Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080445

Reducing False Positives of a Bloom Filter using Cross-Checking Bloom Filters

Hyesook Lim[1], *Nara Lee*[1], *Jungwon Lee*[1] *and Changhoon Yim*[2,*]

[1] Department of Electronics Engineering, Ewha W. University, Seoul, Korea
[2] Department of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea

Received: 14 Aug. 2013, Revised: 16 Nov. 2013, Accepted: 17 Nov. 2013
Published online: 1 Jul. 2014

Abstract: A Bloom filter is a compact data structure that supports membership queries on a set, allowing false positives. The simplicity and the excellent performance of a Bloom filter make it a standard data structure of great use in many network applications. In reducing the false positive rate of a Bloom filter, it is well known that the size of a Bloom filter and accordingly the number of hash indices should be increased. In this paper, we propose a new architecture reducing the false positive rate of a Bloom filter more efficiently. The proposed architecture uses cross-checking Bloom filters that are queried in case of positives of a main Bloom filter to cross-check the results. If every cross-checking Bloom filters produce negatives, the positive of the main Bloom filter can be determined as a false positive. The main Bloom filter is not necessarily large to reduce the false positive rate, since more numbers of the false positives of the main Bloom filter are identified by cross-checking Bloom filters. Simulation results show that the false positive of the proposed scheme converges to zero faster, while requiring the total memory size for Bloom filters smaller, than that of a single Bloom filter architecture.

Keywords: Internet, router, network, algorithm, Bloom filter, false positive, cross-checking Bloom filters.

1. Introduction

The amount of data handled by Internet users is growing fast in recent years, and accordingly, the amount of traffic has been significantly increased [1]–[3]. In order to handle a large amount of data safely and efficiently, filtering out unnecessary data is useful in many applications. A Bloom filter is a simple and efficient data structure identifying whether an input is a member of a given set. It is used in filtering out inputs not included in a set. Bloom filters have been used in many applications since its emergence in 1970 [4]. Recently they are popularly used in networking areas such as IP address lookup [5]–[8], packet classification [9] traffic measurement [10], [11], peer-to-peer systems [12], [13], firewall [14], intrusion detection [15]–[17], and web caching [18]–[20]. Therefore, a small improvement related to a Bloom filter can give a large impact on many applications.

A Bloom filter has an intrinsic problem of false positives, which identifies an input as a member even though the input is not actually a member of the set. It is well-known that the false positive rate can be controlled by increasing the size of a Bloom filter and the number of hash indices. However, when a given set is large, increasing the size of a Bloom filter is limited by the required memory amount since Bloom filters are usually implemented using an on-chip memory for fast processing.

Researches to improve the performance of Bloom filters have been carried out in [21]–[24]. It is not easy to improve the performance, while maintaining the processing simplicity and the storage efficiency of a Bloom filter. In a dual Bloom filter structure [22], two Bloom filters are serially connected. Both Bloom filters are programmed for the same set of elements using different hash functions. The second Bloom filter is queried only when the first Bloom filter produces a positive. If the second Bloom filter produces a negative, then the positive of the first Bloom filter is a false positive, since both Bloom filters were programmed using the same set. In order to reduce the hardware complexity caused by using different hash functions, the dual Bloom filter structure uses one's complemented values of the elements in programming the second Bloom filter. Since both Bloom filters are programmed by the entire elements

* Corresponding author e-mail: cyim@konkuk.ac.kr

of a given set, this structure does not provide a flexibility in required memory amount for implementing Bloom filters.

This paper is motivated to improve the performance of a Bloom filter by reducing the false positive rate and still maintain the processing simplicity and the storage efficiency. Our proposed approach is to use cross-checking Bloom filters, each of which is programmed for a subset of the elements and queried when the main Bloom filter produces a positive. In our proposed approach, since more numbers of false positives can be identified by the cross-checking Bloom filters, the main Bloom filter does not have to be large enough, and hence the required memory size for Bloom filters is more flexible. Moreover, since elements can be grouped depending on their characteristics such as priority in programming into cross-checking Bloom filters, the characteristics as well as the membership of inputs can be identified by finding out which cross-checking Bloom filter produces a positive result for the input.

This paper is organized as follows. Section 2 describes a Bloom filter with the detailed description of the false positive. Section 3 proposes a new architecture to reduce the false positive of a Bloom filter. In Section 4, simulation result and the performance comparison with related works are shown. Section 5 gives a brief conclusion.

2. Bloom Filter

A Bloom filter is a bit-vector based data structure used in storing and identifying the membership of each element of a set. It has two operations; programming and querying. Let U represent the universe set. A Bloom filter is used to represent a set $S = \{x_1, x_2, \cdots, x_n\}$ of n elements from the universe U [25]. Let $H = \{h_1, h_2, \cdots, h_k\}$ be the set of hash functions, where k is the number of hash functions. The m-bit Bloom filter (BF) is initialized as zero. The programming procedure for a set S with n elements is shown as follows [5].

```
BFProgramming
    for ( j = 1 to n )
        for ( i = 1 to k )
            BF[h_i(x_j)] = 1;
```

For $x_j \in S$, $j = 1, \cdots, n$, $h_i(x_j)$ for $i = 1, \cdots, k$ are obtained, where $0 \le h_i(x_j) < m$, and the corresponding bits to the hash indices, BF$[h_i(x_j)]$, are set for the element.

The procedure of querying whether an input y is the member of the set S is as follows [5].

```
BFQuery(y)
    for (i = 1 to k)
        if (BF[h_i(y)] == 0) return negative;
    return positive;
```

For an input y, k hash indices, $h_i(y)$ for $i = 1, \cdots, k$, are obtained, where $0 \le h_i(y) < m$. If at least a bit of the Bloom filter corresponding to the indices, BF$[h_i(y)]$, is 0, the input is not a member of the set and it is called *negative*. If every bit of the Bloom filter corresponding to the hashed indices is 1, the input is a member of the set with a high probability. This is called *positive*.

An intrinsic problem of a Bloom filter is that it may produce positive even though an input is not a member of the set, and it is called *false positive*. The false positive is generated since hashing is used in programming and querying a Bloom filter, and the hashing does not guarantee one-to-one mapping between an element and a group of indices. In other words, an index can be mapped by multiple elements. Therefore, for an input, even though all queried bits were 1, it cannot be sure that every bit was programmed by the queried input. Even though a hash function making one-to-one correspondence between each element of a set and a bit of a Bloom filter is assumed in programming a Bloom filter, the universe set of inputs used in membership query can be much larger or it can be infinite in worst-case. Since the space of a Bloom filter is finite, it is impossible to have a Bloom filter which does not have a false positive.

The false positive of Bloom filters gives negative effects on overall system performance. Many network systems perform a next step of processing in case that an input is a member of a set by accessing an off-chip hash table storing the elements of the set [26]. Accessing the off-chip hash table takes longer processing time. Hence it is important to reduce the false positive rate of a Bloom filter and accordingly the number of off-chip hash table accesses for improving the overall system performance.

To discuss the false positive of a Bloom filter in detail, let $A = \{a_1, a_2, \cdots, a_m\}$ be the set of addresses, where m is the number of addresses (bits) in a Bloom filter. For $x \in S$, $h_i(x)$ is the hashing indices of x and $h_i(x) \in A$. Let $V(h_i(x))$ represent the value at the indices $h_i(x)$, and $V(h_i(x)) \in \{1, 0\}$. Initial values of $V(h_i(x))$ are all 0. In programming, if x is mapped to $h_i(x)$ by hash function h_i, then $V(h_i(x))$ is set to 1.

Let p represent the probability that a specific bit is still 0 after all elements of S are hashed into the Bloom filter by k hash functions. If hash functions are assumed to be perfectly random, it can be calculated as [5], [25]

$$p = (1 - \frac{1}{m})^{kn}. \quad (1)$$

For large values of m, it can be approximated as

$$p \approx e^{-kn/m}. \quad (2)$$

If $V(h_i(x)) = 1$ for all $i = 1, \cdots, k$, x is a positive and is accepted as a candidate of $x \in S$ by the Bloom filter. Let $\mathscr{F}(S)$ be the set of x such that $V(h_i(x)) = 1$ for all $i = 1, \cdots, k$, where the Bloom filter was programmed by the set S. We call $\mathscr{F}(S)$ as the set of positives or the set of candidates of S by the Bloom filter.

If $x \in S$ is queried, then $V(h_i(x)) = 1$ for all $i = 1, \cdots, k$, and $x \in \mathscr{F}(S)$, since the Bloom filter was programmed by the set S. Hence $S \subset \mathscr{F}(S)$. There might be some $x \in U$ and $x \notin S$ such that $V(h_i(x)) = 1$ for all $i = 1, \cdots, k$. In this case, $x \in \mathscr{F}(S)$ and $x \notin S$, i.e., $x \in \mathscr{F}(S) - S$. We call the set $\mathscr{F}(S) - S$ as the false positive set of S. Note that the false positive set cannot be identified just by querying a Bloom filter.

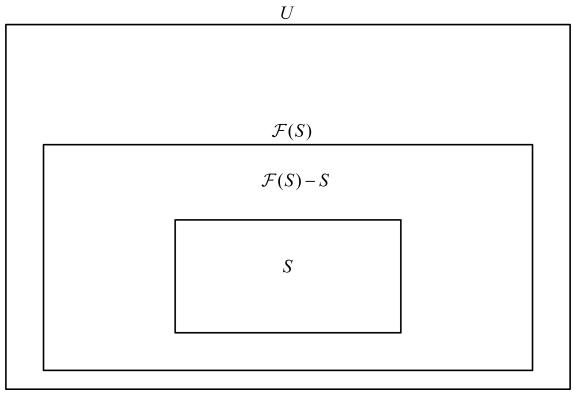

Figure 1 Set diagram for $\mathscr{F}(S)$ including S and $\mathscr{F}(S) - S$.

The set diagram for $\mathscr{F}(S)$ including S is shown in Fig. 1. In Fig. 1, $\mathscr{F}(S) = S \cup (\mathscr{F}(S) - S)$ and $S \cap (\mathscr{F}(S) - S) = \emptyset$. For an element x that is not in S ($x \notin S$), the probability f_S that $V(h_i(x)) = 1$ for all $i = 1, \cdots, k$ can be calculated as [5], [25]

$$f_S = (1-p)^k = (1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-kn/m})^k. \quad (3)$$

The probability f_S in (3) is termed as the false positive probability in [5], [25]. Note that the probability f_S is calculated with the assumption $x \notin S$. Let $P(\cdot)$ represent the probability and S^c represent the complement set of S. We define the probability f_S as the conditional false positive probability given S^c as

$$f_S = P(\mathscr{F}(S)|S^c) \quad (4)$$

which is the probability that x is included in $\mathscr{F}(S)$ given $x \in S^c$.

If the conditional false positive probability f_S is minimized with respect to k, then $k = (m/n) \cdot (\ln 2)$ [5], [25]. In this case, f_S can be approximated as [5], [25]

$$f_S = (1/2)^k \approx (0.6185)^{(m/n)}. \quad (5)$$

In (5), if $m \to \infty$, then $f_S \to 0$. Hence if m approaches to infinity, then the false positive set, $\mathscr{F}(S) - S$, approaches to an empty set. In other words, as the Bloom filter size m becomes large, the set $\mathscr{F}(S)$ shrinks and converges to the set S.

Now we consider the false positive probability without assuming $x \in S^c$. $P(S)$ is the probability of S in the universe set U with $0 \le P(S) \le P(U) = 1$. The probability of $\mathscr{F}(S)$, which is $P(\mathscr{F}(S))$, can be formulated as

$$P(\mathscr{F}(S)) = P(S)P(\mathscr{F}(S)|S) + P(S^c)P(\mathscr{F}(S)|S^c). \quad (6)$$

Since $S \subset \mathscr{F}(S)$,

$$P(\mathscr{F}(S)|S) = \frac{P(\mathscr{F}(S) \cap S)}{P(S)} = \frac{P(S)}{P(S)} = 1. \quad (7)$$

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

$$P(\mathscr{F}(S)) = P(S) + (1 - P(S))\, f_S. \quad (8)$$

The $P(\mathscr{F}(S))$ in (8) represents the probability that all $V(h_i(x)) = 1$ for all $i = 1, \cdots, k$ for an element $x \in U$.

Since $\mathscr{F}(S) = S \cup (\mathscr{F}(S) - S)$ and $S \cap (\mathscr{F}(S) - S) = \emptyset$,

$$P(\mathscr{F}(S) - S) = P(\mathscr{F}(S)) - P(S). \quad (9)$$

From (8) and (9),

$$\begin{aligned} P(\mathscr{F}(S) - S) &= P(S) + (1 - P(S))\, f_S - P(S) \\ &= (1 - P(S))\, f_S. \end{aligned} \quad (10)$$

The probability $P(\mathscr{F}(S) - S)$ in (10) represents the false positive probability of S for an element $x \in U$. Note that the false positive probability $P(\mathscr{F}(S) - S)$ in (10) is formulated without the condition of S^c. If we consider the false positive probability with the condition of S^c ($x \notin S$), then $P(S) = 0$ and the false positive probability $P(\mathscr{F}(S) - S)$ in (10) is equal to the conditional false positive probability f_S in (3) and (4).

3. Proposed Architecture

In this section, we propose a new architecture reducing the rate of false positive while maintaining the simplicity and the storage efficiency of a Bloom filter. The proposed architecture uses cross-checking Bloom filters to identify the false positive of a main Bloom filter.

As an example of the proposed architecture, we consider the set S in U as the union of two disjoint sets A and B as $S = A \cup B$, $A \cap B = \emptyset$. In other words, the example structure of the proposed architecture has two cross-checking Bloom filters as shown in Fig. 2. There is no constraint in the number of cross-checking Bloom filters, and we discuss here the case that there are two cross-checking Bloom filters for simplicity. In Figure 2, the first Bloom filter is the main Bloom filter which is programmed for the entire elements of a given set S. The second and the third Bloom filters are the cross-checking Bloom filters, and they are programmed for a subset A and B, respectively. The cross-checking filters are queried only when the main filter produces positive. Our proposed

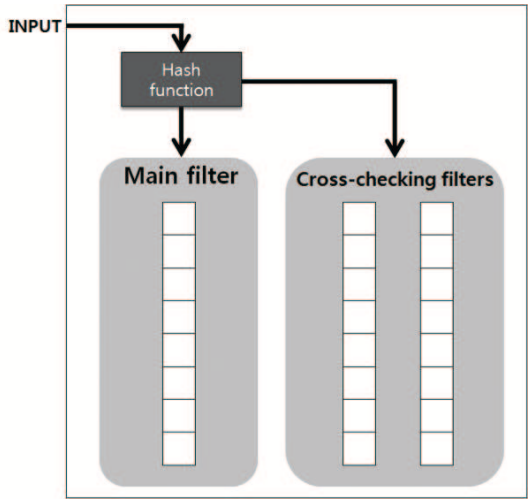

Figure 2 The proposed architecture with two cross-checking Bloom filters.

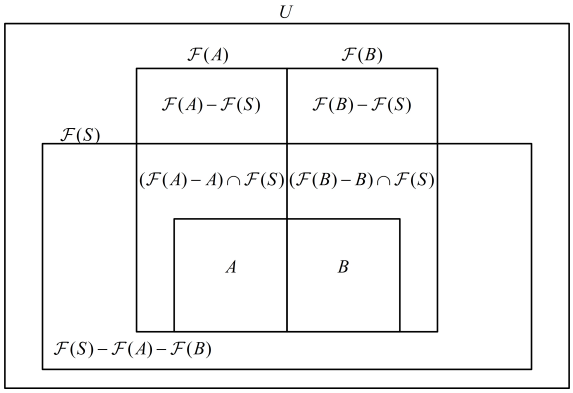

Figure 3 Set diagram with $\mathcal{F}(A)$, $\mathcal{F}(B)$, and $\mathcal{F}(S)$.

idea is that since $S = A \cup B$, if both cross-checking filters produce negatives, the positive of the main filter is a false positive, since there is no false negative in Bloom filters.

We describe the false positive of our proposed architecture as follows. The set diagram with $\mathcal{F}(A)$, $\mathcal{F}(B)$, and $\mathcal{F}(S)$ is represented in Fig. 3 as an extension of Fig. 1. The set $\mathcal{F}(A)$ can be decomposed into two disjoint sets A and $\mathcal{F}(A) - A$. The set $\mathcal{F}(A) - A$ can be decomposed further into two disjoint sets as $(\mathcal{F}(A) - A) \cap \mathcal{F}(S)$ and $(\mathcal{F}(A) - A) \cap (\mathcal{F}(S))^c$. Since $A \subset S \subset \mathcal{F}(S)$, $(\mathcal{F}(S))^c \subset A^c$. Hence the set $(\mathcal{F}(A) - A) \cap (\mathcal{F}(S))^c$ can be simplified as $(\mathcal{F}(A) \cap A^c) \cap (\mathcal{F}(S))^c = \mathcal{F}(A) \cap (\mathcal{F}(S))^c = \mathcal{F}(A) - \mathcal{F}(S)$. Hence the set $\mathcal{F}(A)$ is decomposed into three disjoint sets A, $(\mathcal{F}(A) - A) \cap \mathcal{F}(S)$, and $\mathcal{F}(A) - \mathcal{F}(S)$ as in Fig. 3. Similarly, the set $\mathcal{F}(B)$ is decomposed into three disjoint sets B, $(\mathcal{F}(B) - B) \cap \mathcal{F}(S)$, and $\mathcal{F}(B) - \mathcal{F}(S)$.

In the proposed architecture, if $x \in (\mathcal{F}(S))^c$, then it is firstly filtered out by the main Bloom filter and is not queried to the cross-checking Bloom filters. Since $(\mathcal{F}(A) - \mathcal{F}(S)) \subset (\mathcal{F}(S))^c$ and $(\mathcal{F}(B) - \mathcal{F}(S)) \subset (\mathcal{F}(S))^c$, inputs in those sets are firstly filtered out by the main Bloom filter. Hence $\mathcal{F}(A) - \mathcal{F}(S)$ and $\mathcal{F}(B) - \mathcal{F}(S)$ need not be queried to the cross-checking Bloom filters.

If $x \in \mathcal{F}(S)$, then it would be further queried to the cross-checking Bloom filters. The set $\mathcal{F}(S)$ can be decomposed into five disjoint sets as in Fig. 3, A, B, $(\mathcal{F}(A) - A) \cap \mathcal{F}(S)$, $(\mathcal{F}(B) - B) \cap \mathcal{F}(S)$, and $\mathcal{F}(S) - \mathcal{F}(A) - \mathcal{F}(B)$.

If $x \in \mathcal{F}(S)$, $x \notin \mathcal{F}(A)$, and $x \notin \mathcal{F}(B)$, i.e., $x \in \mathcal{F}(S) \cap (\mathcal{F}(A))^c \cap (\mathcal{F}(B))^c = \mathcal{F}(S) - \mathcal{F}(A) - \mathcal{F}(B)$, it is queried and filtered out by the cross-checking Bloom filters in our proposed architecture. In other words, the set $\mathcal{F}(S) - \mathcal{F}(A) - \mathcal{F}(B)$ represents the positive in the main filter but the negatives in both the cross-checking filters, and elements in the set are filtered out by the cross-checking Bloom filters.

Let the set E be the union of $((\mathcal{F}(A) - A) \cap \mathcal{F}(S))$ and $((\mathcal{F}(B) - B) \cap \mathcal{F}(S))$. The set E is the false positive set of our proposed architecture, and we will present that it is converged to an empty set when the sizes of cross-checking Bloom filters are increased even though the size of a main Bloom filter is not large enough.

We first assume that $\mathcal{F}(A)$ and $\mathcal{F}(B)$ are disjoint. Then the two sets composing the set E are disjoint, i.e., $((\mathcal{F}(A) - A) \cap \mathcal{F}(S)) \cap ((\mathcal{F}(B) - B) \cap \mathcal{F}(S)) = \emptyset$.

In the following statements of this section, we formulate the probability of the set E, and analyze the relationship of the probability of the set E with the sizes of cross-checking Bloom filters and the main Bloom filter.

Let η_A, η_B, and η_S represent the probabilities of A, B, and S, respectively, i.e., $\eta_A = P(A)$, $\eta_B = P(B)$, and $\eta_S = P(S)$. Since $S = A \cup B$ and $A \cap B = \emptyset$,

$$\eta_S = P(S) = P(A) + P(B) = \eta_A + \eta_B. \tag{11}$$

Hence

$$P(S^c) = 1 - P(S) = 1 - \eta_A - \eta_B. \tag{12}$$

Let f_A and f_B represent the conditional false positive probabilities of A and B given A^c and B^c, respectively, as (4), i.e., $f_A = P(\mathcal{F}(A)|A^c)$ and $f_B = P(\mathcal{F}(B)|B^c)$.

Since the set E is the false positive set of our proposed architecture, the probability of the set E represent the false positive probability where the main Bloom filter of S is firstly queried and then the cross-checking Bloom filters of A and B are queried.

Lemma 1.*The probability of the set E can be formulated as*

$$P(E) = (\eta_B + (1 - \eta_A - \eta_B) f_S) f_A + (\eta_A + (1 - \eta_A - \eta_B) f_S) f_B.$$

Proof: The probability $P(\mathscr{F}(A) - A)$ can be formulated as

$$\begin{aligned} P(\mathscr{F}(A)-A) &= P(\mathscr{F}(A)\cap A^c) \\ &= P(A^c)P(\mathscr{F}(A)|A^c) \\ &= (1-\eta_A)f_A. \end{aligned} \tag{13}$$

Similarly,

$$P(\mathscr{F}(B)-B) = (1-\eta_B)f_B. \tag{14}$$

The probability of $\mathscr{F}(S)$ can be formulated as

$$P(\mathscr{F}(S)) = P(S)P(\mathscr{F}(S)|S) + P(S^c)P(\mathscr{F}(S)|S^c). \tag{15}$$

From (7), (11), (12), and (15),

$$P(\mathscr{F}(S)) = \eta_A + \eta_B + (1-\eta_A-\eta_B)f_S. \tag{16}$$

The probability $P(\mathscr{F}(A) - \mathscr{F}(S))$ can be calculated as

$$\begin{aligned} P(\mathscr{F}(A)-\mathscr{F}(S)) &= P(\mathscr{F}(A)\cap(\mathscr{F}(S))^c) \\ &= P((\mathscr{F}(S))^c)P(\mathscr{F}(A)|(\mathscr{F}(S))^c). \end{aligned} \tag{17}$$

Since $A \subset S \subset \mathscr{F}(S)$, $(\mathscr{F}(S))^c \subset A^c$. In other words, if $x \in (\mathscr{F}(S))^c$, then $x \in A^c$. Hence

$$P(\mathscr{F}(A)|(\mathscr{F}(S))^c) = P(\mathscr{F}(A)|A^c) = f_A. \tag{18}$$

From (16),

$$\begin{aligned} P((\mathscr{F}(S))^c) &= 1 - P(\mathscr{F}(S)) \\ &= 1-\eta_A-\eta_B-(1-\eta_A-\eta_B)f_S. \end{aligned} \tag{19}$$

From (17), (18), and (19)

$$\begin{aligned} &P(\mathscr{F}(A)-\mathscr{F}(S)) \\ &= (1-\eta_A-\eta_B-(1-\eta_A-\eta_B)f_S)f_A. \end{aligned} \tag{20}$$

Since $\mathscr{F}(A) - A$ is the union of two disjoint sets $(\mathscr{F}(A)-A)\cap\mathscr{F}(S)$ and $\mathscr{F}(A)-\mathscr{F}(S)$ as in Fig. 3,

$$\begin{aligned} &P((\mathscr{F}(A)-A)\cap\mathscr{F}(S)) \\ &= P(\mathscr{F}(A)-A) - P(\mathscr{F}(A)-\mathscr{F}(S)). \end{aligned} \tag{21}$$

From (13), (20), and (21)

$$\begin{aligned} &P((\mathscr{F}(A)-A)\cap\mathscr{F}(S)) \\ &= (1-\eta_A)f_A - (1-\eta_A-\eta_B-(1-\eta_A-\eta_B)f_S)f_A \\ &= (\eta_B+(1-\eta_A-\eta_B)f_S)f_A. \end{aligned} \tag{22}$$

Similarly,

$$P((\mathscr{F}(B)-B)\cap\mathscr{F}(S)) = (\eta_A+(1-\eta_A-\eta_B)f_S)f_B. \tag{23}$$

Since the two sets composing the set E are disjoint,

$$\begin{aligned} P(E) &= P((\mathscr{F}(A)-A)\cap\mathscr{F}(S)) \\ &\quad + P((\mathscr{F}(B)-B)\cap\mathscr{F}(S)). \end{aligned} \tag{24}$$

From (22), (23), and (24),

$$\begin{aligned} P(E) &= (\eta_B+(1-\eta_A-\eta_B)f_S)f_A \\ &\quad + (\eta_A+(1-\eta_A-\eta_B)f_S)f_B. \end{aligned} \tag{25}$$

The probability $P(E)$ in Lemma 1 represents the false positive probability of S for an element $x \in U$ in our proposed architecture.

Now we consider the false positive probability given S^c ($x \notin S$).

$$P(E|S^c) = \frac{P(E\cap S^c)}{P(S^c)}$$

Since $E \subset S^c$, $P(E\cap S^c) = P(E)$.

Given S^c, $P(S^c) = 1$, $P(S) = 0$, and $\eta_A = \eta_B = 0$ from (11). Given S^c, the false positive probability in Lemma 1 is simplified as

$$P(E|S^c) = f_S(f_A + f_B). \tag{26}$$

In (26), $f_A + f_B = P(\mathscr{F}(A)|A^c) + P(\mathscr{F}(B)|B^c)$. Since $S^c \subset A^c$ and $S^c \subset B^c$, $f_A = P(\mathscr{F}(A)|S^c)$ and $f_B = P(\mathscr{F}(B)|S^c)$. If $\mathscr{F}(A)$ and $\mathscr{F}(B)$ are disjoint,

$$P(\mathscr{F}(A)\cap\mathscr{F}(B)|S^c) = 0. \tag{27}$$

Hence

$$\begin{aligned} f_A + f_B &= P(\mathscr{F}(A)|S^c) + P(\mathscr{F}(B)|S^c) \\ &= P(\mathscr{F}(A)\cup\mathscr{F}(B)|S^c). \end{aligned} \tag{28}$$

The term $f_A + f_B$ in (26) is the probability of $\mathscr{F}(A)\cup\mathscr{F}(B)$ given S^c, when $\mathscr{F}(A)$ and $\mathscr{F}(B)$ are disjoint.

We consider the case that $\mathscr{F}(A)$ and $\mathscr{F}(B)$ are not disjoint as

$$\begin{aligned} P(\mathscr{F}(A)\cup\mathscr{F}(B)|S^c) &= P(\mathscr{F}(A)|S^c) + P(\mathscr{F}(B)|S^c) \\ &\quad - P(\mathscr{F}(A)\cap\mathscr{F}(B)|S^c). \end{aligned} \tag{29}$$

Since the events $\mathscr{F}(A)$ given S^c and $\mathscr{F}(B)$ given S^c are independent,

$$P(\mathscr{F}(A)\cap\mathscr{F}(B)|S^c) = P(\mathscr{F}(A)|S^c)P(\mathscr{F}(B)|S^c) = f_A f_B \tag{30}$$

If $\mathscr{F}(A)$ and $\mathscr{F}(B)$ are not disjoint, (26) needs to be modified as

$$P(E|S^c) = f_S(f_A + f_B - f_A f_B). \tag{31}$$

In (31), we can see that the conditional false positive probability $P(E|S^c)$ is the multiplication of two terms: f_S and $f_A + f_B - f_A f_B$. The f_S term is related to the main Bloom filter and $f_A + f_B - f_A f_B$ term is related to the cross-checking Bloom filters. It would be possible to control the false positive probability $P(E|S^c)$ to be very small by adjusting the $f_A + f_B - f_A f_B$ term to be small.

To be more specific, we consider the relationship between Bloom filter sizes and the false positive probability. Let m_A, m_B, and m_S represent the Bloom filter sizes (the numbers of bits) for sets A, B, and S, respectively. Let n_A, n_B, and n_S represent the numbers of elements in sets A, B, and S, respectively. Let k_A, k_B, and k_S represent the numbers of hash functions for sets A, B, and S, respectively. Similar to Eq. (3), the false positive

probabilities f_S, f_A, and f_B can be represented approximately as

$$f_S \approx (1-e^{-k_S n_S/m_S})^{k_S}, \quad (32)$$

$$f_A \approx (1-e^{-k_A n_A/m_A})^{k_A}, \quad (33)$$

$$f_B \approx (1-e^{-k_B n_B/m_B})^{k_B}. \quad (34)$$

From (31), (32), (33), and (34), the conditional false positive probability $P(E|S^c)$ can be represented approximately as

$$\begin{aligned} P(E|S^c) \approx\ & (1-e^{-k_S n_S/m_S})^{k_S} \\ & \cdot\ ((1-e^{-k_A n_A/m_A})^{k_A} + (1-e^{-k_B n_B/m_B})^{k_B} \\ & -\ (1-e^{-k_A n_A/m_A})^{k_A}(1-e^{-k_B n_B/m_B})^{k_B}). \end{aligned} \quad (35)$$

In (33) and (34), if $m_A \to \infty$ and $m_B \to \infty$, then $f_A \to 0$ and $f_B \to 0$, respectively. In other words, as the sizes of cross-checking Bloom filters, m_A and m_B, becomes large, f_A and f_B converge to 0 theoretically and the set E shrinks to an empty set.

In (3), as the m is increased to a large value and the corresponding k is increased, the f_s would converge to 0 theoretically. In other words, as the Bloom filter size and the corresponding number of hash functions are increased, the false positive rate should be continuously decreased as in (3). However, the false positive performance depends on the randomness of the hash indices used in programming a Bloom filter, and implementing the hash functions providing the theoretically achievable performance in the false positive rate is not feasible [28]. In our simulations that we will present in Section 4, it is shown that f_s is saturated with a small value even though m and k are increased, mainly because of the limitation of the hash functions.

The saturation phenomenon would also happen in the cross-checking Bloom filters, but the overall false positive rate would be much smaller in the proposed architecture. For example, if the f_S, f_A, and f_B values are saturated in 10^{-4} with some moderately large values of m_S, m_A, and m_B, then the false positive rate in the proposed architecture would converge to $10^{-4}(10^{-4}+10^{-4}-10^{-4}10^{-4}) \approx 2\cdot 10^{-8}$ as in (31), while the false positive rate with a single Bloom filter would converge to the saturated value of 10^{-4}. Note that the saturated false positive rate can be obtained with some moderate sizes of Bloom filters. Hence the proposed architecture with two-stage Bloom filters can achieve much smaller saturated value of false positive rate compared to the conventional architecture with a single Bloom filter.

4. Performance Evaluation

In this section, we compare the number of false positives of our proposed architecture according to the required number of bits in Bloom filters with other structures: a single Bloom filter structure (single-BF) and a dual Bloom filter structure (dual-BF) [22].

Instead of using multiple different hash functions, we use a hash function based on a cyclic redundancy check (CRC) generator of 64 bits and obtain multiple hash indices from a CRC-64 code. There are several advantages of using a CRC generator as a hash generator. A fixed-length CRC code is obtained regardless of the length of each element of a set, and hence hash indices of elements with various lengths can be obtained using a single CRC generator. Any numbers of hash indices can be obtained from a single CRC code by combining different bits of the CRC code. Figure 4 shows an example of an 8-bit CRC generator. All the registers of the CRC generator are initially set to 0. Once an element with an arbitrary length is serially entered to the CRC generator, a fixed-length CRC code is obtained. By selecting a set of registers or multiple sets of registers from the code, we obtain as many hash indices in any length as desired.

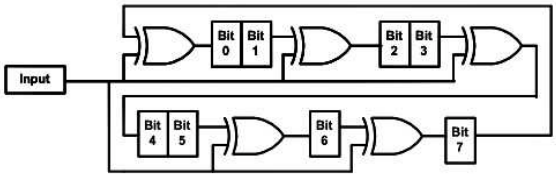

Figure 4 CRC-8 generator.

In our simulation, hashing indices for the main filter is obtained directly from a CRC code, and hashing indices for cross-checking Bloom filters are obtained from the string of the CRC code XORed with each input to give a variety in hashing indices. Similarly, for the single-BF implementation, the half number of hash indices are obtained directly from the CRC code itself and the remaining half are obtained from the string of a CRC code XORed with each input. For the dual-BF implementation, the second Bloom filter is programmed using the hash indices obtained from the CRC code of one's complemented values of inputs.

In evaluating the performance of a Bloom filter, any kind of data sets can be used. We have used the sets composed of nodes in binary tries of two routing data. The routing data are downloaded from actual backbone routers [29]. A binary trie is a tree-based data structure used in an IP address lookup. Each routing prefix resides in a node of the trie, in which the level and the path of the node from the root node correspond to the prefix length and the value, respectively, where each left edge represents bit 0 and each right edge represents bit 1. Figure 5 shows the binary trie for an example set of routing prefixes $\{0*, 1*, 11*, 101*\}$, which has maximum of 3-bit prefixes for simplicity, even though prefixes can

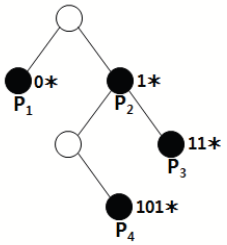

Figure 5 The binary trie for an example set of prefixes.

be up to 32 bits in real routing data. In Fig. 5, black nodes represent prefixes, and white nodes represent internal nodes.

In our simulation of the single-BF and the dual-BF implementations, the Bloom filters are programmed for entire nodes of a binary trie. For the simulation of our proposed architecture, the nodes of a binary trie are separated into two subsets: prefix node set and internal node set. The main Bloom filter is programmed for the entire nodes of the trie, and cross-checking Bloom filters A and B are programmed for prefix nodes and internal nodes, respectively. The Bloom filters are queried to identify whether an input is a node of the binary trie.

In our example, $S = \{*, 0*, 1*, 11*, 101*, 10*\}$, and two subsets, $A = \{0*, 1*, 11*, 101*\}$, $B = \{*, 10*\}$. The set S is the union of two disjoint sets A and B. The Bloom filters programmed for these sets using the hash indices obtained from the CRC-8 generator are shown in Figure 6. We define the basic size of a Bloom filter, $N = 2^{\lceil \log_2 n \rceil}$, where n is the number of elements in a set. The Bloom filters shown in Figure 6 are the cases of $4N$.

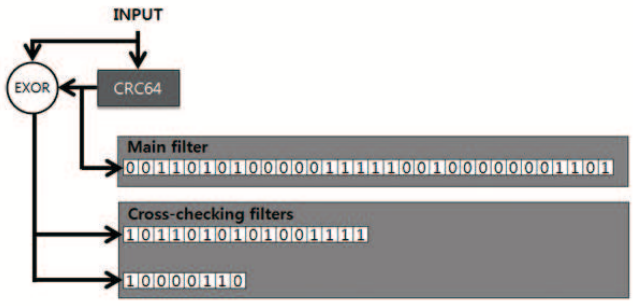

Figure 6 The proposed architecture programmed for the binary trie of Fig. 5.

Table 1 shows the number of prefixes, the number of internal nodes, and the basic size of each Bloom filter for two different routing data that were used for our simulation. For the single-BF structure, the Bloom filter was programmed for every node. For the dual-BF structure, the first Bloom filter was programmed for every node, and the second Bloom filter was programmed for one's complemented value of every node. For the proposed structure, the main Bloom filter was programmed for entire nodes, and the cross-checking Bloom filters were programmed for prefix nodes and internal nodes, respectively.

Table 2 shows the number of hash indices according to the size of each Bloom filter. The multiple M related to the basic size is defined as $M = m/N$, where m is the size (number of bits) of a Bloom filter. Hence the size of each Bloom filter is obtained as $m = MN$. Note that the size of each Bloom filter can be independently controlled by the multiple M times the basic size. The number of hash indices is calculated by $k = M \cdot (\ln 2)$ as in [5], [25].

The input traces for querying each structure consist of variable-length strings, of which each string is not included as a node of the binary tries. This condition corresponds to the assumption of S^c in the conditional probability formulation of $P(E|S^c)$ in Eq. (26). Hence, the querying result should be always negative. Every positive result is a false positive.

For the MAE-WEST routing data, 538916 input strings were applied. The size of each Bloom filter, the number of false positives, and the required memory amount is shown in Table 3. The number in the column of BF size in Table 3 represents the multiple M of the basic size.

Since the size of each Bloom filter is increased by the multiples of the basic size, the required memory amount cannot be controlled exactly the same for all three structures. Hence we compare the number of false positives of three structures for several combinations. Each row of Table 3 represents the combination which has a similar amount of memory for Bloom filters. If a combination does not exist for a specific memory amount, it is represented as NA, which means *not available*.

When the memory amount used for implementing Bloom filters is small, the proposed structure shows more number of false positives than the single-BF structure or the dual-BF structure, mainly because the main Bloom filter as well as the cross-checking Bloom filters are too small. However, when the memory amount is increased, the cross-checking Bloom filters in our proposed structure effectively remove the false positives. Hence the number of false positives is decreased much faster in our proposed structure than other structures. When the required memory amount is 448KB, the number of false positives of our proposed structure is 0, while other structures still have false positives even when the required memory amount is bigger. In case of the single-BF structure, even though the size of the Bloom filter is increased to 32 times of the basic size and the memory size is 512KB, the number of false positives is still more than 300, mainly because of the lack of variety in hashing indices generated by CRC-64.

Table 1 Characteristics of routing data and the basic size of each Bloom filter used for simulation

	No. of nodes			Single-BF	Dual-BF		Proposed multi-BF		
	total (n_S)	Prefixes (n_A)	Internal (n_B)	N_S (bit)	First BF N_S(bit)	Second BF N_S(bit)	Main BF N_S(bit)	Cross-checking BFs N_A(bit)	N_B(bit)
MAE-WEST	76989	14553	62436	131072	131072	131072	131072	16384	65536
Telstra	452905	227223	225682	524288	524288	524288	524288	262144	262144

Table 2 Number of hash indices according to the size of a Bloom filter

Multiples of basic size (M)	2	4	8	16	32	64
No. of hash index (k)	1	3	6	11	22	44

Table 3 Number of false positives according to the size of Bloom filters using hash indices from CRC-64 generator for MAE-WEST

	Single-BF		Dual-BF			Proposed multi-BF		
M	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)
NA	NA	NA	2-4	6346	96	4-4-4	5560	104
8	1482	128	4-4	1205	128	4-8-4	3541	112
NA	NA	NA	NA	NA	NA	4-8-8	740	144
16	331	256	8-8	11	256	8-16-16	3	288
NA	NA	NA	8-16	7	384	8-32-16	2	320
NA	NA	NA	NA	NA	NA	16-16-16	1	416
32	328	512	16-16	7	512	8-32-32	0	448

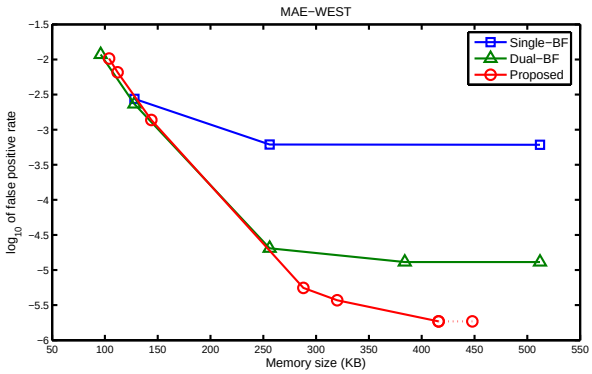

Figure 7 False positive rate in $\log_{10}$ scale according to the memory size of Bloom filters using hash indices from CRC generator for MAE-WEST.

The performance evaluation result depending on the Bloom filter memory size for MAE-WEST is shown in Fig. 7. The false positive rate is obtained by dividing the number of false positives with the number of input strings applied. When the false positive rate becomes smaller (e.g., less than 10^{-3}), it is hard to distinguish visually in graph. Hence it is displayed in log scale, i.e., $\log_{10}$ of false positive rate is displayed in Fig. 7. When the false positive rate approaches to 0, $\log_{10}$ of false positive rate approaches to minus infinity, hence it can not be displayed in Fig. 7. Hence we have used 1 instead of 0 and connected by a dotted line in order to denote in the figure.

We can see that the proposed architecture results in better performance compared to other single-BF and dual-BF structures, especially at large memory sizes.

For Telstra routing data, the input trace of 1358712 bit strings was applied. Every positive result is a false positive as in the case of MAE-WEST data. The size of each Bloom filter, the number of false positives, and the required memory size are shown in Table 4. It is shown that the dual-BF structure shows better performance than our proposed structure up to 512KB, but if the memory usage is bigger, the proposed structure shows better performance. When the memory usage is extremely large, which is 4096KB, the number of false positive is zero in our proposed structure, but it is still more than several ten-thousands in the single-BF structure and several thousands in the dual-BF structure. The $\log_{10}$ of false positive rate according the size of Bloom filters for Telstra is depicted in Fig. 8. In Fig. 8, we can see that the proposed architecture outperforms other architectures, especially at large memory sizes.

As shown in Table 3 and Table 4, the numbers of false positives in the single-BF structure and the dual-BF structure are not converged to 0 even though the size of the Bloom filter is significantly increased. It is mainly

Table 4 Number of false positives according to the size of Bloom filters using hash indices from CRC-64 generator for Telstra

	Single-BF			Dual-BF			Proposed multi-BF	
M	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)
4	163015	256	2-2	188941	256	2-2-2	274864	256
NA	NA	NA	2-4	57954	384	2-4-4	98313	384
8	60381	512	4-4	24172	512	4-4-4	38535	512
16	50584	1024	8-8	7820	1024	8-8-8	3225	1024
32	50510	2048	16-16	7184	2048	16-16-16	127	2048
64	50428	4096	32-32	7164	4096	32-32-32	0	4096

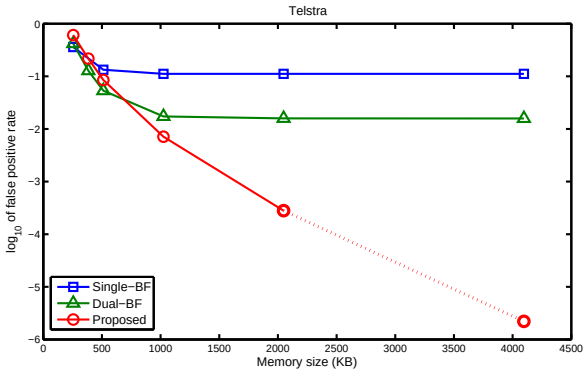

Figure 8 False positive rate in $\log_{10}$ scale according to the memory size of Bloom filters using CRC generator for Telstra.

because of the limitation using the CRC-64 generator as the hash function. We found out that extracting 22 or 44 hash indices from a 64-bit CRC code does not give much randomness in hash indices, and some inputs repeatedly produce false positives even though the size of the Bloom filter is increased. In other words, increasing the size of the Bloom filter (and accordingly the number of hash indices) does not effectively reduce the number of false positives for some of the input strings because of the lack of the variety in hash indices.

Therefore, we have performed another simulation using a different hash function: the random function provided in ANSI C. Using random function gives an advantage that the size of a Bloom filter does not have to be the power of 2 since we can obtain a hash index by performing the modulo operation with the size of a Bloom filter after generating a big random number. Hence we have used the size of Bloom filters as the multiples of the number of total nodes (and prefix nodes or internal nodes in our multi-BF structure) in this simulation. The simulation results are shown in Table 5 and Table 6 for the MAE-WEST and for the Telstra, respectively.

As shown, the filtering performance of the Bloom filters are better in every structure compared with the case that the CRC-64 is used as a hash function. The proposed multi-BF structure consistently shows better performance in the number of false positives than the single-BF and the dual-BF structures.

The $\log_{10}$ scales in the false positive rate according to the size of Bloom filters for the MAE-WEST and the Telstra are depicted in Fig. 9 and Fig. 10, respectively, when the random function is used as a hash generator. We can see that the proposed architecture still outperforms other architectures as the total size of Bloom filter is increased.

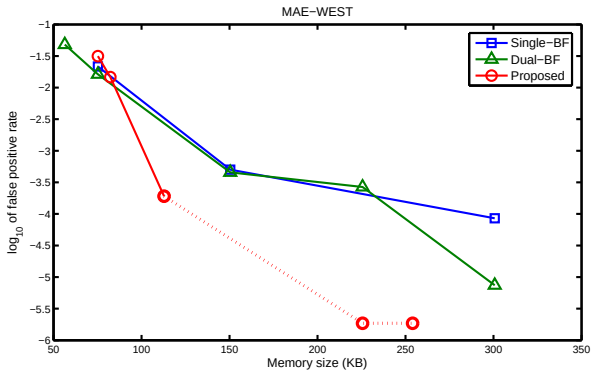

Figure 9 False positive rate in $\log_{10}$ scale according to the memory size of Bloom filters using hash indices obtained from a random function for MAE-WEST.

The comparison of false positive rate between simulation and probability model calculation for the proposed architecture is shown in Fig. 11 and Fig. 12. Figure 11 and Figure 12 are also displayed in log scale, since it is easier to distinguish when the false positive rate becomes small. The dotted lines represented by $simulation(modified)$ are for the cases that the number of false positives becomes zero. They are denoted by using 1 instead of 0 in the number of false positives since log of 0 converges to minus infinity. The false positive rate for probability model is obtained by Eq. (31). In these

Table 5 Number of false positives according to the size of Bloom filters using hash indices obtained from a random function for MAE-WEST

	Single-BF			Dual-BF			Proposed multi-BF	
M	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)
NA	NA	NA	2-4	25856	56.4	NA	NA	NA
8	11661	75.2	4-4	8752	75.2	4-4-4	16966	75.2
NA	NA	NA	NA	NA	NA	4-8-4	7905	82.3
NA	NA	NA	NA	NA	NA	4-8-8	103	112.8
16	271	150.4	8-8	245	150.4	NA	NA	NA
NA	NA	NA	8-16	144	225.6	8-16-16	0	225.6
NA	NA	NA	NA	NA	NA	8-32-16	0	254.0
32	46	300.7	16-16	4	300.7	NA	NA	NA

Table 6 Number of false positives according to the size of Bloom filters using hash indices obtained from a random function for Telstra

	Single-BF			Dual-BF			Proposed multi-BF	
M	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)	BF size	No. of false positives	Memory size(KB)
4	198897	221.1	2-2	208669	221.1	2-2-2	332055	221.1
NA	NA	NA	2-4	77212	331.7	2-4-4	143324	331.7
8	29495	442.3	4-4	28279	442.3	4-4-4	52465	442.3
16	1137	884.6	8-8	760	884.6	8-8-8	1233	884.6
32	552	1769.2	16-16	12	1769.2	16-16-16	4	1769.2
64	528	3538.3	32-32	9	3538.3	32-32-32	1	3538.3

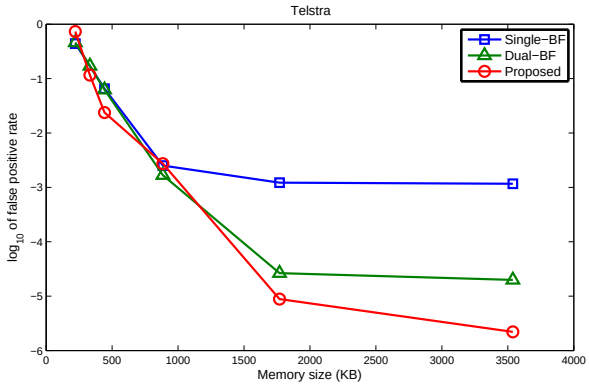

Figure 10 False positive rate in $\log_{10}$ scale according to the memory size of Bloom filters using hash indices obtained from a random function for Telstra.

figures, we can see that the false positive rate of simulation decreases in a similar way to that of probability model calculation as the memory size becomes larger.

5. Conclusion

A Bloom filter is popularly used in many applications because of its storage efficiency and operation simplicity. However, it has an intrinsic issue of false positives. Improving the false positive performance of a Bloom filter will give positive impacts to various fields which have Bloom filter applications.

The proposed structure of this paper has cross-checking Bloom filters which identify the false positive of a main Bloom filter. Simulation results show that the proposed structure has a smaller number of false positives when the memory usage is similar to the single-BF structure or the dual-BF structure. The performance difference is large when the sizes of simulation sets are large. It is also shown that the proposed structure has a better flexibility in required memory amount.

Our proposed architecture has an additional advantage. Since elements in a given set are grouped in a proposed structure, if we group the elements in their characteristics, the characteristics of inputs also can be identified by finding out which group the input belongs to as well as the membership of the input. For example, if some of the elements in a set have higher priorities than other elements, we can create a subset of higher priorities. The priority of each input can be identified by finding out the membership of inputs for the priority group.

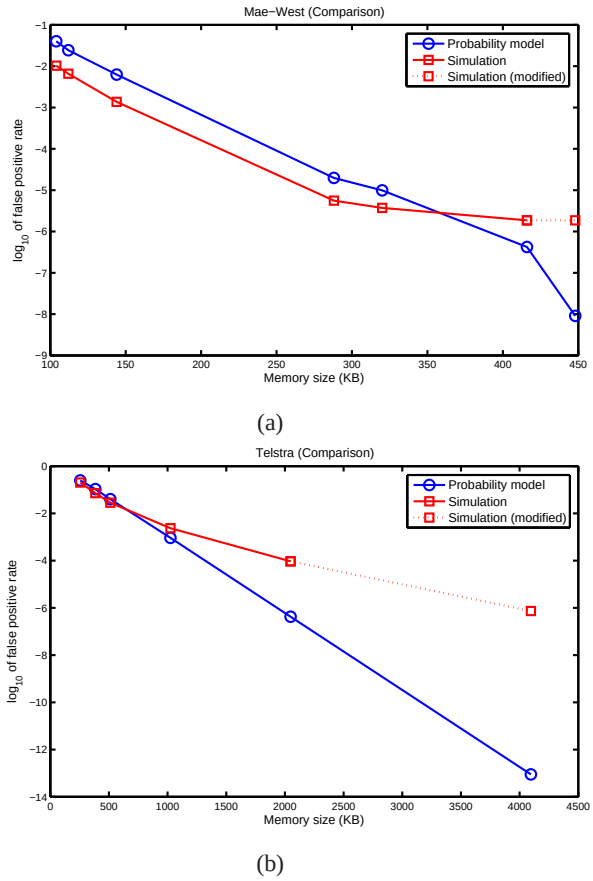

Figure 11 Comparison of false positive rate between simulation (CRC-64 generator) and probability model calculation for the proposed architecture. (a) MAE-WEST. (b) Telstra.

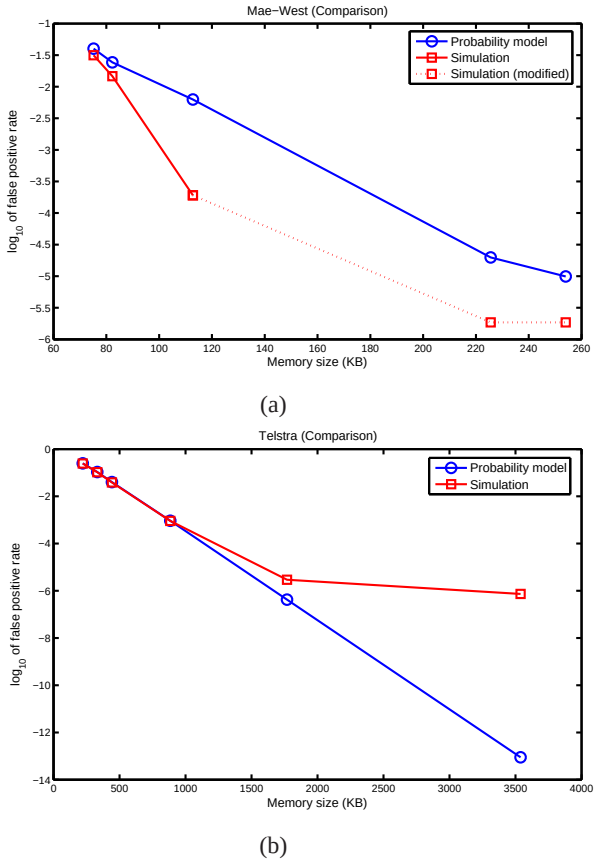

Figure 12 Comparison of false positive rate between simulation (random function) and probability model calculation for the proposed architecture. (a) MAE-WEST. (b) Telstra.

Acknowledgement

The research of the first author (H. Lim) was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (2012-005945). This research was also supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC support program supervised by the NIPA (NIPA-2013-H0301-13-1002).

The research of the corresponding author (C. Yim) was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0009426).

References

[1] Lim, Jae S., Two-dimensional signal and image processing, ADS, 710 (1990).

[2] Kane Yee, Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media, Antennas and Propagation, IEEE Transactions on, **14**, 302-307 (1966)

[3] Emre Telatar, Capacity of Multi-antenna Gaussian Channels, European Transactions on Telecommunications, European Transactions on Telecommunications, **10**, 585-595 (1999).

[4] B. Bloom, Space/Time Tradeoffs in Hash Coding with Allowable Errors, *Communications of the ACM*, **13**, 422–426 (1970).

[5] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, Longest prefix matching using Bloom filters, *IEEE/ACM Trans. Networking*, **14**, 397–409 (2006).

[6] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, Fast Hash Table Lookup Using Extended Bloom Filter: an Aid to Network Processing, *Proc. ACM SIGCOMM*, 181–192, (2005).

[7] H. Song, F. Hao, M. Kodialam, and T. Lakshman, IPv6 Lookups Using Distributed and Load Balanced Bloom Filters for 100Gbps Core Router Line Cards, *Proc. IEEE INFOCOM*, 2518–2526 (2009).

[8] H. Lim, K. Lim, N. Lee, and K. Park, On Adding Bloom Filters to Longest Prefix Matching Algorithms, *IEEE Trans. Computers*, IEEE Early Access, (2012).

[9] H. Lim and S. Kim, Tuple pruning using Bloom filters for packet classification, *IEEE Micro*, 30, 784–794 May/June (2010).
[10] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement, *IEEE Journal on Selected Areas in Communications*, **24**, 2327–2339 (2006).
[11] Y. Lu and B. Prabhakar, Robust Counting via Counter Braids: an Error-Resilient Network Measurement Architecture, *Proc. of IEEE INFOCOM*, 522–530 (2009).
[12] P. Reynolds and A. Vahdat, Efficient Peer-to-Peer Keyword Searching, *Proc. ACM/IFIP/USENIX International Conference on Middleware*, 21–40 (2003).
[13] A. Kumar, J. Xu, and E. Zegura, Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks, *Proc. of IEEE INFOCOM*, 1162–1173 (2005).
[14] L. Maccari, R. Fantacci, P. Neira, and R. Gasca, Mesh Network Firewalling with Bloom Filters, *IEEE International Conference on Communications*, 1546–1551 (2007).
[15] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, Deep Packet Inspection Using Parallel Bloom Filters, *IEEE Micro*, 52–61 (2004).
[16] S. Dharmapurikar and J. W. Lockwood, Fast and Scalable Pattern Matching for Network Intrusion Detection Systems," *IEEE Journal on Selected Areas in Communications*, **24**, 1781–1792 (2006).
[17] D. Suresh, Z. Guo, B. Buyukkurt, and W. Najjar, Automatic Compilation Framework for Bloom Filter Based Intrusion Detection, *Reconfigurable Computing: Architectures and Applications*, **3985**, 413–418 (2006).
[18] F. Chang, F. Wu-chang, and L. Kang, Approximate Caches for Packet Classification, *Proc. IEEE INFOCOM*, 2196–2207 (2004).
[19] L. Fan, P. Cao, J. Almeida, and A. Broder, Summary cache: A scalable wide-area web cache sharing protocol, *Proc. ACM SIGCOMM*, 254–265 (1998).
[20] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, Summary cache: a scalable wide-area Web cache sharing protocol, *IEEE/ACM Transactions on Networking*, **8**, 281–293 (2000).
[21] Y. Qiao, T. Li, and S. Chen, One Memory Access Bloom Filters and Their Generalization, *IEEE INFOCOM*, 1745–1753 (2011).
[22] F. Tabataba and M. Hashemi, Improving False Positive In Bloom Filter, *19th Iranian Conference on Electrical Engineering(ICEE)*, 1–5 (2011).
[23] O. Rottenstreich and I. Keslassy, The Bloom paradox:When not to use a Bloom filter?, *IEEE INFOCOM*, 1638–1646 (2012).
[24] D. Kim, D. Oh, and W. Ro, Design of power-efficient parallel pipelined bloom filter, *Electronics Letters*, **48**, 367–369 (2012).
[25] M. Mitzenmacher, "Compressed Bloom filters", *IEEE/ACM Trans. Networking*, **10**, 604–612 (2002).
[26] H. Song, J. Turner, S. Dharmapurikar, and J. Lockwood, Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing, *Proc. of Applications, Technologies, Architectures, and Protocols for Computer Communications*, 181-192 (2005).
[27] K. Lim, K. Park and H. Lim, Binary search on levels using a Bloom filter for IPv6 address lookup, *IEEE/ACM ANCS*, 185–186 (2009).
[28] A. Kirsch and M. Mitzenmacher, Less Hashing, Same Performance: Building a Better Bloom Filter, *Lecture Notes in Computer Science 4168*, 456–467 (2006).
[29] http://www.potaroo.net

Hyesook Lim received the B.S. and the M.S. degrees at the Department of Control and Instrumentation Engineering in Seoul National University, Seoul, Korea, in 1986 and 1991, respectively. She got the Ph.D. degree at the Electrical and Computer Engineering from the University of Texas at Austin, Texas, in 1996. From 1996 to 2000, she had been employed as a member of technical staff at Bell Labs in Lucent Technologies, Murray Hill, NJ. From 2000 to 2002, she had worked as a hardware engineer for Cisco Systems, San Jose, CA. She is currently a professor in the Department of Electronics Engineering, Ewha Womans University, Seoul, Korea, where she does research on router design issues such as IP address lookup, packet classification, and deep packet inspection and on hardware implementation of various network protocols such as TCP/IP and Mobile IPv6.

Nara Lee received the B.S. degree and the M.S. degree from the Department of Electronics Engineering at Ewha Womans University, Seoul, Korea, in 2009 and 2012, respectively. She is currently working as a research engineer for IP Technical Team of DTV SoC Department at SIC Lab. in LG Electronics Inc. Her research interests include various network algorithms such as IP address lookup, packet classification, web caching, and Bloom filter application to various distributed algorithms.

Jungwon Lee received the B.S. degree from the Department of Mechatronics Engineering at Korea Polytechnic University, Gyeonggi-do, Korea, and received the M.S. degree from the Department of Electronics Engineering at Ewha Womans University, Seoul, Korea, in 2011 and 2013, respectively. She is currently pursuing a Ph.D. degree from the same university. Her research interests include address lookup and packet classification algorithms and packet forwarding technology at content centric networks.

Changhoon Yim received the B.Eng. degree from the Department of Control and Instrumentation Engineering, Seoul National University, Korea, in 1986, the M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology, Korea, in 1988, and the Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin, USA, in 1996. He was a research engineer working on HDTV at Korean Broadcasting System, Korea, from 1988 to 1991. From 1996 to 1999, he was a member of technical staff in HDTV and Multimedia Division, Sarnoff Corporation, NJ, USA. From 1999 to 2000, he worked at Bell Labs, Lucent Technologies, NJ, USA. From 2000 to 2002, he was a software engineer in KLA-Tencor Corporation, CA, USA. From 2002 to 2003, he was a principal engineer in Samsung Electronics, Suwon, Korea. He is currently a professor in the Department of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea. His research interests include digital image processing, video compression, multimedia network, and multimedia communication.