Journal of The Institute of Electronics and Information Engineers Vol.54, NO.1, January 2017 https://doi.org/10.5573/ieie.2017.54.1.003
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2017-54-1-1

카운팅 블룸필터를 개선하는 터너리 블룸필터

(Ternary Bloom Filter Improving Counting Bloom Filter)

변 하 영*, 이 정 원*, 임 혜 숙**

(Hayoung Byun, Jungwon Lee, and Hyesook Lim©)

요 약

카운팅 블룸필터는 표준 블룸필터에서 제공하지 못하는 삭제 기능을 제공하여, 동적 집합에 대한 멤버쉽 쿼리를 허용하므로, 다양한 네트워크 알고리즘과 어플리케이션에 널리 사용된다. 그러나 카운팅 기능으로 인해 표준 블룸필터에는 없었던 오버플로우가 발생할 수 있고 이에 따라 거짓 음성이 발생할 수 있다. 4-비트 카운팅 블룸필터가 일반적으로 많이 사용되는데, 이는 모든 카운터에 4 비트를 할당하므로 메모리를 낭비한다는 단점이 있다. 거짓 음성의 가능성을 제거하고 메모리 사용량을 줄이기 위해서, 본 논문은 카운팅 블룸필터의 변형인 터너리 블룸필터(Ternary Bloom filter)를 제안한다. TBF는 하나의 카운터에 2개 이상의 원소가 대응될 경우, 더 이상의 삽입이나 삭제가 불가능하게 정한 구조이다. 실험을 통해 4-비트 카운팅 블룸필터와 같은 크기의 메모리 사용 시 TBF는 거짓 음성을 발생시키지 않을 뿐 아니라 거짓 양성률에 있어서도 상당한 우위를 보임을 확인하였다.

Abstract

Counting Bloom filters (CBFs) have been popularly used in many network algorithms and applications for the membership queries of dynamic sets, since CBFs can provide delete operations, which are not provided in a standard 1-bit vector Bloom filter. However, because of the counting functions, a CBF can have overflows and accordingly false negatives. CBFs composed of 4-bit counters are generally used, but the 4-bit CBF wastes memory spaces by allocating 4 bits for every counter. In this paper, we propose a simple alternative of a 4-bit CBF named ternary Bloom filter (TBF). In the proposed TBF structure, if two or more elements are mapped to a counter in programming, the counters are not used for insertion or deletion operations any more. When the TBF consumes the same amount of memory space as a 4-bit CBF, it is shown through simulation that the TBF provides a better false positive rate than the CBF as well as the TBF does not generate false negatives.

Keywords : Counting Bloom filter, dynamic set, counter, overflow, false positive

Ⅰ. 서 론

블룸필터[1]는 비트 벡터 형식의 공간효율적인 자료구조로써, 주어진 집합에 대한 멤버쉽 쿼리(membership query)를 제공해준다. 블룸필터를 사용하면 공간을 적게 사용하면서도 빠른 검색이 가능하다는 장점이 있어, 회계(accounting), 모니터링(monitoring), 로드 밸런싱(load balancing), 정책 적용(policy enforcement), 라우팅(routing), 필터링(filtering), 보안(security), 차등화 서비스(differentiated service), 캐시 디렉터리(cache directory)와 같은 다양한 응용 프로그램을 위해 널리 사용되고 있다[2~7]. 블룸필터는 집합에 포함되지 않는 원소를 집합에 포함된다고 판단하는 거짓 양성을 생성할 수 있지만, 집합에 포함되는 원소를 집합에 포함되지 않는다고 판단하는 거짓 음성은 생성하지 않는다. 블룸필터의 거짓 양성의 확률을 1% 이하로 조절하려할 때, 프로그래밍 되는 원소당 10비트 이하의 메모리를 요구하므로 메모리 사용량이 매우 낮다[8~9].

표준 블룸필터의 한 가지 취약점은 이미 프로그래밍된 원소를 삭제하는 것이 불가능하다는 것이다. 집합의

* 학생회원, 이화여자대학교 전자전기공학과 (Department of Electronic and Electrical Engineering, Ewha Womans University)
** 정회원, 이화여자대학교 전자전기공학과 (Department of Electronic and Electrical Engineering, Ewha Womans University)
© Corresponding Author(E-mail : hlim@ewha.ac.kr)
Received ; September 22, 2016 Revised ; October 17, 2016
Accepted ; December 13, 2016

원소가 동적으로 변화하는 경우 삭제기능이 필요한데, 삭제 기능을 제공하는 대표적인 변형은 카운팅 블룸필터[10]이다. 카운팅 블룸필터는 블룸필터의 각 비트를 c-비트 카운터로 대신한 데이터 구조로써, 하나의 블룸필터 인덱스에 대응된 원소의 개수를 카운팅함으로 원소의 삽입과 삭제를 가능하게 한다[11~12]. 그러나, 카운팅 블룸필터에는 두 가지 문제점이 있다. 첫 번째, 오버플로우 및 해시 충돌로 인한 잘못된 카운터 감소 때문에 거짓 음성을 발생시킬 수 있다. 두 번째, 최적화된 블룸필터의 경우 0으로 프로그래밍 된 카운터의 개수가 전체 블룸필터의 반 정도를 차지하는데, 0으로 프로그래밍 된 카운터들을 위해서도 c 비트를 할당하므로 메모리 낭비가 크다. 또한 c 비트가 모두 사용되는 확률도 매우 낮기 때문에 그 값이 0이 아닌 카운터의 경우에도 사용되지 않는 비트가 존재하여 메모리가 낭비된다.

본 논문에서는 이러한 문제들을 해결한 블룸필터 구조를 제안한다. 본 논문에서 제안하는 블룸필터는 블룸필터 동작에 치명적인 거짓 음성의 가능성을 없애고, 카운팅 블룸필터와 같은 메모리 사용량에 대하여 보다 낮은 거짓 양성률을 제공하는 구조이다. 본 논문의 구성은 다음과 같다. 2장에서는 해당 연구와 관련된 연구들을 살펴보고, 3장에서는 제안하는 구조에 대하여 소개하고 설명한다. 4장에서는 기존의 구조들과 제안하는 구조를 실험을 통해 비교 분석하여 성능을 평가하고, 5장에서 결론을 맺는다.

Ⅱ. 관련 연구

1. 블룸필터와 카운팅 블룸필터

블룸필터[1]는 주어진 집합에 대한 멤버쉽 쿼리를 제공하는 단일 비트 벡터 형식의 자료구조로써, 어떤 입력 값이 그 집합의 원소인지 아닌지를 판단해준다. 블룸필터를 사용하면 메모리를 적게 사용하면서도 빠른 검색이 가능하다는 장점이 있지만, 거짓 양성이 발생할 수 있다는 단점이 있다[9]. 블룸필터의 동작에는 프로그래밍(programing) 과정과 쿼링(querying) 과정이 있다.

크기가 m 비트인 블룸필터를 가정할 때 먼저 모든 비트를 0으로 초기화한다. 프로그래밍 과정에서 집합 $S = \{x_1, x_2, \cdots, x_n\}$의 모든 원소는 k개의 독립적인 해시 함수를 사용하여 프로그래밍 되는데, 사용되는 해시 인덱스는 0부터 $m-1$ 범위 안에서 랜덤하게 선택되어져야 한다. 하나의 원소를 프로그래밍 할 때, 이 원소에 대하여 구해진 k개의 해시 인덱스가 가리키는 비트를 모두 1로 셋한다. n개의 원소에 대하여, 해시 함수의 최적 개수 $k = (m/n)\ln 2$ 이다.

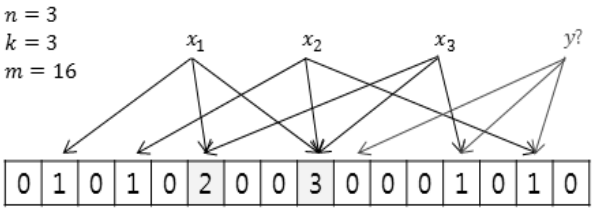

그림 1. 카운팅 블룸필터의 구조
Fig. 1. The structure of the counting Bloom filter.

쿼링 과정은 어떤 입력이 집합에 포함된 원소인지를 판단하는 과정으로, 프로그래밍에 사용된 해시 함수가 동일하게 사용된다. 주어진 입력에 대하여 추출된 k개의 인덱스가 가리키는 비트가 모두 1이면 양성 결과를 나타내고, 하나라도 0이면 음성을 나타낸다. 양성 결과 중에는 해시 충돌로 인한 거짓 양성이 존재할 수 있다. 거짓 양성은 프로그래밍 되지 않은 원소가 쿼링 과정을 거칠 때, 프로그래밍 된 비트들을 인덱스로 가지게 되어 발생한다. 즉, 주어진 입력이 해당 집합의 원소가 아님에도 원소라고 판단하는 것이다. 거짓 양성은 일반적으로 블룸필터 크기를 키워 개선할 수 있으나, 완전히 없앨 수는 없다.

카운팅 블룸필터(CBF)[10]는 동적으로 변하는 집합에 대하여 원소의 삽입과 삭제를 가능하게 하는 블룸필터로써, 블룸필터 각각의 인덱스에 c 비트를 할당하여 대응되는 원소의 개수를 카운팅한다. 그림 1은 카운팅 블룸필터의 구조를 나타낸다. x_1, x_2, x_3는 CBF에 삽입된 원소로서, 각 인덱스가 가리키는 위치의 카운터는 대응된 원소의 개수만큼 증가되었다. 원소 y는 해당 인덱스가 가리키는 위치에 0이 존재하므로 CBF에 삽입된 원소가 아니다. 하나의 원소가 삽입될 때, k개의 인덱스가 가리키는 각 카운터는 1씩 증가하는데, 이미 최대치인 (2^c-1)까지 증가된 카운터는 더 이상 증가할 수 없어 이 경우를 오버플로우가 발생했다고 한다. 쿼링 과정에서는 k개의 인덱스가 가리키는 카운터가 모두 1이상의 값을 가질 때 양성으로 판단하며, 하나의 카운터라도 0의 값을 가지면 음성으로 판단한다. 하나의 원소가 삭제될 때, k개의 인덱스가 가리키는 각 카운터는 1씩 감소하고, 값이 0인 카운터는 더 이상 감소되지 않는다. 오버플로우가 일어난 카운터에 대하여 (2^c-1)개 이상의 원소들이 삭제되는 경우, 해당 인덱스의 카운터가 실제로는 0이 아님에도 불구하고 0으로 감소하여, 거짓 음성을 발생시킬 수 있다. 프로그래밍 되지 않은 원소

가 삭제되는 경우에도 거짓 음성이 발생할 수 있지만, 그러한 경우는 고려하지 않는다. 거짓 음성의 확률을 10^{-15} 이하로 줄인 4-비트 카운팅 블룸필터가 많이 사용되고 있다.

2. 카운팅 블룸필터 관련연구

일반적으로 널리 쓰이는 4-비트 CBF는 표준 BF의 4배의 메모리를 사용하므로 메모리 소모가 크다. 카운팅 블룸필터의 메모리 요구량을 최소화하기 위해서 카운팅 블룸필터의 다양한 변형이 제안되어져 왔다.

완전 해싱(perfect hashing)을 사용하면 원소가 프로그래밍 되는 인덱스가 고르게 분포되어 카운터의 오버플로우가 발생할 확률이 감소하고, 더 적은 메모리 사용으로 같은 성능을 낼 수 있다. D-left CBF[13]는 거의 완전(almost perfect)한 d-left 해싱과 핑거프린트(fingerprints)를 기반으로, 잘 분산된 카운팅 블룸필터를 얻게 하여 전체 메모리 요구량을 감소시킨 구조이다. 그러나 핑거프린트를 사용하여 원소의 정보를 제공하고, 기존의 블룸필터와 같이 여러 위치를 확인하는 것이 아닌 하나의 위치만을 확인하므로, 블룸필터 구조보다는 해시 테이블 구조에 더 흡사하다고 할 수 있다.

Variable-Increment CBF (VICBF)[14]는 CBF의 효율을 증가시키기 위하여 가변적 증가(variable increment)를 사용하는 카운팅 블룸필터이다. 일반적인 CBF에서 각각의 원소가 삽입될 때, k개에 해시 인덱스에 대응하는 카운터가 1씩 증가하게 된다. 그러나 VICBF의 경우에는 이러한 유닛 증가(unit increment) 대신에 해시된 가변적 증가(hashed variable increment)에 의해서 카운터를 증가시킨다. 각 원소의 카운터 증가 값이 모두 다르기 때문에, 하나의 원소를 쿼리할 때, 해시 인덱스가 가리키는 카운터의 값이 해당 원소에 의해 증가 되었는지를 확인할 수 있어 거짓 양성률을 낮출 수 있다. 그러나 이러한 동작을 가능하게하기 위해서 가변적 증가를 선택하는 추가적인 해시 함수가 필요하고, 블룸필터의 인덱스의 값을 보고 해당 인덱스에 삽입된 모든 인덱스를 알 수 있어야 하므로 삽입 가능 제한이 있으며, 계산이 복잡하여 큰 프로세싱 오버헤드를 갖게 된다.

Multiple-Partitioned CBF[15]는 대규모 데이터 프로세싱 문제를 해결하기 위해 제안된 데이터 구조이다. k개의 해시함수에 대하여 k번의 메모리 접근을 해야 하는 표준 CBF와 비교하여, 일반적인 MPCBF-1경우에는 오직 1번의 메모리 접근만으로 검색을 완료할 수 있다. 그러나 g번의 메모리 접근으로 쿼리를 완료하도록 하는 MPCBF-g를 사용하는 것이 거짓 양성률을 개선시키기 위해 적합하며, 이를 위해서 비교적 더 높은 프로세싱 오버헤드를 갖게 된다. 또한 MPCBF는 계층적 구조를 사용하여 메모리 사용량을 줄이므로, CBF와 같은 메모리 사용량에 대하여 거짓 양성률을 낮출 수 있다. 그러나 이를 위해 사용되는 계층-CBF (hierarchical CBF)는 현재 집합의 상황에 맞추어 최대한 여분의 비트나 공간이 없도록 계층이 구성된 것이므로, 새로운 원소가 삽입될 경우 오버플로우가 좀 더 빈번하게 발생할 수 있다. 따라서 동적으로 변화가 큰 집합일 경우 오버플로우가 많이 발생하거나 전체 구조가 자주 바뀌어야 한다.

Deletable BF[16]는 블룸필터를 몇 개의 구역으로 나눈 후, 프로그래밍 시 어느 구역에서 충돌이 발생했는지에 대한 정보를 표현하는 추가적인 데이터를 갖는 구조이다. 블룸필터에는 없는 삭제기능을 제공하면서도 카운팅 블룸필터보다 적은 메모리 소모량을 가진다. 그러나 원소를 삭제할 때, 충돌이 발생했다고 표시된 구역의 비트들은 삭제 가능하지 않고, 오직 인덱스당 하나의 원소만이 삽입된 구역의 비트들만 삭제 가능하므로, 만약 하나의 구역에 오직 하나의 충돌만이 존재하더라도 그 구역의 모든 비트들은 삭제가 불가능하다는 단점이 있다.

이외에도 카운팅 블룸필터에 계층적인 압축을 사용하여 성능을 향상시키는 Huffman CBF[17], 하나의 카운터에 고정된 길이의 비트를 사용하는 것이 아닌 가변 길이의 비트를 사용하는 Variable Length CBF[18], Rank-Indexed 해싱을 사용하여 카운팅 블룸필터의 기능을 제공하는 핑거프린트 해시 테이블 구조[19] 등의 다양한 연구들이 제안되어졌다.

Ⅲ. 제안하는 알고리즘

제안하는 구조는 CBF에서 0으로 프로그래밍 된 카운터를 위해서도 c-비트를 할당해야 하는 낭비를 해결하기 위하여 출발하였다. 본 논문에서는 CBF와 같은 양의 메모리를 사용하되 하나의 카운터에 c보다 적은 비트를 할당하고, 그 대신 블룸필터에 포함되는 카운터의 개수를 키울 것을 제안한다. 카운터의 개수가 많아지면 이에 따라 해시 인덱스의 최적 개수도 늘어나게 되므로, 입력의 멤버십 여부를 판단하거나 프로그래밍 된 원소를 삭제하고자 할 때 오버플로우가 일어난 카운터는 사용하지 않는 방식이다. 제안하는 블룸필터 구조는 이러한 방식으로 CBF의 치명적인 단점인 거짓 음성의 가능성을 제거하였고, CBF와 같은 크기의 메모리

사용하였을 때 거짓 양성의 비율을 줄여 더욱 향상된 성능을 보인다.

1. 터너리 블룸필터(Ternary Bloom filter)

제안하는 블룸필터 구조에서의 각 카운터를 셀로 칭하겠다. 제안하는 Ternary Bloom filter(TBF)에서의 각 셀은 총 3종류(0, 1, X)의 값을 갖는다. 값 X는 하나의 셀에 원소가 2개 혹은 그 이상이 대응되었을 때를 의미하여, 이 셀에는 더 이상 삽입이나 삭제가 가능하지 않다는 것을 의미한다. 그림 2는 집합의 원소 $\{x_1, x_2, x_3\}$가 프로그래밍 된 터너리 블룸필터의 구조를 보여준다. 그림 1에서와 같은 예제로 집합의 원소 $\{x_1, x_2, x_3\}$를 프로그래밍 하고자 할 때 기존의 블룸필터와 같이 0으로 초기화 되어 있는 블룸필터에 대하여 해시 인덱스에 해당하는 셀의 값을 확인하여 0인 값들은 1로 셋한다. 만약 이미 1로 셋되어 있는 셀에 대응되었다면 이는 이미 다른 원소에 의하여 프로그래밍 된 셀이므로, X로 변경한다.

제안하는 구조에서의 쿼링은 입력에 대응되는 셀들의 값을 확인하되 그 값이 X인 셀은 사용하지 않는다. 그 값이 X가 아닌 셀 중에서 0이 존재하면 음성 결과이고, X가 아닌 모든 셀들의 값이 1이면 양성 결과이다.

그림 2에서 원소 y는 해당 인덱스가 가리키는 셀에 0이 존재하므로 음성 결과임을 알 수 있다. 본 논문에서는 쿼링에 있어 어떤 입력에 대응되는 셀의 값이 모두 X인 경우를 고려하여 'Not determinable'을 새로이 정의하였다. 'Not determinable'이란 쿼링된 원소의 멤버십 여부를 판단할 수 없다는 의미이다. 제안하는 블룸필터에 이미 프로그래밍 된 어떤 원소를 삭제하고자 할 때에는 그 값이 X가 아닌 셀 중에서 블룸필터 셀의 값이 1인 셀들을 0으로 변경한다. 삭제 시에 대응된 블룸필터의 셀 값이 모두 X인 경우는 'Not deletable'로 정의한다. 'Not deletable'이란 해당 원소를 삭제할 수 없다는 의미이다.

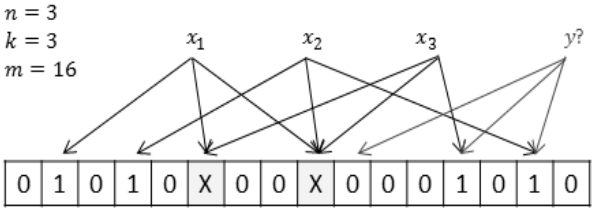

그림 2. 터너리 블룸필터의 구조
Fig. 2. The structure of the ternary Bloom filter.

2. 쿼터너리 블룸필터(Quarternary Bloom filter)

상용 메모리 기술에서는 값이 종류가 0, 1, X의 3종류를 갖는 경우에도 2비트의 카운터를 사용하여야 하고 이 경우 0, 1, 2, X의 4종류의 값을 표현할 수 있으므로, 본 논문에서 제안하는 TBF의 변형인 Quarternary Bloom filter(QBF)를 정의하였다. TBF에서의 하나의 셀은 1개의 원소만 대응될 수 있는데 반하여 QBF에서의 하나의 셀은 최대 2개의 원소가 대응될 수 있다. QBF는 어떤 셀에 3개 이상의 원소가 대응되는 경우 X로 표현하는 블룸필터로서, X의 값을 갖는 셀은 더 이상 사용되지 않는다는 점에서 TBF와 동일하다.

Ⅳ. 성능 평가

실험은 Web Information Company인 ALEXA[20]에서 제공하는 URL중 임의로 선정하여 수행하였다. 집합 S를 프로그래밍 한 후 S의 여집합(S^c)에 대해 쿼링을 수행하였는데, 이러한 방식으로 쿼링을 수행하면 집합에 포함된 원소는 하나도 존재하지 않으므로 모두 음성이 나와야 참인 결과이다. 따라서 양성이 나온 경우 이는 모두 거짓 양성이라 판단할 수 있다. 집합 S의 크기를 2의 승수(2^{11}, 2^{13}, 2^{15}, 2^{17})로 정한 다음 S^c의 크기는 집합 S의 2배로 정하였다. 표 1은 실험 집합에 포함된 원소의 개수를 나타낸다.

표 1. 실험 집합의 개수
Table1. The number of experimental data sets.

Sets	The number of elements in the universal set U	The number of elements in the set S	The number of elements in the set S^c
A	$2^{11}+2^{11}*2$	2^{11}	$2^{11}*2$
B	$2^{13}+2^{13}*2$	2^{13}	$2^{13}*2$
C	$2^{15}+2^{15}*2$	2^{15}	$2^{15}*2$
D	$2^{17}+2^{17}*2$	2^{17}	$2^{17}*2$

블룸필터의 크기는 한 셀 당 가장 많은 비트를 차지하는 4-비트 CBF를 기준으로 메모리 사용량이 같도록 설정하였다. 원소의 개수가 N인 집합에 대하여, 4-비트 CBF의 셀의 개수를 N, $2N$, $4N$으로 정할 때, TBF의 경우 값의 종류가 3개여서 하나의 셀 당 1.5 비트를 갖는다고 가정할 수 있으므로 TBF 셀의 개수는 $8N/3$, $16N/3$, $32N/3$이 되며, QBF 셀의 개수는 $2N$, $4N$, $8N$이 된다.

먼저 제안하는 블룸필터 구조에서 'Not determinable'이 나오는 경우에 대하여 실험하였다. 가장 많은 원소를 가지고 있는 집합 D에 대하여 TBF와 QBF를 만든 후 'Not determinable'이 나오는 비율을 측정하여 그림 3에 보였다. 그림 3의 x축은 블룸필터에 소요된 메모리 사용량을 나타낸다. y축은 'Not determinable'이 나온 비율로서, 'Not determinable'이 나온 입력의 개수를 전체 입력 개수로 나누어 나타내었다. 블룸필터가 256KB의 메모리를 사용할 때, 즉 TBF의 경우 셀의 개수가 $32N/3$이고, QBF의 경우 셀의 개수가 $8N$일 때, 'Not determinable'이 나오는 비율이 0으로 수렴하였음을 볼 수 있다.

다음은 원소의 삭제기능에 대하여 실험하였다. 삭제는 프로그래밍에 사용된 집합에 속한 원소만을 삭제함을 가정한다. 그러므로 삭제할 원소가 0으로 프로그래밍 된 셀과 대응되는 경우는 존재하지 않는다. 본 논문에서는 2장에서 언급한 여러 연구들 중에서 제안하는 구조와 같은 성능평가 기준을 갖는 Deletable BF[16]와 성능 비교를 진행하였다. Deletable BF(DBF) 셀의 개수는 $4N$, $8N$, $16N$으로 설정하되, 블룸필터의 일부 비트를 충돌이 일어난 구역 표시를 위하여 사용한다. DBF 구조에서 r-비트를 블룸필터의 충돌표시를 위하여 할당한다고 가정하면, 실제 프로그래밍 되는 블룸필터 사이즈(m')는 $m-r$ 비트이고, 본 실험에서는 r-비트가 전체 블룸필터 사이즈(m)의 1/5을 차지하는 경우를 DBF_1으로 표현하였고, 1/20을 차지하는 경우를 DBF_2로 표현하여 실험하였다. DBF_1의 경우 블룸필터 4비트 당 1비트가 충돌표시로 할당되었음을 의미하여, DBF_2의 경우 블룸필터 19비트당 1비트가 충돌표시로 할당되었음을 의미한다.

그림 4는 집합 D에 대한 'Not deletable'로 판단되는 비율을 보여준다. 이는 집합 S를 먼저 프로그래밍 한 다음 집합 S의 원소를 삭제할 때 해당 셀의 값이 모두 X여서 삭제가 가능하지 않은 경우의 비율이다. DBF의 경우 제안하는 블룸필터와 달리 정해진 구역 당 한 비트를 할당하여 충돌이 일어난 구역을 표시하기 때문에, 각 셀과 충돌 표시 비트가 일대일 대응이 아니다. DBF_1이 DBF_2보다 'Not deletable'이 적게 나오지만, 제안하는 구조보다는 높은 비율을 차지함을 볼 수 있다. 제안하는 구조인 QBF와 TBF를 비교해 보면 QBF가 TBF보다 값 X의 발생이 적기 때문에 'Not deletable'로 판단되는 비율 역시 작게 나옴을 볼 수 있다. 블룸필터의 사이즈가 증가할수록 해시 인덱스 k가 커지기 때문에 k개의 셀 값이 모두 X인 비율이 줄어들어 'Not deletable'의 발생률이 감소됨을 볼 수 있으며, TBF의 경우 셀의 개수가 $32N/3$이고, QBF의 경우 셀의 개수가 $8N$일 때, 'Not deletable'의 발생률이 0으로 수렴됨을 볼 수 있다. DBF의 경우 셀의 개수, 즉 블룸필터 크기가 커져도 'Not deletable'의 발생률이 0으로 수렴되지 않는다.

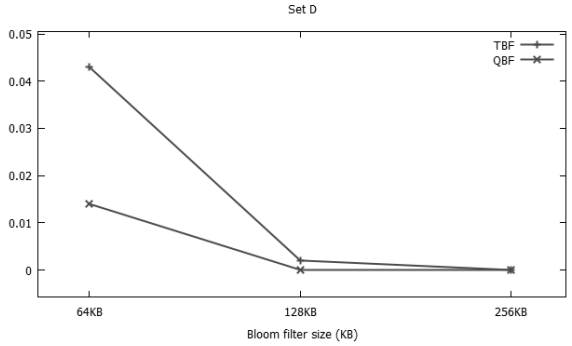

그림 3. Not determinable의 비율
Fig. 3. Rate of Not determinable.

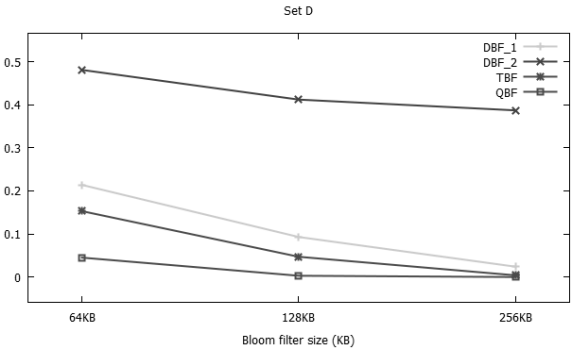

그림 4. Not deletable의 비율
Fig. 4. Rate of Not deletable.

그림 5에서는 각 알고리즘의 거짓 양성률을 비교하였다. 제안하는 블룸필터와 동일한 메모리 소요량을 보이는 4-비트 CBF와 DBF의 성능을 함께 비교하였다. 거짓 양성률은 각 집합 별로 거짓 양성의 개수를 쿼링된 입력 개수로 나눈 값으로 나타내었다. 비교를 진행한 4개의 블룸필터 모두, 사이즈가 증가할수록 거짓 양성률이 감소됨을 볼 수 있다. 4-비트 CBF의 경우 동일한 크기의 메모리를 사용하더라도 다른 블룸필터보다 거짓 양성의 비율은 현저히 높음을 알 수 있는데, 이는 앞서 설명한 바와 같이 모든 카운터에 4비트가 할당되었기 때문에 상대적으로 셀의 개수가 적은 블룸 필터이기 때문이다. DBF의 경우 거짓 양성의 발생률이 가장 적게 나타난다. 그 이유는 DBF는 각 셀의 크기가 표준 블룸필터 형태인 1 비트이기 때문이다. 즉 DBF_1의 경우 충돌 구역표시를 위하여 전체 블룸필터 크기의 1/5

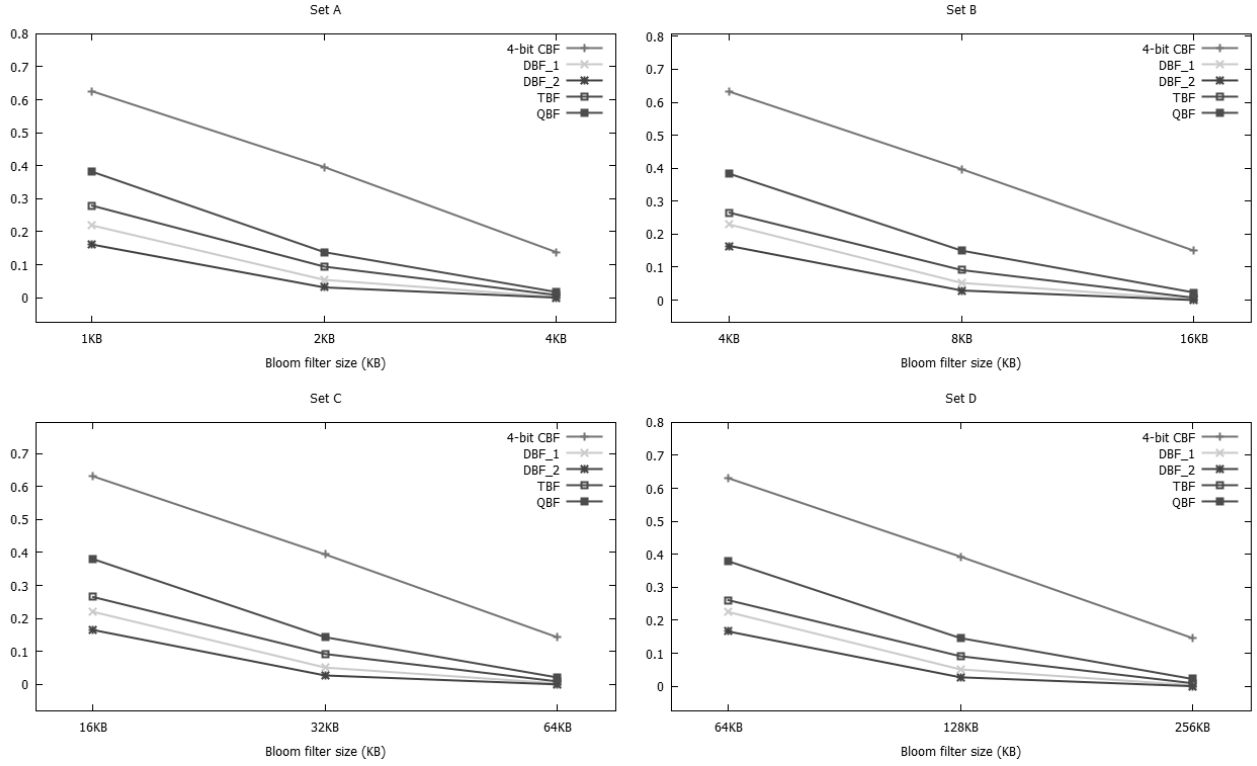

그림 5. 거짓 양성률
Fig. 5. False positive rate.

을 사용하여도 나머지 4/5가 모두 블룸필터로 사용될 수 있어 같은 메모리 사용량에서 셀의 개수가 더 많은 블룸필터를 만들 수 있고 이에 따라 더 많은 해시 인덱스를 갖게 된다. DBF_1보다 DBF_2일 때 셀의 개수가 더 많아지므로 거짓 양성률이 더 작게 나타난다. 작은 사이즈의 블룸필터에서는 TBF의 거짓 양성률에 관한 성능이 DBF보다는 좋지 못하지만 셀의 개수가 증가하면 모든 집합에서 거짓 양성률이 0.01 이하로 감소하면서 DBF와 비슷한 성능을 갖게 됨을 볼 수 있다.

V. 결 론

본 논문에서는 기존의 카운팅 블룸필터의 불필요한 메모리 낭비를 줄여 같은 메모리 사용량에서 거짓 양성률을 줄임과 동시에, 거짓 음성의 가능성을 제거하는 새로운 블룸필터 구조를 제안하였다. 제안하는 TBF 구조는 하나의 셀에 두개 이상의 원소가 대응되는 경우 X로 선언하여, 원소의 삭제 시 사용되지 않음을 표현하므로 거짓 음성의 가능성을 제거하였다. 제안하는 구조에서는 쿼링 시 'Not determinable'이 발생하는 경우와 삭제 시 'Not deletable'이 발생하는 경우를 새롭게 정의하였는데, 집합에 속한 N개의 원소에 대하여 TBF의 셀의 개수가 $32N/3$일 때, 그리고 QBF의 셀의 개수가 $8N$일 때 모두 0으로 수렴함을 보았다. 또한 4-비트 카운팅 블룸필터와 같은 메모리 사용량을 가질 때의 거짓 양성률을 비교하였을 때, 제안하는 구조에서의 거짓 양성률이 카운팅 블룸필터보다 월등히 우수함을 보았다.

REFERENCES

[1] B. Bloom, "Space/time Tradeoffs in Hash Coding with Allowable Errors," *Communications of the ACM*, Vol.13, No.7, pp. 422-426, Jul. 1970.
[2] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese, "Beyond Bloom Filters: From Approximate Membership Checks to Approximate State Machines," *in Proc. ACM SIGCOMM*, pp. 315-326, Pisa, Italy, Sep. 2006.
[3] H. Song, F. Hao, M. Kodialam, and T. V. Lakshman, "IPv6 Lookups Using Distributed and Load Balanced Bloom Filters for 100 Gbps Core Router Line Cards," *in Proc. IEEE INFOCOM*, pp. 2518-2526, Rio, Brazil, Apr. 2009.
[4] O. Rottenstreich and I. Keslassy, "The Bloom Paradox: When Not to Use a Bloom Filter," *IEEE/ACM Trans. on Networking*, Vol. 23, No. 3, pp. 703-716, Jun. 2015.

[5] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, “Longest Prefix Matching Using Bloom Filters,” *IEEE/ACM Trans. on Networking*, Vol. 14, No. 2, pp. 397-409, Apr. 2006.
[6] H. Lim, K. Lim, N. Lee, and K. Park, ”On Adding Bloom Filters to Longest Prefix Matching Algorithms,” *IEEE Trans. on Computers*, Vol. 63, No. 2, pp. 411-423, Feb. 2014.
[7] J. Mun, and H. Lim, “New Approach for Efficient IP Address Lookup Using a Bloom Filter in Trie-Based Algorithms,” *IEEE Trans. on Computers*, Vol. 65, No.5, pp.1558-1565, May. 2016.
[8] A. Broder and M. Mitzenmacher, “Network Applications of Bloom Filters: A Survey,” *Internet Mathematics*, Vol. 1, No.4, pp. 485-509, 2004.
[9] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, ”Theory and Practice of Bloom Filters for Distributed Systems,” *IEEE Communications Surveys and Tutorials*, Vol. 14, No. 1, pp. 131-155, First Quarter, 2012.
[10] L. Fan, P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” *IEEE/ACM Trans. on Networking*, Vol. 8, No.3, pp. 281-293, Jun. 2000.
[11] D. Ficara, A. Di Pietro, S. Giordano, G. Procissi, and F. Vitucci, “Enhancing Counting Bloom Filters through Huffman-Cooded Multilayer Structures,” *IEEE/ACM Trans. on Networking*, Vol. 18, No. 6, pp. 1977-1987, Dec. 2010.
[12] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, “Counter Braids: A Novel Counter Architecture for Per-Flow Measurement,” *in Proc. ACM SIGMETRICS 2008*, pp. 121-132, Annapolis, USA, 2008.
[13] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “An Improved Construction for Counting Bloom Filters,” *in Proc. ESA*, pp. 684-695, Zurich, Switzerland, Sep. 2006.
[14] O. Rottenstreich, Y. Kanizo, and I. Keslassy, “The Variable-Increment Counting Bloom Filter,” *IEEE/ACM Trans. on Networking*, Vol. 22, No. 4, pp. 1092-1105, Aug. 2014.
[15] K. Huang, J. Zhang, D. Zhang, G. Xie, K. Salamatian, A. Liu, and W. Li, “A multi-partitioning approach to building fast and accurate counting bloom filters,” *IEEE IPDPS*, pp. 1159-1170, Boston, USA, May. 2013.
[16] C. Rothenberg, C. Macapuna, F. Verdi, and M. Magalhaes, “The Deletable Bloom Filter: A New Member of the Bloom Family,” *IEEE Communications Letters*, Vol. 14, No. 6, pp. 557-559, Jun. 2010.
[17] D. Ficara, A. Di Pietro, S. Giordano, G. Procissi, and F. Vitucci, “Enhancing counting Bloom filters through Huffman-coded multilayer structures,” *IEEE/ACM Trans. on Networking*, Vol. 18, No. 6, pp. 1977-1987, Dec. 2010.
[18] L. Li, B. Wang, and J. Lan, “A variable length counting Bloom filter,” *in Proc. 2nd Int. Conf. on ICCET*, pp. 504-508, Kuala Lumpur, Malaysia, Apr. 2010.
[19] N. Hua, H. Zhao, B. Lin, and J. Xu, “Rank-indexed hashing: A compact construction of Bloom filters and variants,” *in IEEE ICNP*, pp. 73-82, Orlando, USA, Oct. 2008.
[20] Alexa the Web Information Company, http://www.alexa.com/

저 자 소 개

변 하 영(학생회원)

2014년 이화여자대학교 전자공학과 학사 졸업.

2014년 3월～이화여자대학교 전자전기공학과 석박사통합과정 재학.

<주관심분야: 라우터나 스위치 등의 네트워크 관련 알고리즘 및 구조 설계, 콘텐츠 중심 네트워크(CCN)>

이 정 원(학생회원)

2011년 한국산업기술대학교 메카트로닉스공학과 학사 졸업.

2013년 이화여자대학교 전자공학과 석사 졸업.

2013년 3월～이화여자대학교 전자전기공학과 박사과정 재학.

<주관심분야: 라우터나 스위치 등의 네트워크 관련 알고리즘 및 구조 설계, 콘텐츠 중심 네트워크(CCN)>

임 혜 숙(정회원)

1986년 서울대학교 제어계측공학과 학사 졸업.

1986년 8월～1989년 2월 삼성휴렛팩커드 연구원.

1991년 서울대학교 제어계측공학과 석사 졸업.

1996년 The University of Texas at Austin, Electrical and Computer Engineering 박사 졸업.

1996년 11월～2000년 7월 Lucent Technologies-Bell Labs, Member of Technical Staff.

2000년 7월～2002년 2월 Cisco Systems, Hardware Engineer.

2002년 3월～이화여자대학교 공과대학 전자전기공학과 정교수.

<주관심분야: 라우터나 스위치 등의 네트워크장비 설계 관련 알고리즘 및 하드웨어 구조 설계, 콘텐츠 중심 네트워크 (CCN), 소프트웨어 정의 네트워크 (SDN)>