Dynamically Allocated Bloom Filter-Based PIT Architectures

SAEYOUNG JANG1, (Member, IEEE), HAYOUNG BYUN2, (Member, IEEE), AND HYESOOK LIM1, (Senior Member, IEEE)

1Department of Electronic and Electrical Engineering, Smart Factory Convergence Major, Ewha Womans University, Seoul 03760, Republic of Korea.
2Department of Electronics Engineering, Myongji University, Yongin 17058, Republic of Korea.

Corresponding author: Hayoung Byun (e-mail: hbyun@mju.ac.kr).

This work was supported by 2021 Research Fund of Myongji University and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2021R1F1A1051646).

ABSTRACT As a key component in implementing Named Data Networking (NDN), Pending Interest Table (PIT) requires an efficient exact-matching algorithm for a scalable and fast PIT lookup. A Bloom filter (BF) is a memory-efficient data structure for performing exact matching operations. In this paper, three different BF-based PIT architectures are proposed: PIT using functional Bloom filters (FBF-PIT), PIT using counting Bloom filters with return values (rCBF-PIT), and a refined rCBF-PIT with signatures (R-rCBF-PIT). The proposed BF-based PITs incrementally allocate a new BF for storing multiple incoming faces of Interest packets with the same content name. For a Data packet lookup, the proposed PIT architectures simultaneously access every BF structure to find matching faces and delete the faces (i.e., matching Interest packet information). The functional Bloom filter (FBF) used in an FBF-PIT is a key-value data structure that stores values only without keys. However, because the number of non-reusable conflict cells in the FBF increases as the number of stored packets increases in the FBF-PIT, the indeterminable rate increases. To decrease the indeterminable rate, we propose the rCBF-PIT, which uses counting Bloom filters with return values (rCBFs), allowing reusable conflict cells. False positives for Interest packets lead to incorrect deletions that can cause false negatives for incoming Data packets. Because most of the false positives occur in the first BF structure, we finally propose the R-rCBF-PIT, in which the first rCBF is replaced with an rCBF with a signature field. The proposed PITs also provide an aging mechanism using a valid bit and a hit bit for entry expiration. Simulation results show that rCBF-PIT and R-rCBF-PIT both reduce the indeterminable rate by more than 81% compared with FBF-PIT. The results also show that R-rCBF-PIT resolves false negatives caused by incorrect deletions by including the signature fields in the first rCBF.

INDEX TERMS Bloom filter, dynamic data structure, Named Data Networking, Pending Interest Table.

I. INTRODUCTION

The forwarding engine of an NDN router includes three components: a Content Store (CS), Forwarding Information Base (FIB), and Pending Interest Table (PIT). The CS caches incoming Data packets [11]–[16], the FIB forwards Interest packets toward content providers [17]–[22], and the PIT stores unsatisfied Interest packets [23]–[28]. Unlike IP routers, NDN routers serve as a cache by temporarily storing data from incoming Data packets. Therefore, if the CS of an intermediate router along the path between a customer and a provider has the cached data corresponding to an
Interest packet of the customer, the Interest packet is not forwarded toward the provider and the consumer can receive the requested data more quickly.

PIT keeps track of Interest packets that have not yet been satisfied [23]–[28]. PIT allows Interest and Data packets to be routed without a source or a destination address by using content names. An entry of PIT is created for the content name and incoming face of each unsatisfied Interest packet in an NDN router. A matching entry is removed from the PIT for each received Data packet.

Constructing a scalable, fast, and efficient PIT is challenging owing to the following characteristics involved in a PIT lookup. First, whenever Interest and Data packets arrive at an NDN router, the insert (or update) and delete operations should be repeated in real time. An efficient exact matching algorithm is required to check if a content name for an incoming Interest or Data packet is stored in the PIT. Second, multiple Interest packets for a single content can arrive from different faces, and it is not known in advance how many faces the same request will arrive from. Furthermore, the PIT should provide an entry expiration mechanism to remove old data and free up memory. Therefore, it is essential to explore an efficient PIT lookup algorithm that satisfies these characteristics and meets the high-speed packet arrival rate.

In this paper, we propose three Bloom filter (BF) based PIT architectures using various BF structures. Proposed PITs enable a fast name lookup with exact matching, a low memory cost by dynamically allocating memory, an efficient insertion and deletion for an Interest and Data packet, and entry expiration. For Interest packets, the proposed BF-based PITs store multiple incoming faces of Interest packets with the same content name by incrementally creating a BF. When a Data packet arrives, all matching faces are looked up at the same time by accessing every BF simultaneously.

A BF is a space-efficient structure that stores membership information [29]–[31]. Among many BF variants [32]–[35], the proposed PIT architectures use functional Bloom filters (FBFs), counting Bloom filters with return values (rCBFs), and rCBFs with signatures. The rCBF and rCBF with signatures are the novel data structures proposed in this study. The proposed PIT architectures reduce the memory requirements by incrementally adding a BF structure and enable fast lookup by simultaneously searching all BF structures used in the PIT.

The remainder of this paper is organized as follows. Section II describes various Bloom filter structures including a counting Bloom filter and a functional Bloom filter. Section III describes the PIT in NDN and existing PIT lookup algorithms. Section IV describes our proposed Bloom filter-based PIT architectures using FBFs, rCBFs, and rCBFs with signatures. This section also includes the entry expiration mechanism. Section V compares the performance of the three proposed PIT lookup architectures and discusses the performance compared with other previous algorithms. Section VI concludes briefly the paper.

II. BLOOM FILTER STRUCTURES

A. COUNTING BLOOM FILTER

A Bloom filter (BF) is a bit-vector-based structure that identifies whether an input is a member of a stored set [29]. A BF comprising an array of m bits can return only the membership of an input. Whereas a standard BF does not provide delete operations for programmed elements, a counting Bloom filter (CBF) can provide membership queries as well as deletions for programmed elements [32], [36]. A cell of an m-cell CBF consists of an R-bit counter, and each cell counts the number of programmed elements. A CBF employs k hash functions for programming, querying, and deleting. Let n be the number of programmed elements in the set. The optimal number of hash functions k is obtained as follows [30], [31].

\[
k = \frac{m}{n} \log 2.
\]  

(1)

In programming element \(x_1\) to a CBF, \(k\) counters mapped to \(k\) hash functions are incremented by 1. Counters with a maximum count of \(2^R - 1\) cannot be incremented. Figure 1 shows a CBF with \(k = 3, m = 8,\) and \(R = 2,\) for set \(S = \{x_1, x_2, x_3\}\). The counters mapped to \(h_1(x_3), h_2(x_3),\) and \(h_3(x_3)\) are incremented by 1.

![Figure 1. Counting Bloom filter.](image)

In querying an input, a CBF can return a **negative** or **positive**. If at least one of \(k\) counters has 0, the CBF returns a negative, which means that the input is not a member of the programming set. If all \(k\) counters are greater than zero, the CBF returns a positive, which means that the input would be a member of the set. However, it is possible that the positive is false because of hash collisions.

In deleting an inserted element, all counters mapped to \(k\) hash functions are decremented by 1. A counter with \(2^R - 1\) can cause false negatives if more than \(2^R - 1\) elements were mapped to a counter in programming and \(2^R - 1\) or more elements are deleted from the counter.

B. FUNCTIONAL BLOOM FILTER

A functional Bloom filter (FBF) can return a value corresponding to an input as well as membership information regarding whether the input is an element of a set [35], [37]. The FBF has been used as a key-value structure in various network applications owing to its space-efficient characteristics [19], [34], [38]–[40]. The FBF utilizes the fact that various combinations of hash indexes of each key can be used to distinguish one key from other keys. For set \(S = \{(x_1, v_1), (x_2, v_2), \ldots, (x_n, v_n)\}\), an FBF only stores the value \(v_i\) corresponding to key \(x_i\) without storing the key itself. An FBF is an array of \(m\) cells, and each cell comprising
L bits can represent $0 \sim 2^L - 1$. All cells are initialized to zero. $v_i$ for $1 \leq i \leq n$ is a value within the range \{1, 2, $\ldots$, $2^L - 2$\}. The maximum value, $2^L - 1$, is reserved for conflict in which two or more return values are mapped. An FBF employs $k$ hash functions for programming, querying, and deleting, and the optimal number of hash functions used in an FBF is the same as in (1). The terms inserting and searching can be used instead of programming and querying because the FBF is used as a key-value structure.

When inserting an element, for key $x_i$ in $S$, every cell pointed to by $k$ hash indexes is set to $v_i$ for $1 \leq i \leq n$. However, if hash index collision occurs because two or more values are mapped to the same cell, the cell is set to $2^L - 1$ and it is called a conflict cell. Figure 2 shows an FBF with $k = 3$ and $m = 8$ for $S = \{(x_1, v_1), (x_2, v_2), (x_3, v_3)\}$. To program element $(x_3, v_3)$ to an FBF, $k$ hash indexes corresponding to $x_3$ are used to access the FBF cells. $v_3$ is stored in the cells pointed to by $h_1(x_3)$ and $h_2(x_3)$, and the conflict cell pointed to by $h_3(x_3)$ has $2^L - 1$ (denoted by X in Fig. 2) because the cell was already programmed with other values.

When searching for an input, an FBF can return a negative, positive, and indeterminable. If at least one of $k$ cells has 0 or if any of the $k$ cells except conflict cells have different values, the FBF returns a negative. If all $k$ cells except conflict cells have the same value, the FBF returns the value, which is a positive. It is possible that positives may be false because of hash index collisions. If all $k$ cells are conflict cells, it is termed an indeterminable. False positives and indeterminables are classified as search failure.

In deleting an inserted element, the same $k$ hash functions are used to obtain the hash indexes of a key. If every cell pointed to by $k$ hash indexes is not zero and if every cell except conflict cells has the same value, the element can be deleted from the FBF by resetting the non-conflict cells to zero. For example, if key $x_3$ is deleted in Fig. 2, cells pointed to by $h_1(x_3)$ and $h_3(x_3)$ are set to 0, but the cell pointed to by $h_2(x_3)$ with hash collision remains unchanged. If all $k$ cells are conflict cells, the element is undeletable. Elements with indeterminables cause undeletables.

III. NAMED DATA NETWORKING

In Named Data Networking (NDN) [1], [2], data are requested and provided based on content names. Figure 3 illustrates packet lookup and forwarding process at an NDN router. Figure 3(a) shows the case in which an Interest packet is arrived at the NDN router. The CS is first examined to find a matching Data packet corresponding to the Interest.
characteristics of the PIT should be considered when designing PIT lookup algorithms. Multiple Interests for the same content should be merged into a single PIT entry, which is known as Interest aggregation. In addition, dynamic data structures suitable for frequent update operations (i.e., insertion upon Interest packets and deletion upon Data packets) are required in a PIT.

Figure 4 shows an example of a PIT in an NDN router. Whereas the current IP network uses IP addresses to communicate, NDN communication is based on URL names with a hierarchical structure as content names. The URL names and incoming faces for Interest packets are stored in a PIT. As shown in Fig. 4, content name /com/airflight/find-us and incoming faces 0 and 2 are stored in an entry because Interest packets with the same name have arrived at the router from faces 0 and 2. When a Data packet with name /com/airflight/find-us arrives at the NDN router, exact matching for the name occurs. Therefore, the Data packet is forwarded to faces 0 and 2, and the matching entry is deleted since the Interest packets with name /com/airflight/find-us are sent as a response.

B. EXISTING PIT LOOKUP ALGORITHMS

Fingerprint-only PIT (FO-PIT) [23] uses a d-left hash table for the pending Interest packets. A d-left hash table consists of d subtables and each bucket in a subtable can hold several entries. A d-left hash table stores the information of an Interest packet in the least loaded bucket among buckets pointed to by d hash functions. FO-PIT using a d-left hash table stores fingerprints (i.e., signatures) instead of content names and uses a bit vector to store the list of incoming faces. However, the bit vector may not be a suitable representation as the number of faces of an NDN router increases. In addition, the memory requirements of hash table-based structures are generally greater than those of BF-based structures [37].

DiPIT [24] uses a CBF per face and a shared BF to store the information of Interest packets. As a simple lookup algorithm, the DiPIT does not support Interest aggregation. Hence, same content requests are forwarded, which creates unnecessary traffic, and for a Data packet, all sub-PITs (i.e., CBFs) should be looked up. Furthermore, because a CBF is required for each face, memory cannot be dynamically allocated according to incoming Interest packets. In other words, even if no Interest packet arrives from face v, the CBF for face v is initially constructed.

MaPIT [26] uses a mapping Bloom filter comprising an index table in on-chip memory and a packet store in off-chip memory, and a CBF in the off-chip memory. An index table consisting of a BF and a mapping array is utilized to access the packet store recording the information of Interest packets by obtaining the offset address of the packet store. However, since the information of Interest packets is stored in the off-chip memory, the off-chip memory access is inevitable for every packet, and hence the PIT lookup performance is degraded.

FTDF-PIT [28] with a fast two-dimensional filter, consisting of a matrix, uses a quotient and the remainder of a content name in a packet as two indexes to map the packet information to a position inside the matrix. The FTDF-PIT can be looked up in O(1). However, because the PIT uses a single hash function, its performance depends heavily on the function. In addition, the PIT uses a bit vector to store incoming faces, and the bit vector is not a suitable representation as the number of faces of an NDN router increases.

IV. BLOOM FILTER-BASED PIT ARCHITECTURES

We propose three Bloom filter-based PIT architectures that allow dynamic memory allocation for an efficient packet lookup. In packet lookup and forwarding processes of the NDN, packet information should be frequently inserted and deleted. Moreover, multiple requests for the same data can arrive from different faces. Therefore, efficient data structures that can effectively process dynamic data with low memory costs are required.

Considering these design requirements, this paper proposes three PIT architectures using various BF structures: PIT using functional Bloom filters (FFB-PIT), PIT using counting Bloom filters with return values (rCBF-PIT), and a refined rCBF-PIT with signatures (R-rCBF-PIT). FFB and rCBF are memory-efficient key-value structures that store incoming faces (i.e., values) only, without content names (i.e., keys). In addition, rCBF is a novel data structure suitable for dynamic data. The proposed architectures use multiple BFs to store multiple incoming faces for a single content. All BF structures in a single PIT have the same BF size (i.e., the same number of cells) and employ the same hash indexes. Hence, the same cells are pointed to by the hash indexes for packets with the same name, which enables efficient handling of PIT operations (insert, update, and delete). In particular, for Data packets, O(1)-time lookup is possible.

Algorithm 1 presents Interest packet processing for the three proposed PITs. The overall procedure of the three PIT architectures is the same, despite their different details. When Interest packet PI with name x arrives from face v, the PIT lookup algorithm first checks if the content for PI is in the CS (lines 1-2). If the content for PI is not in the CS, the algorithm sequentially searches from the first BF structure.
Algorithm 1: Interest packet processing for three proposed PITs

```
Input: $P_I$: Interest packet with name $x$ from face $v$
/* PIT has $b$ number of BF structures */
/* $B_i$: $i^{th}$ BF structure in PIT for $1 \leq i \leq b$ */
1 if Data with $x$ in CS then  
2     return matching Data;
3 else  
4     $D \leftarrow$ False;
5     for $i \leftarrow 1$ to $b$ do  
6         search $B_i$;
7         if $B_i$ returns positive then  
8             $D \leftarrow$ True;
9             if $B_i$ returns $v$ then  
10                break;
11         end
12     end
13     if insert condition satisfied in $B_i$ then  
14         // insert and update operations
15         insert $v$ to $B_i$;
16         break;
17     end
18     if $i == b$ then  
19         allocate $B_{b+1}$ in PIT;
20         insert $v$ to $B_{b+1}$;
21     end
22     if $D$ then  
23         drop $P_I$;
24     else  
25         send $P_I$ to FIB to forward $P_I$;
26     end
```

Algorithm 2 presents the Data packet processing for the three proposed PITs. When Data packet $P_D$ arrives in an NDN router, all BF structures in the PIT are simultaneously looked up to return matching faces (line 1). If at least one face is returned, $P_D$ is forwarded to all returned faces (lines 2-3), and the matching entries are then deleted (line 4). The time complexity of the algorithm for Data packet lookup is $O(1)$, because all of $b$ BF structures in the PIT are simultaneously looked up.

```
Algorithm 2: Data packet processing for three proposed PITs

Input: $P_D$: Data packet with name $x$
/* PIT has $b$ number of BF structures */
/* $B_i$: $i^{th}$ BF structure in PIT for $1 \leq i \leq b$ */
1 lookup PIT: // simultaneously search every $B_i$
2 if at least one face is returned then  
3     forward $P_D$ to all returned faces;
4     delete the entries in PIT; // delete operation
5     if $P_D$ not in CS then  
6         cache $P_D$ in CS;
7     end
8 else  
9     drop $P_D$;
10 end
```

We first describe FBF-PIT in detail, and then describe rCBF-PIT, which is more suitable for dynamic data than the FBF-PIT. Finally, we propose R-rCBF-PIT to solve the problem of incorrect deletions of rCBF-PIT.

A. FBF-PIT

The basic idea of the proposed FBF-PIT was briefly introduced in [41]. A functional Bloom filter is a variant of a BF that returns the value corresponding to a stored key. In the proposed PIT using multiple FBFs (FBF-PIT), a key is the content name included in an Interest packet, and the return value is the incoming face where the Interest packet arrives. The FBF-PIT creates a new FBF when hash index collisions exceed a pre-defined collision threshold in already created FBFs.

Figure 5 indicates an example of packet lookup in FBF-PIT. Figure 5(a) shows an Interest packet lookup including insert and update operations. For example, Interest packets with name $x_1$ arrive from faces $v_1$, $v_2$, $v_3$, and $v_4$, an Interest packet with name $x_2$ arrives from face $v_5$, and Interest packets with name $x_3$ arrive from faces $v_6$ and $v_7$ in this sequential order.

For a single FBF of FBF-PIT with $m$ cells and $k$ hash indexes, let $k_c$ be the number of conflict cells among the accessed $k$ cells, and let $k_{th}$ be the pre-defined collision threshold. Here, $k_{th}$ distributes Interest packets to multiple FBFs, resulting in reduced indeterminable rates in the Data packet lookup.

For a given Interest packet from face $v$, the PIT lookup algorithm of FBF-PIT first searches for the first FBF (i.e., $B_1$). If $B_1$ returns face $v$, the algorithm completes the Interest packet lookup, and the packet is dropped because the packet is a duplicate. Otherwise, the algorithm compares $k_c$ and $k_{th}$ in $B_1$. If $k_c \leq k_{th}$, the algorithm ends by inserting $v$ in $B_1$, and conflict value $X$ is set to $k_c$ cells. If $k_c > k_{th}$ in $B_1$, $B_2$ is examined to check whether $k_c \leq k_{th}$. If there are no more FBFs to examine, a new FBF is incrementally created.
Fig. 5(a), in the case of the Interest with name $x$, to insert the face information of the packet. As shown in Fig. 5(b), the first two FBFs in FBF-PIT return faces $v_3$. When at least one of the FBFs returns a positive, the Data packet is forwarded to the faces and then the FBF-PIT are looked up in parallel. Each FBF can return a positive, negative, and indeterminable values in the cells pointed to by $k_c$ from face $v_3$, the first two FBFs in FBF-PIT return faces $v_5$ and $v_6$. However, in case of a false positive, the deletion results in the deletion of faces for other content names and causes false negatives.

The proposed FBF-PIT enables a fast Data packet lookup with a single access and prevents memory waste by incrementally creating a new FBF. However, a Data packet with a false positive result in incorrect deletions, leading to false negatives for other packets later. In addition, as content information is dynamically inserted and deleted, the number of conflict cells increases, thereby increasing the indeterminable rate in the packet lookup.

### B. rCBF-PIT

#### 1) Counting Bloom filter with return values

The concept of the proposed counting Bloom filter with return values (rCBF) was briefly introduced by the same authors in [42]. As a key-value structure, the rCBF is suitable for dynamic data. Figure 6 presents an rCBF with an array of $m$ cells using $k$ hash indexes. Each cell in the rCBF consists of two fields: an $L$-bit value field and an $R$-bit counter field. The value field has a range of $\{1, 2, \cdots, 2^L - 1\}$ and stores multiple values using the XOR operation. The counter field stores the number of stored values ($c$) in the cell.

In the rCBF, deletion can be performed including conflict cells, except when counters in the cells have maximum count $c_{\text{max}}$ (i.e., $2^R - 1$). Hence, rCBF can effectively process dynamic data because insert and delete operations can be performed in every cell pointed to by $k$ hash indexes. The optimal number of hash functions used in an rCBF is the same as that in (1).

All cells in the rCBF are initialized to zero. When inserting element $(x, v)$, the values stored in $k$ cells are XORed with $v$, and counters in $k$ cells are incremented by 1. If a counter already has $c_{\text{max}}$, the counter maintains the maximum count. Figure 6 shows an example of an rCBF with $m = 8$, $k = 3$, and $R = 2$ (i.e., $c_{\text{max}} = 2^R - 1 = 3$). For $x = v_3$, value $v_1$ is XORed with stored value $v_1 \oplus v_2$ in the cell pointed to by $h_1(x_3)$, and it is XORed with 0 in the other two cells. The counters in the cells pointed to by the three hash indexes are incremented by 1.

When searching for an input, an rCBF can return a negative, positive, and indeterminable. If at least one of $k$ cells has 0 or if any of the $k$ cells with a single count ($c = 1$) have different values, the rCBF returns a negative. If all the $k$ cells with a single count have the same value, the rCBF returns the value, which is a positive. Returned positives might be false because of hash index collisions. If all the $k$ cells have counts with $c > 1$, the search ends in failure, which is termed an indeterminable. False positives and indeterminables are classified as search failures.

In deleting element $(x, v)$, $v$ is XORed with all stored values in the cells pointed to by $k$ hash indexes of $x$, and all counters in the cells are decremented by 1. However, the counters with $c_{\text{max}}$ are not decremented to prevent false negatives in the search procedure. If every counter has $c_{\text{max}}$, the element cannot be deleted in the rCBF, which is termed a undeletable. For example, in Fig. 6, in deleting $x_3$, the $k$ cells would have values of $v_1 \oplus v_2$, 0, and 0 by XORing with $v_3$. However, only two counters with 1 are decremented to.

**FIGURE 6.** Counting Bloom filter with return values (rCBF).
zero and the counter in the cell pointed to by $h_2(x_3)$ cannot be decremented because the counter has $c_{\max}$, where $R = 2$ and $c_{\max} = 3$.

Assuming the same number of cells, an FBF and an rCBF have the same indeterminable rate for a static set. However, for a dynamic set, the rCBF would be better than the FBF in terms of indeterminable and undeletable rates. As insert and delete operations are repeated, while all cells except the cells with $c_{\max}$ can be deleted in the rCBF, the number of unusable conflict cells in the FBF increases because delete operations cannot be performed in conflict cells.

2) rCBF-PIT architecture

Written by the same authors of this paper, [41] only includes the basic idea of the proposed FIB-PIT, and it does not include rCBF-PIT and R-rCBF-PIT. [42] only includes the basic concept of the rCBF in terms of data structures, and it does not include any of PIT algorithms. In this paper, we propose a PIT using multiple rCBFs (rCBF-PIT) more suitable for dynamic data than FBF-PIT.

As with FBF-PIT, rCBF-PIT incrementally creates a new rCBF, and for a Data packet, all matching faces can be returned with a single access. Figure 7 shows the packet lookup in the rCBF-PIT. For a single rCBF of rCBF-PIT with $m$ cells and $k$ hash indexes, let $k_c$ be the number of cells already storing one or more values among the $k$ cells accessed, and let $k_{th}$ be a pre-defined collision threshold. In the proposed rCBF-PIT, to utilize cells with $c_{\max}$, if at least one of $k$ cells with $c_{\max}$ exists, Interest information is stored in the next rCBF. Hence, delete operations can be performed in conflict cells, which is the cells with $c_{\max}$. Moreover, the use of counters allows the information of Interest packets to be distributed across multiple rCBFs rather than stored intensively in a single rCBF (i.e., first rCBF), which results in a reduced indeterminable rate in a Data packet lookup.

Figure 7(a) shows an Interest packet lookup in rCBF-PIT. When an Interest packet arrives from face $v$, the PIT lookup algorithm of rCBF-PIT first searches for the first rCBF (i.e., $B_1$). If $B_1$ returns face $v$, the algorithm then completes the Interest packet lookup. Otherwise, the algorithm examines whether $k_c \leq k_{th}$ in $B_1$ and whether $c < c_{\max}$ in all $k$ cells. If $k_c \leq k_{th}$ and all counters are smaller than $c_{\max}$, the algorithm ends by inserting the face in the rCBF through the XOR operation. If $k_c > k_{th}$ in the rCBF or if at least one of the $k$ counters has $c_{\max}$, the next rCBF is examined. If there are no more rCBFs to examine, a new rCBF is incrementally created to insert the face information of the packet. As shown in Fig. 7(a), for an Interest with name $x_3$ from face $v_7$, because $k_c > k_{th}$ in $B_1$ (or the cell pointed to by $h_2(x_3)$ already has $c_{\max}$ in $B_1$), face $v_7$ is XORed in $B_2$ (i.e., update).

Figure 7(b) shows a Data packet lookup in rCBF-PIT. When the Data packet arrives, every rCBF is simultaneously searched with $k$ hash indexes obtained by the content name included in the packet. Each rCBF can return a positive, negative, and indeterminable. When at least one of the rCBFs returns a positive, the Data packet is forwarded to the returned faces. The faces are then deleted from the matching rCBFs by XORing the faces and decrementing counters by 1. In Fig. 7(b), for the Data packet with name $x_3$, the two rCBFs in rCBF-PIT return faces $v_6$ and $v_7$. Hence, the Data packet is forwarded to the faces, which are then deleted from the rCBF-PIT by XORing $v_6$ and $v_7$, respectively, and by decrementing all counters by 1. However, if there is a false positive resulting in the deletion of a face corresponding to other content names, false negatives can occur.

The proposed rCBF-PIT solves the weakness of FBF-PIT, which is the high indeterminable rate in a packet lookup as the content information is dynamically inserted and deleted. However, for a Data packet lookup, incorrect deletions resulting from false positives still exist, leading to false negatives later for other packets. Because a single false positive causes a delete operation in $k$ cells, the $k$ cells storing other content requests can be affected. Hence, because quite a few false negatives can occur even if a single incorrect deletion occurs, it is necessary to reduce the false positive rate in order to improve the accuracy of a packet lookup.

![FIGURE 7. Packet lookup in rCBF-PIT.](image)
C. R-rCBF-PIT

1) rCBF with signatures

We propose rCBF with signatures to reduce the false positive rate of rCBF. In the rCBF with signatures, a $G$-bit signature field is added to each cell. Figure 8 presents an rCBF with signatures. The rCBF with signatures uses additional hash function $g(\cdot)$ to obtain the signature of a key, which is different from the $k$ hash functions. In the three operations (insert, search, and delete), the signature of a key is obtained by hashing the key with $g(\cdot)$.

![FIGURE 8. rCBF with signatures.](image)

When inserting element $(x, v)$, the stored values in $k$ cells are XORed with $v$, and $k$ counters are incremented by 1 except for counters with $c_{\text{max}}$ (i.e., $2^R - 1$). In addition, signature $s$ (i.e., $s = g(x)$) is also XORed with stored signatures in the $k$ cells. Figure 8 shows an example of an rCBF with signatures, with $m = 8$, $k = 3$, and $R = 2$. For $x_{3,1}$, value $v_3$ is XORed with stored value $v_1 \oplus v_2$ in the cell pointed to by $h_2(x_{3,1})$, and is XORed with 0 in the other two cells. Signatures are stored in the same manner. In addition, the counters in the $k$ cells are incremented by 1.

When searching for input $y$, an rCBF with signatures can return a negative, positive, and indeterminable. If at least one of $k$ cells has 0, if any of the $k$ cells with a single count ($c = 1$) have different values, or if at least one of the cells with a single count does not have signature $g(y)$ (even though the cells with a single count have the same value), the rCBF returns a negative. If all the cells with a single count have the same value and have signature $g(y)$, the rCBF returns the value, which is a positive. Using signatures significantly reduces the false positive rate of an rCBF because a false positive occurs only when both hash index collision and signature collision occur. If all $k$ cells have counters with $c > 1$, the search ends in failure, which is termed an indeterminable. False positives and indeterminables are classified as search failures.

In deleting element $(x, v)$, $v$ is XORed with all stored values in $k$ cells, and every counter in the cells is decremented by 1 except for counters with $c_{\text{max}}$. In addition, signature $s$ is also XORed with the stored signatures in the $k$ cells. If every counter has $c_{\text{max}}$, the element cannot be deleted in the rCBF with signatures, which is termed an undeletable.

2) R-rCBF-PIT architecture

To improve the accuracy of packet lookup by reducing the error rate resulting from false positives in rCBF-PIT, we propose a refined rCBF-PIT (R-rCBF-PIT), in which only the first rCBF of R-rCBF-PIT is replaced with an rCBF with signatures. Because the proposed BF-based PITS store Interest information sequentially from the first BF structure, the number of packets stored in the first structure is the largest. Thus, although only the first rCBF is replaced with an rCBF with signatures, false positives causing incorrect deletions can be effectively reduced, thereby reducing false negatives.

Figure 9 describes the packet lookup of R-rCBF-PIT. Since the first rCBF in rCBF-PIT in Fig. 7 is replaced with an rCBF with signatures, an insertion is differently performed only in the first BF structure. The insertion is performed identically in other rCBFs to the rCBF-PIT. A Data packet lookup is still completed with only a single access.

When an Interest packet with content name $x$ arrives from $v$, the PIT lookup algorithm of R-rCBF-PIT first searches the rCBF with signatures (i.e., $B_1$) using $k$ hash indexes and the signature of $x$ (i.e., $g(x)$). If $B_1$ returns face $v$, the algorithm completes the Interest packet lookup and the packet is dropped. Otherwise, the algorithm examines whether $k_c \leq k_{th}$ in $B_1$ and whether $c < c_{\text{max}}$ in $k$ cells. If $k_c \leq k_{th}$ and $c < c_{\text{max}}$ in all $k$ cells, the algorithm ends by inserting the face in $B_1$ by XORing $v$ in the value fields, XORing $g(x)$ in the signature fields, and incrementing $k$ counters by 1. If $k_c > k_{th}$, or if at least one of $k$ counters has $c_{\text{max}}$, the next insertion is differently performed only in the first BF structure.
rCBF is examined, and the procedure is the same as that in rCBF-PIT. If there are no more rCBFs to examine, a new rCBF is incrementally created to insert the face information of the packet.

When a Data packet with content name $y$ arrives, every BF structure is simultaneously searched with $k$ hash indexes. When at least one of the BF structures returns a positive, the Data packet is forwarded to the returned faces. A delete operation for $y$ is then performed in the BF structures returning positives by XORing the faces, XORing $g(y)$, and decrementing the $k$ counters by 1. For example, for a Data packet with name $x_3$, as shown in Fig. 9(b), the packet is forwarded to faces $v_6$ and $v_7$ and then the faces are deleted from R-rCBF-PIT by XORing $v_6$ in $B_1$ and $v_7$ in $B_2$, XORing $s_3$ (i.e., $g(x_3)$) in $B_1$ (i.e., $s_1 \oplus s_2 \oplus s_3 \oplus s_3 = s_1 \oplus s_2$), and decrementing all counters by 1.

Figure 10 compares the Data packet lookup between rCBF-PIT and R-rCBF-PIT. For a Data packet with name $y$, the content with name $y$ has never been requested and hash index collisions occur in all $k$ cells storing $v_6$ of $B_1$. In the case of rCBF-PIT in Fig. 10(a), face $v_6$ is returned from $B_1$, and a delete operation is then performed in the $k$ cells, which is an incorrect deletion leading to false negatives. In other words, in $B_1$, the value in the cell pointed to by $h_i(y)$ (for $i = 1, 2, 3$) is XORed with $v_6$ (i.e., deletion) even if the content with name $y$ has never been requested. Hence the cells are ruined. However, in case of R-rCBF-PIT in Fig. 10(b), an incorrect deletion does not occur because signature $g(y)$ would not be the same as $s_3$ in the cell pointed to by $h_2(y)$.

Because most of false positives occur in the first BF structure ($B_1$), R-rCBF-PIT can resolve almost all false positives that cause false negatives later by incorrect deletions. Hence, R-rCBF-PIT improves the PIT lookup accuracy of rCBF-PIT.

**D. ENTRY EXPIRATION OF BF-BASED PITS**

Entry expiration is required to prevent Interest flooding attacks and avoid explosively increasing the memory requirement of a PIT over time. In the proposed BF-based PITSs, an aging mechanism is applied using two bits for each cell: hit and valid [43].

When an Interest packet arrives, valid and bit hits in the accessed cells are set to 1. In other words, if the incoming face of the Interest packet is inserted in $B_0$, all of valid and hit bits in the accessed $k$ cells of $B_0$ are set to 1. In each specific aging time $T_e$, all the hit bits in every cell are reset to 0 and valid bits with hit bit 0 are reset to 0, implying that these entries were not used during the last $T_e$ period. Valid bits with hit bit 1 are unchanged, implying that these entries were used or created during the last $T_e$ period. For cells with hit $= 0$ and valid $= 0$, delete operations need to be invoked. Although this approach is extremely simple, it enables the effective implementation of timer function without recording accurate timestamps for each cell.

Figure 11 shows an aging mechanism for rCBF-PIT in chronological order. In Fig. 11(a), when an Interest packet with name $x_3$ arrives from face $v_5$, an update operation is performed, which means that valid and hit bits in the accessed cells are set to 1. In other words, during $T_e$ before the packet arrives, because the cells pointed to by $h_1(x_3)$, $h_2(x_3)$, and $h_3(x_3)$ were not accessed, hit bits in the cells change 0 to 1. The cells of $B_2$ pointed to by $h_1(x_3)$ and $h_3(x_3)$ do not have return values when the packet arrives, and hence valid bits in the cells also change 0 to 1.

Figure 11(b) shows that all the hit bits are reset to 0 after $T_e$, because the cells were not accessed during the last $T_e$ period. For a Data packet with name $x_3$, a delete operation is performed as shown in Fig. 11(c). Since cells pointed to by $h_2(x_3)$ are accessed before another $T_e$ has elapsed, and they still have values after the delete operation, the hit bits in the cells change 0 to 1. Because cells pointed to by $h_1(x_3)$ and $h_3(x_3)$ do not have values after the operation, valid and hit bits in the cells are set to 0.

Even though entry expiration can cause false negatives, PIT does not need to notify the content consumers of entry expiration through the faces because the content consumers who have not received corresponding Data packets are assumed to re-request desired contents [23]. In other words, if a pending Interest packet expires, content consumers will resend the Interest packet even if they do not know the expiration. In addition, informing consumers can create unnecessary traffic because the requests of consumers could
be satisfied by other routers in different paths. Even if false negatives in PIT do not result in serious problems unlike other applications, reducing them is still important for a PIT lookup algorithm to meet the high-speed packet arrival rate.

V. PERFORMANCE EVALUATION

Performance evaluation has been carried out using random URL names provided by ALEXA [44], which is a web information company. Content names (i.e., URLs) for Interest and Data packets are extracted for our simulation and PIT lookup algorithms are constructed using C++ language. Since our proposed PIT algorithms do not require router interactions, simulation using network simulator is not considered. Since several Interest packets requesting the same content can arrive from different faces, we reasonably assume that Interests for a single content arrive from one to four different faces in our simulation: a quarter of the content requests from a single face, another quarter from two different faces, the third quarter from three different faces, and the final quarter from four different faces. Assuming that the number of requested contents included in set \( S \) is \( N \) (i.e., \( n(S) = N \)), the number of Interest packets for our simulation is \( 2.5N \), since

\[
0.25N(1 + 1 \times 2 + 1 \times 3 + 1 \times 4) = 2.5N.
\]

Data packets consist of packets with requested contents (in \( S \)) and packets with unrequested contents (in \( S^c \)). Because Data packets for unrequested contents rarely arrive, the number of unrequested contents in set \( S^c \) is assumed to be 1% in the simulation. Hence, the number of Data packets is 1.01\( N \). Table 1 shows three experimental cases: \( N = 2^{10} \), \( N = 2^{14} \), and \( N = 2^{18} \) (labeled 1k, 16k and 262k, respectively).

For the three proposed PITs, the size factor \((\alpha)\) of the BF structures is fixed at 8, implying that the number of cells \((m)\) in each BF structure is set to \( 8N \). Therefore, from (1), the optimal number of hash indexes \( k \) is 6. Pre-defined collision threshold \( k_{th} \) is set to 4. Because 2-bit counters are used in rCBFs and rCBFs with signatures, maximum count \( c_{max} \) is 3. We assume that the size of a value field in a cell of an FBF and rCBF is 8 bits, which indicates 254 kinds of incoming faces excluding the initial value and the maximum value.

We assume networks that allow a small number of lookup failures, such as false positives, false negatives, and indeterminables. PITs, particularly in edge routers, should support Interest aggregation to reduce Interest traffic overhead. Our proposed PITs are compared with fingerprint-only PIT (FO-PIT) using a \( d \)-left hash table with \( d = 2 \) [23]. A \( d \)-left hash table stores the information of an Interest packet in the bucket with the smallest number of loads. FO-PIT stores fingerprints (i.e., signatures) instead of content names and uses a bit vector to store the list of incoming faces.

| Set   | Requested content names \((n(S) = N)\) | Interest packets \((2.5N)\) | Data packets \((n(U) = 1.01N)\) |
|-------|------------------------------------|-----------------------------|---------------------------------|
| 1k    | \(2^{10}\)                         | \(2.5 \times 2^{10}\)      | \(1.01 \times 2^{10}\)         |
| 16k   | \(2^{14}\)                         | \(2.5 \times 2^{14}\)      | \(1.01 \times 2^{14}\)         |
| 262k  | \(2^{18}\)                         | \(2.5 \times 2^{18}\)      | \(1.01 \times 2^{18}\)         |

A. MEMORY REQUIREMENTS

The memory requirements of our proposed BF-based PITs are determined by the number of BF structures \((b)\), size factor \((\alpha)\), cell size, and number of requested contents \((N)\). The cell size of an FBF is \( L \) \(=\) \( 2 \), considering an \( L \)-bit value field and a 2-bit aging field. Hence, the memory requirement of the FBF-PIT, \( M_f \), is as follows.

\[
M_f = \alpha bN(L + 2).
\]  

In rCBF-PIT, each cell in rCBFs additionally has an \( R \)-bit counter, and hence the cell size of an rCBF is \( L + R + 2 \). The memory requirement of rCBF-PIT, \( M_r \), is as follows.

\[
M_r = \alpha bN(L + R + 2).
\]

In R-rCBF-PIT, \( G \)-bit signatures are additionally used in the first rCBF, where \( G \) depends on the number of requested contents \((N)\). Hence, the memory requirement of R-rCBF-PIT, \( M_s \), is calculated as follows.

\[
M_s = \alpha GN + \alpha bN(L + R + 2) = \alpha N\left(G + b(L + R + 2)\right).
\]

TABLE 1. Experimental sets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3158368, IEEE Access
In our simulation, FBF-PIT is first constructed by incrementally creating an FBF until \(2.5N\) Interest packets are stored in the FBF-PIT. To use the same amount of memory as the FBF-PIT, the number of rCBFs, \(b\), is adjusted in constructing the rCBF-PIT. The R-rCBF-PIT is then constructed using the same number of BF structures as the rCBF-PIT. Since R-rCBF-PIT has an additional field of the signature in the first BF only, it is inappropriate to adjust the number of rCBFs. Several parameters other than \(b\) could be adjusted to allocate the exact same amount of memory as FBF-PIT, but it can lead to confusion in understanding the performance evaluation. We think that using a slightly more amount of memory is a reasonable trade-off for better performance in other metrics such as the accuracy of a packet lookup.

Table 2 shows the number of BF structures (\(b\)) for three experimental sets. Because a cell of rCBFs consists of a counter field as well as a value field, the memory requirement of an rCBF is larger than that of an FBF with the same number of cells. Hence \(b\) of rCBF-PIT is smaller than that of FBF-PIT to build both architectures using the same amount of memory.

FO-PIT is also constructed using the same amount of memory as the FBF-PIT. The FO-PIT consists of \(d\) hash tables, and each bucket in the hash tables has 8 entries. Each entry comprises a 1-bit occupy field, a \(\log_2 N\)-bit fingerprint field (i.e., signature), a 10-bit expiration field, and a 254-bit face-list field (represented by a bit vector). To support Interest aggregation, a collision field is not used.

### Table 2. Number of BF structures (\(b\)) in three proposed PITs.

| Set | FBF-PIT | rCBF-PIT | R-rCBF-PIT |
|-----|---------|----------|------------|
| 1k  | 5       | 4        | 4          |
| 16k | 6       | 5        | 5          |
| 262k| 6       | 5        | 5          |

Table 3 presents the memory requirements of the three proposed BF-based PITs and the FO-PIT with \(d = 2\). Because R-rCBF-PIT occupies additional memory for signatures in the first BF structure, the memory requirement of R-rCBF-PIT is slightly larger than those of the other PITs. For R-rCBF-PIT, 4-, 5-, and 6-bit signatures are used for 1k, 16k, and 262k, and for FO-PIT, 10-, 14-, and 18-bit fingerprints are used for 1k, 16k, and 262k, respectively.

### B. INTEREST PACKET LOOKUP

Table 4 shows the comparison of the average \((n_A)\) and worst-case numbers \((n_{lw})\) of PIT accesses for Interest packets. For an Interest packet, the proposed PIT lookup algorithm inserts the information of the packet by accessing sequentially from the first BF structure. If no hash collision occurs, \(n_A\) for the proposed PITs is 2 (i.e., \(n_A = 5N/2.5N\)), because the number of the first BF accesses is \(N\), the number of the second BF accesses is \(0.75N\), the number of the third BF accesses is \(0.5N\), and the number of the fourth BF accesses is \(0.25N\). However, since hash collisions exist in BF structures, \(n_A\) of every proposed PIT is slightly larger than 2. The proposed PITs have the worst-case number of PIT accesses \((n_{lw})\) proportional to \(O(b)\), and hence \(n_{lw}\) for each PIT is the same as the number of BF structures \((b)\) shown in Table 2. In addition, \(n_{lw}\) for FO-PIT is \(d\) (i.e., 2), and \(n_A\) for FO-PIT is smaller than \(d\) because a matching entry for an Interest packet can exist in the first hash table.

### Table 3. Memory requirements of PITs (\(MB\)).

| Set  | FO-PIT  | FBF-PIT | rCBF-PIT | R-rCBF-PIT |
|------|---------|---------|----------|------------|
| 1k   | 0.049   | 0.049   | 0.047    | 0.051      |
| 16k  | 0.938   | 0.938   | 0.938    | 1.016      |
| 262k | 15.000  | 15.000  | 15.000   | 16.500     |

### Table 4. Comparison of average and worst-case numbers of PIT accesses for Interest packets.

| Set  | FO-PIT  | FBF-PIT | rCBF-PIT | R-rCBF-PIT |
|------|---------|---------|----------|------------|
| 1k   | 1.70    | 2.03    | 2.03     | 2.03       |
| 16k  | 1.70    | 2.03    | 2.11     | 2.11       |
| 262k | 1.70    | 2.03    | 2.11     | 2.11       |

Because the number of BF structures is limited in rCBF-PIT and R-rCBF-PIT in order to use the same amount of memory as with FBF-PIT, some of the Interest packets cannot be stored in the PITs, which is called overflows (\(O\)). An overflow results in a false negative (i.e., a false negative by an overflow, FNO), which means that a BF-based PIT cannot return matching faces when a Data packet corresponding to a requested content arrives. Table 5 presents the comparison of the overflow rates for the three experimental sets. The overflow rate \((P_o)\) is the number of overflows in the PIT, \(n(\cdot)\), divided by the total number of Interest packets, \(2.5N\). Hence, \(P_o = \frac{n(\cdot)}{2.5N}\). Because FBF-PIT is constructed to store all Interest packets, no overflows occur. However, since rCBF-PIT and R-rCBF-PIT are constructed based on the memory requirement of FBF-PIT, they have overflows. It is shown that the overflow rates are less than 1%. The PITs have the same overflow rate, because the condition for moving to the next BF structure (resulting from not being able to store an Interest packet in an accessed BF) is the same and the number of BF structures \((b)\) is the same. If there is no constraint on the maximum number of rCBFs, overflows will not occur.

The overflows of an FO-PIT occur when Interest packets are not inserted properly. In other words, the overflows of FO-PIT include two cases: the overflows in \(d\) buckets and the collisions resulting from both bucket collision and fingerprint collision (i.e., signature collision). It is shown that the overflow rates of the proposed PITs are smaller than those of the FO-PIT.

### C. DATA PACKET LOOKUP

For a Data packet, all BF structures in the proposed BF-based PIT are simultaneously looked up, and a delete operation is invoked in every BF structure returning a matching face. All BF structures in a BF-based PIT have the same number of
TABLE 5. Comparison of overflow rates (%).

| Set   | FO-PIT | FBF-PIT | rCBF-PIT | R-rCBF-PIT |
|-------|--------|---------|----------|------------|
| 1k    | 1.72   | 0       | 0.32     | 0.82       |
| 16k   | 0.74   | 0       | 0.25     | 0.25       |
| 262k  | 0.68   | 0       | 0.21     | 0.21       |

cells \((m)\) and the same number of hash indexes \((k)\); hence, the performance of the Data packet lookup of the proposed PIT lookup algorithm is proportional to \(O(1)\).

The result returned from each BF structure in the proposed PIT is classified as true positive (TP), true negative (TN), false positive (FP), false negative (FN), or indeterminable (INDET). For a Data packet with name \(x\), TP indicates that a BF structure in PIT returns a true face, from which an Interest packet with \(x\) has arrived. TN indicates that a BF structure does not return any matching face (i.e., the structure has never stored the incoming face corresponding to \(x\)). FP indicates that \(x\) has never been stored in a BF structure but the structure returns a face owing to hash collision. An FP causes an incorrect deletion resulting in FNs. An FN is returned by two reasons: a false negative by an overflow (FNO) or a false negative by an incorrect deletion (FND). INDET indicates that a BF structure fails to return a face.

Table 6 presents the lookup failure rates of the proposed BF-based PITs for Data packets. Because the total number of Data packets is \(1.01N\), the total number of returned results for all the Data packets is \(1.01N \times b\). Hence, the rates of TP, TN, FP, FN, and INDET are the number of each event divided by \(1.01N \times b\), respectively. The FNO of FBF-PIT is zero because FBF-PIT uses the minimum number of FBFs that enables no overflow. In rCBF-PIT and R-rCBF-PIT, although overflows occur because of limited memory, it is shown that the FNO rate is less than 1% owing to the overflow rates of the PITs being less than 1% from Table 5. If \(b\) of rCBF-PIT and R-rCBF-PIT is not limited, the rates of FNO will be zero. Compared to FBF-PIT, it is shown that rCBF-PIT and R-rCBF-PIT have lower INDET rates, because rCBFs allow the reuse of cells storing two or more faces, while FBFs cannot reuse conflict cells.

In rCBF-PIT and R-rCBF-PIT, faces are more evenly distributed to multiple BF structures than in FBF-PIT because the use of counters prevents more than \(c_{\text{max}}\) faces from being stored in a single cell. Hence, for 16k and 262k, the rates of FP and FND (resulting from FP) are lower than those in FBF-PIT. Especially for 16k, the rates of FP and FND in rCBF-PIT and R-rCBF-PIT are zero. Because a false positive in rCBF-PIT occurs only when both hash index collision and signature collision occur, R-rCBF-PIT solves the FP and FND by using signatures in the first rCBF. In other words, since the largest number of Interest packets are stored in the first rCBF in rCBF-PIT (approximately 37% of all Interest packets), adding signatures to the first rCBF can effectively remove all FPs and FNDs. It is shown that the lookup failure rates of the three PITs are less than 1%, regardless of \(N\).

Figure 12 shows the comparison of the lookup failure rates of the proposed BF-based PITs and FO-PIT for Data packets.

Note that the lookup failure rates in Figure 12 are calculated based on the number of Data packets (i.e., \(1.01N\)). In other words, for a Data packet, if several faces are returned and one of them is false, the result of the packet lookup is classified as a lookup failure, while for a Data packet, each of the \(b\) results returned from the BF structures is classified as a lookup failure or a true result in Table 6 even though the proposed PITs simultaneously return the \(b\) results. Table 6 shows accurate performance evaluations of the proposed PITs; however, the lookup failure rates in Figure 12 should be calculated based on \(1.01N\) to fairly compare the lookup failure rates for different data structures (i.e., BF structures and \(d\)-left hash table). The lookup failure rates of rCBF-PIT and R-rCBF-PIT are lower than those of FO-PIT.

It is interesting to note that the lookup failure rate of FBF-PIT is the smallest among the PITs for 1k because of the insufficient number of rCBFs in rCBF-PIT and R-rCBF-PIT (i.e., \(b = 4\)) for 1k. However, as the number of Interest packets increases, \(b\) of FBF-PIT increases, as shown in Table 2; hence, \(b\) of the other proposed PITs also increases. Therefore, for 16k and 262k, the lookup failure rates of rCBF-PIT and R-rCBF-PIT are lower than those of FBF-PIT.

D. COMPARISON WITH OTHER ALGORITHMS

Our proposed PITs have been compared with FO-PIT in previous subsections because they have common characteristics: supporting Interest aggregation and being stored in on-chip memory. In this subsection, we discuss and compare the characteristics of other BF-based PITs which cannot be quantitatively compared with our proposed PITs, because each PIT has its own unique approach, different from the proposed.

DiPIT [24] is a space-efficient PIT that uses a CBF per face. When an Interest packet arrives from a face, the information of the Interest packet is stored in the CBF of the arriving face, and hence the information can be stored with a single access (i.e., \(O(1)\)). However, when a Data packet arrives, all CBFs should be queried to find every matching face (i.e., \(O(l)\), where \(l\) is the number of faces).
TABLE 6. Lookup failure rates of three proposed PITs for Data packets (%).

| Set       | True results (TR) | Lookup failures (LF) |
|-----------|-------------------|----------------------|
|           | TP                | TN                   | Total     |
| FBF-PIT   |                   |                      |           |
| 1k        | 49.15             | 50.48                | 99.63     |
| 16k       | 40.99             | 58.74                | 99.73     |
| 262k      | 41.00             | 58.75                | 99.75     |
| rCBF-PIT  |                   |                      |           |
| 1k        | 61.32             | 38.10                | 99.42     |
| 16k       | 49.33             | 50.50                | 99.83     |
| 262k      | 49.35             | 50.50                | 99.85     |
| R-rCBF-PIT|                   |                      |           |
| 1k        | 61.32             | 38.10                | 99.42     |
| 16k       | 49.33             | 50.50                | 99.83     |
| 262k      | 49.35             | 50.50                | 99.85     |

Thus, the performance of DiPIT for Data packets depends on the number of faces. Moreover, because the DiPIT does not support the Interest aggregation, unnecessary traffic would be created by forwarding the same content requests repeatedly. On the other hand, our proposed PITs support the Interest aggregation and result in reduced Interest traffic overhead.

In our experiment, DiPIT is constructed using the same amount of memory as the FBF-PIT for three experimental sets in Table 1. For a Data packet, 254 on-chip memory accesses are required because every CBF in 254 faces should be queried to find all matching faces, while our PITs can return all matching faces with a single access because all BF structures are looked up in parallel. In addition, our PITs can reduce Interest traffic overhead by supporting Interest aggregation.

The space-efficient DiPIT does not have a lookup failure when using the same amount of memory as the FBF-PIT in the experiment. DiPIT keeps an advantage in terms of memory space and lookup failures when Interest packets come in evenly from all faces as in the experiment because of using a CBF per face. However, if Interest packets are flooded from a few faces, DiPIT may have a number of lookup failures. By contrast, our proposed BF-based PITs can process the packets flooded from a few faces, because a BF structure in the PITs can store packets from all faces and a new BF structure is dynamically allocated when hash index collisions exceed a pre-defined collision threshold.

MaPIT [26] stores a Bloom filter and a mapping array (MA) in on-chip memory, and packet store (PS) and a CBF in off-chip memory, while our proposed PITs store b BF structures in on-chip memory. MaPIT should access off-chip memory for most of the Interest and Data packets to store an incoming face or find all matching faces. Moreover, MaPIT requires a large amount of off-chip memory, even though most of the entries in PS are not used because PS is accessed using the MA as the offset address of PS. Assuming that the number of hash indexes (k) of the BF is 12 and the MA consists of 30 bits as in [26], the number of entries in the PS is 2^{30} and the number of entries that can never be accessed is 30C_0 + \sum_{i=1}^{30} 30C_i = 1 + \sum_{i=13}^{30} 30C_i. Therefore, a large amount of off-chip memory is wasted in MaPIT.

In our experiment, MaPIT is constructed using the same memory size as that in [26] because MaPIT uses both on-chip and off-chip memories. For every Interest packet, the BF in on-chip memory is programmed, the CBF in off-chip memory is programmed, and then PS is accessed by using the MA to store the incoming face of the packet. Hence, the number of on-chip memory accesses is one (i.e., BF) and the number of off-chip memory accesses is two (i.e., CBF and PS).

For a Data packet, the BF in on-chip memory is first queried. If the BF returns a negative, Data packet lookup is over, while if the BF returns a positive, the PS is accessed. If there is no matching entry in the PS, the lookup is over because the positive is false. Otherwise, if a matching entry exists in the PS, the packet is forwarded through matching faces, the information of the packet is then deleted from the CBF and the PS, and the BF is synchronized with the CBF.

In case of MaPIT using the same amount of memory as the FBF-PIT, assuming that an 8N-cell BF and an 8N-cell CBF are used with k = 6, MAs are 10, 15, and 19 bits, and the numbers of entries in the PSs are 2^{10}, 2^{15}, and 2^{19}, for 1k, 16k, and 262k, respectively. Considering the entries that can never be accessed, the load factor of the PS in MaPIT is calculated as the number of entries that can be accessed divided by the number of total entries minus the number of entries that can never be accessed. In the experiment, the load factors of the PSs for three sets are 1.21, 1.65, and 5.99, respectively, and the lookup failure rates of MaPIT for 1.01N Data packets are 84.3, 88.5, and 98.5%, respectively. Hence, it is not appropriate to use MaPIT using the same amount of memory as the FBF-PIT.
to compare MaPIT and the proposed PITs, because MaPIT uses both on-chip and off-chip memories. MaPIT is suitable for a PIT which requires to minimize the on-chip memory consumption.

VI. CONCLUSION

This paper proposes three Bloom filter-based PIT architectures that enable a fast PIT lookup with a low memory cost. In Interest packet processing, the proposed PIT architecture dynamically allocates memory by incrementally creating more Bloom filter structures. In a Data packet lookup, our proposed PIT architectures simultaneously return and delete all matching faces with a single access. We also provide entry expiration using two bits in a cell.

The simulation results show that the lookup failure rates for the Data packets of the proposed BF-based PITs are less than 1%. Moreover, the rCBF-based PIT architectures reduce the number of indeterminables by more than 81% compared to FBF-PIT. Even though R-rCBF-PIT cannot avoid false negatives by overflow (FNO), the results also show that R-rCBF-PIT reduces false negatives caused by incorrect deletions (FND) because false positives causing incorrect deletions are mostly avoided by the signature field in the first BF.

REFERENCES

[1] L. Zhang, A. Afanaseyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named Data Networking,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, Jul. 2014.
[2] V. Jacobson, K. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, “Networking Named Content,” ACM CoNEXT, 2009, pp. 1–12.
[3] A. Vasilakos, Z. Li, G. Simon, and W. You, “Information centric network: research challenges and opportunities,” Journal of Network and Computer Applications, vol. 52, pp. 1–10, Jun. 2015.
[4] H. Zhang, R. Xie, S. Zhu, T. Huang, and Y. Liu, “DENA: An Intelligent Content Discovery System Used in Named Data Networking,” IEEE Access, vol. 4, pp. 9093–9107, Dec. 2016.
[5] T. Zhi, Y. Liu and J. Wu, “A Reputation Value-Based Early Detection Mechanism Against the Consumer-Provider Collusive Attack in Information-Centric IoT,” IEEE Access, vol. 8, pp. 38262–38275, Feb. 2020.
[6] M. Zhang, V. Lehman, and L. Wang, “Scalable Name-based Data Synchronization for Named Data Networking,” IEEE INFOCOM 2017, Atlanta, GA, USA, May 2017.
[7] Z. Li, Y. Xu, B. Zhang, L. Yan, and K. Liu, “Packet Forwarding in Named Data Networking Requirements and Survey of Solutions,” IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1950–1987, Second quarter 2019.
[8] Md. Bari, S. Chowdhury, R. Ahmed, and B. Mathieu, “A survey of naming and routing in information-centric networks,” IEEE Communications Magazine, vol. 50, no. 12, Dec. 2012.
[9] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo, “Recent Advances in Information-Centric Networking-Based Internet of Things (ICN-IoT),” IEEE Internet of Things Journal, vol. 6, no. 2, Apr. 2019.
[10] Z. Li, Y. Xu, B. Zhang, L. Yan, and K. Liu, “Enabling Push-Based Critical Data Forwarding in Vehicular Named Data Networks,” IEEE Communications Letters, vol. 21, no. 4, pp. 873–876, Dec. 2016.
[11] T. Pan, X. Lin, E. Song, C. Xu, J. Zhang, H. Li, J. Lv, T. Huang, B. Liu, and B. Zhang, “NB-Cache: Non-Blocking In-Network Caching for High-Performance Content Routers,” IEEE/ACM Transactions on Networking, Early Access, pp. 1–14, Jun. 2021.
[12] J. Mun and H. Lim, “Cache sharing using Bloom filters in named data networking,” Journal of Network and Computer Applications, vol. 90, pp. 74–82, May 2017.
[13] C. Chen, C. Wang, T. Qiu, M. Atiquzzaman, and D. Wu, “Caching in Vehicular Named Data Networking: Architecture, Schemes and Future Directions,” IEEE Communications Surveys & Tutorials, vol. 22, no. 4, pp. 2378–2407, Fourthquarter 2020.
[14] M. Naeem, M. Rehman, R. Ullah, and B. Kim, “A Comparative Performance Analysis of Popularity-Based Caching Strategies in Named Data Networking,” IEEE Access, vol. 8, pp. 50057–50077, Mar. 2020.
[15] R. Hou, L. Zhang, T. Wu, T. Mao, and J. Luo, “Bloom-filter-based request node collaboration caching for named data networking, Cluster Computing,” Cluster Computing, vol. 22, no. 3, pp. 6681–6692, May 2019.
[16] L. Fan, S. Hassan, M. Khan, M. Guizani, O. Ghazali, and A. Hobal, “Caching in Information-Centric Networking: Strategies, Challenges, and Future Research Directions,” IEEE Communications Surveys & Tutorials, vol. 20, no. 2, pp. 1443–1474, Secondquarter 2018.
[17] H. Dai, J. Lu, Y. Wang, T. Pan, and B. Liu, “BFAST: High-Speed and Memory-Efficient Approach for NDN Forwarding Engine,” IEEE/ACM Transactions on Networking, vol. 25, no. 2, pp. 1235–1248, Apr. 2017.
[18] J. Lee, M. Shim, and H. Lim, “Name prefix bloom filter pre-searching for content centric network,” Journal of Network and Computer Applications, vol. 65, pp. 36–47, Apr. 2016.
[19] H. Byun and H. Lim, “A New Bloom Filter Architecture for FIB Lookup in Named Data Networking,” Applied Sciences, vol. 9, no. 2, pp.329–345, Jan. 2019.
[20] O. Karrakchou, N. Samaan, A. Karmouch, “FCTrees: A Front-Coded Family of Compressed Tree-Based FIB Structures for NDN Routers,” IEEE Transactions on Network and Service Management, vol. 17, no. 2, pp. 1167–1180, Jun. 2020.
[21] J. Takemasa, Y. Koizumi, and T. Hasegawa, “Data prefetch for fast NDN software routers based on hash table-based forwarding tables,” Computer Networks, vol.173, no. 22, May 2020.
[22] F. Berto, L. Calderoni, M. Conti, and E. Losiouk, “Spatial Bloom filter in named data networking; a memory efficient solution,” SAC ’20:Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, pp.274–277, Mar. 2020.
[23] H. Yuan and P. Crowley, “Scalable Pending Interest Table design: From principles to practice,” IEEE INFOCOM 2014, Toronto, ON, Canada, pp.329–345, Apr./May 2014.
[24] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, “DiPIT: A Distributed Bloom-Filter Based PIT Table for CCN Nodes,” 21st International Conference on Computer Communications and Networks (ICCCN), Munich, Germany, Jul/Aug. 2012.
[25] M. Varvello, D. Perino, and L. Linguaglossa, “On the design and implementation of a wire-speed pending interest table,” IEEE INFOCOM WKSHPS, Turin, Italy, Apr. 2013.
[26] Z. Li, K. Liu, Y. Zhao, and Y. Ma, “MaPIT: An Enhanced Pending Interest Table for NDN With Mapping Bloom Filter,” IEEE Communications Letters, vol. 18, no. 11, pp.1915–1918, Nov. 2014.
[27] D. Saxena and V. Raychoudhury, “Radiant: Scalable, memory efficient name lookup algorithm for named data networking,” Journal of Network and Computer Applications, vol. 63, pp. 1–13, Mar. 2016.
[28] R. Shubbar and M. Ahmadi, “A Filter-based Design of Pending Interest Table in Named Data Networking,” Journal of Network and Systems Management, vol. 27, pp. 998–1019, Mar. 2019.
[29] B. Bloom, “Space/time tradeoffs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422–426, July 1970.
[30] A. Z. Broder and M. Mitzenmacher, “Network applications of Bloom filters: A survey,” Internet Mathematics, vol. 1, no. 4, 2003.
[31] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, “Theory and practice of Bloom filters for distributed systems,” IEEE Communications Surveys & Tutorials, vol. 14, no. 1, pp. 131–153, 1st Quarter, 2012.
[32] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: a scalable wide-area Web cache sharing protocol,” IEEE/ACM Transactions on Networking, vol. 8, no. 3, pp. 281–293, Jun. 2000.
[33] H. Lim, J. Lee, H. Byun, and C. Yim, “Ternary Bloom Filter Replacing Counting Bloom Filter,” IEEE Communications Letters, vol. 21, no. 2, pp. 278–281, Feb. 2017.
[34] F. Bonomi, M. Mitzenmacher, R. Panigrahi, S. Singh, and G. Varghese, “Beyond Bloom filters: From approximate membership checks to approximate state machines,” ACM SIGCOMM Computer Communication Review, vol. 36, no. 4, pp. 315–326, Oct. 2006.
[35] H. Byun, S. Kim, C. Yim, and H. Lim, “Addition of a secondary functional Bloom filter,” IEEE Communications Letters, vol. 24, no. 10, pp. 2123–2127, Oct. 2020.
[36] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “An improved construction for counting Bloom filters,” Proc. ESA, vol. 14, pp. 684–695, 2006.

[37] H. Byun and H. Lim, “Comparison on search failure between hash tables and a functional Bloom filter,” Applied Sciences, vol. 10, no. 15, pp. 5218–5235, Jul. 2020.

[38] H. Byun, Q. Li, and H. Lim, “Vectored-Bloom filter for IP address lookup: Algorithm and hardware architectures,” Applied Sciences, vol. 9, no. 21, pp. 4621–4839, Oct. 2019.

[39] K. Tukhinbekov and D. Kim, “CaseDB: Lightweight Key-Value Store for Edge Computing Environment,” IEEE Access, vol. 8, pp. 149775–149786, Aug. 2020.

[40] S. Xiong, Y. Yao, Q. Cao, and T. He, “kBF: A Bloom filter for key-value storage with an application on approximate state machines,” IEEE INFOCOM 2014, Toronto, ON, Canada, Jul. 2014.

[41] S. Jang, H. Byun, and H. Lim, “PIT Lookup Using Functional Bloom Filters,” IEIE Summer Conference, Aug. 2020, (with Korean).

[42] M. Kim, S. Kim, H. Byun, and H. Lim, “Counting Bloom Filter with Return Values for Dynamic Data,” IEIE Summer Conference, Jun. 2021 (with Korean).

[43] R. Seifert, “The Switch Book: The Complete Guide to LAN Switching Technology” John Wiley Sons, Inc., 2000.

[44] Alexa the Web Information Company. [Online]. Available: http://www.alexa.com/

SAEYOUNG JANG (Member, IEEE) received the B.S. and M.S. degree at the Department of Electronics Engineering, Ewha Womans University, Seoul, Korea, in 2019 and 2021.

HAYOUNG BYUN (Member, IEEE) received the B.S. degree at the Department of Electronics Engineering, Ewha Womans University, Seoul, Korea, in 2014. She received the Ph.D. degree at the Department of Electronic and Electrical Engineering from Ewha Womans University in 2020. She also studied Artificial Intelligence (AI) and Machine Learning (ML) at Carnegie Mellon University, Pittsburgh, PA in 2019. She is an assistant professor at the Department of Electronics Engineering, Myongji University, Seoul, Korea, where she does research on networking, machine learning, and design and analysis of algorithms.

HYEESOOK LIM (Senior Member, IEEE) received the B.S. and M.S. degrees at the Department of Control and Instrumentation Engineering in Seoul National University, Seoul, Korea, in 1986 and 1991, respectively. She received the Ph.D. degree at Electrical and Computer Engineering from the University of Texas at Austin in 1996. From 1996 to 2000, she was a member of technical staff at Bell Labs in Lucent Technologies, Murray Hill, NJ. From 2000 to 2002, she worked as a hardware engineer for Cisco Systems, San Jose, CA. Currently, she is a professor in Electronic and Electrical Engineering at Ewha Womans University, Seoul, Korea from Mar. 2002, where her research is on packet forwarding algorithms and hardware implementation of various network algorithms. She was the Dean of the College of Engineering from Feb. 2018 to Jan. 2020. She was also the President of the Institute of Electronics and Information Engineers in 2020, which is the largest association of IT engineers in Korea. She holds the position of the Minister of Science and ICT of the Republic of Korea from May 14, 2021. She is also a member of the National Academy of Engineering of Korea.