36, nos. 10–11, pp. 1245–1254, Jun. 2013.
[37] *Modelsim*. Accessed: Aug. 1, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/software-kit/665990/intel-quartus-prime-lite-edition-design-software-version-18-1-for-windows.html
[38] *Source Code*. Accessed: Feb. 24, 2024. [Online]. Available: https://github.com/hyesooklim/d-IBF-Decoding-Implementation

EUNJI CHOI (Student Member, IEEE) is currently pursuing the bachelor's degree with the Department of Electronic and Electrical Engineering, Ewha Womans University, Seoul, South Korea. Her research interest includes bloom filter applications to various distributed algorithms.

JUNGWON LEE (Member, IEEE) received the B.S. degree from the Department of Mechatronics Engineering, Korea Polytechnic University, Gyeonggi-do, South Korea, in 2011, and the M.S. and Ph.D. degrees from the Department of Electronics Engineering, Ewha Womans University, Seoul, South Korea, in 2013 and 2017, respectively. From 2017 to 2018, she was a Postdoctoral Researcher with Ewha Womans University. From 2018 to 2020, she was a Research Professor with Ewha Womans University. She joined as a Senior Researcher with the Inter-University Semiconductor Research Center, Seoul National University, in 2021, where she is currently a Research Assistant Professor. Her research interests include named data networking (NDN), blockchain, physical unclonable function (PUF), and bloom filter (BF).



CHANGHOON YIM (Senior Member, IEEE) received the B.S. degree from the Department of Control and Instrumentation Engineering, Seoul National University, South Korea, in 1986, the M.S. degree from the Department of Electrical and Electronics Engineering, Korea Advanced Institute of Science and Technology, in 1988, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, in 1996. He was a Member of Technical Staff with Sarnoff Corporation and Bell Labs, Lucent Technologies. He was the Principal Engineer of Samsung Electronics. Since 2003, he has been a Professor with the Department of Computer Science and Engineering, Konkuk University.

HYESOOK LIM (Senior Member, IEEE) received the B.S. and M.S. degrees from the Department of Control and Instrumentation Engineering, Seoul National University, Seoul, South Korea, in 1986 and 1991, respectively, and the Ph.D. degree from the Electrical and Computer Engineering, University of Texas at Austin, in 1996. She was a Member of Technical Staff with Bell Labs, Lucent Technologies, for three and half years. She was a Hardware Engineer with Cisco Systems, for two years. Since 2002, she has been a Professor of electronic and electrical engineering with Ewha Womans University, Seoul. She was the Dean of the College of Engineering, from 2018 to 2020. She was the first female President of the Institute of Electronics and Information Engineers, in 2020. She was the Minister of Science and ICT of the Republic of Korea, for a year in May 2021. She is a member and also the Division Chair of the Electrical and Electronic Division, National Academy of Engineering of Korea.