http://dx.doi.org/10.7840/kics.2016.41.6.663

Cuckoo Hashing을 이용한 RCC에 대한 성능향상

장 룡 호◆, 정 창 훈*, 김 근 영**, 양 대 헌***, 이 경 희°

Enhancing RCC(Recyclable Counter With Confinement) with Cuckoo Hashing

Rhong-ho Jang◆, Chang-hun Jung*, Keun-young Kim**, Dae-hun Nyang***, Kyung-Hee Lee°

요 약

인터넷 트래픽양의 급증에 따라 고속 라우터의 수요가 많아졌다. 트래픽 통계 또는 보안 등의 목적으로 라우터에서 패킷을 측정해야 하는데 고속 라우터의 특성상 메모리공간이 제한적이다. RCC는 적은 메모리로 트래픽을 정확하고 효율적으로 측정하는 방법을 제시했다. RCC에서는 트래픽을 측정하는데 큰 Flow를 추가적인 Quadratic Probing 기반 해시 테이블에 누적하는 방법 사용한다. 그런데 Quadratic Probing은 적은 메모리 또는 메모리 사용률이 많은 상황에서 연산량이 많으며, 특히 갱신 또는 실시간 조회가 자주 발생하는 시스템에서 오버헤드가 크다. 이 논문에서는 RCC의 특성을 분석하고 실험을 통해 Quadratic Probing의 문제점을 증명하며 갱신 또는 조회에 효율적인 Cuckoo Hashing을 사용하여 RCC의 성능을 개선한다. 실험 결과에 따르면 RCC에서 Cuckoo Hashing을 사용할 때 메모리 사용률이 높은 상황에서도 높은 정확도를 보여주었고, 효율적으로 트래픽을 측정할 수 있었다.

Key Words : RCC, cuckoo hashing, quadratic probing, traffic measurement

ABSTRACT

According to rapidly increasing of network traffics, necessity of high-speed router also increased. For various purposes, like traffic statistic and security, traffic measurement function should performed by router. However, because of the nature of high-speed router, memory resource of router was limited. RCC proposed a way to measure traffics with high speed and accuracy. Additional quadratic probing hashing table used for accumulating elephant flows in RCC. However, in our experiment, quadratic probing performed many overheads when allocated small memory space or load factor was high. Especially, quadratic requested many calculations in update and lookup. To face this kind of problem, we use a cuckoo hashing which performed a good performance in update and loop for enhancing the RCC. As results, RCC with cuckoo hashing performed high accuracy and speed even when load factor of memory was high.

※ 이 논문은 2014년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (NRF-2014R1A1A2059852)
◆ First Author : Inha University Computer Science Engineering, ziyoo521@naver.com, 학생회원
° Corresponding Author : The University of Suwon Electrical Engineering, khlee@suwon.ac.kr, 정회원
* Inha University Computer Science Engineering, jcptk677@gmail.com, 학생회원
** Inha University Computer Science Engineering, kimky@isrl.kr, 학생회원
*** Inha University Computer Science Engineering, nyang@inha.ac.kr
논문번호 : KICS2016-04-072, Received April 29, 2016; Revised Nay 19, 2016; Accepted June 2, 2016

Ⅰ. 서 론

네트워크 통신기술의 발달로 정보화 사회가 되면서 네트워크 트래픽양은 매일 증가하고 있으며, Cisco는 2017년이 되면 전 세계에서 한 달 동안 발생하는 IP 트래픽 양이 급증할 것이라고 전망하고 있다[1]. 이에 따라 고속 라우터와 같은 장치의 수요가 증가할 것이라고 생각된다. 고속 라우터에서는 통계 또는 보안 등의 목적으로 트래픽을 측정하는 기능이 필요하다. 특히 보안 측면에서는 트래픽을 실시간으로 분석함으로서 대규모 Dos(Denial-of-service) 공격 또는 스팸을 예방할 수 있는 효과가 있다. 고속 라우터는 특성상 빠른 성능을 보장하기 위해서 SRAM와 같은 빠르고 비싼 메모리를 사용한다. 따라서 고속 라우터에서 트래픽을 측정하는 기법을 평가할 때는 메모리의 사용량이 가장 중요한 요소 중에 하나가 된다.

네트워크 트래픽을 측정하는 가장 간단한 방법은 샘플링(Sampling)다. Cisco의 Netflow[2]와 InMon의 Sflow[3]가 대표적인 예이다. 그러나 샘플링 기법은 Flow 별로 트래픽을 정확하게 측정하지 못하며, 특히 작은 Flow에 대해 정확하지 않다[4]. Flow별로 트래픽을 측정하는 동시에 메모리 공간을 줄이기 위해서는 Bloom Filters, Coding Techniques, Flajolet-Martin(FM) Sketches 그리고 Randomized Counter Sharing 등의 다양한 방법들이 소개 되었다[4-7]. D. H. Nyang 등은 Linear Counting[9]과 CSE[10] 기반으로 한 RCC(Recyclable Counter with Confinement)를 소개했다[11]. RCC는 Counter A와 B로 구성되어 있으며, Counter A에서는 Small-but-recyclable Counter를 이용하여 측정의 정확도뿐만 아니라 메모리의 절약효과도 얻었다. 또한 메모리 블록을 1 워드 내에 감금(Confinement)함으로서, 한 번에 메모리 액세스로 삽입(Insertion)과 조회(Lookup)가 가능하며, 이를 통해 측정 속도를 향상시켰다. Counter A에서는 작은 블록을 사용하기 때문에 포화 현상이 자주 발생하는데, RCC에서는 이것을 해결하기 위해서 포화된 블록의 값을 Counter B에 누적하고 블록을 재활용하는 방법을 사용한다. Counter B는 Quadratic Probing을 이용하는 해시 테이블이며, 삽입, 조회, 갱신이 모두 같은 연산량으로 이루어지기 때문에, 갱신 또는 조회가 자주 발생하는 상황 또는 테이블의 사용량이 많은 상황에서 좋은 성능을 보여줄 수 없다. 따라서 이 논문에서는 갱신 또는 조회에 있어서 좋은 성능을 보여준 Cuckoo Hashing[12]을 이용해 Counter B를 구성하고, 실험을 통해 Cuckoo Hashing을 이용한 RCC가 메모리 사용량이 높을수록 Quadratic Probing보다 더 적은 연산으로 트래픽을 측정할 수 있다는 것을 보여준다.

Ⅱ. 관련연구

C. Estan[13] 등은 멀티 필터를 사용해 큰 Flow를 측정하는 방법을 제시했고, [14-16]에서는 이와 비슷한 방법을 사용했다. 이러한 접근 방법은 큰 Flow에 대한 측정 방법이며 작은 Flow에 대해서는 측정하지 못한다.

SBF(Spectral Bloom Filter)[17]와 MRSCBF(Multi-resolution-space-code Bloom Filter)[5]는 Bloom Filter를 이용해 Flow별로 트래픽을 측정하는 방법을 제시했다. 그러나 두 기법은 작은 메모리 공간을 사용하는 경우, 트래픽을 측정하는 정확도가 높지 않았다. 특히 MRSCBF 같은 경우에는 많은 Decoding 연산이 측정 속도에 영향을 미치기도 한다.

PMC(Probabilistic Multiplicity Counting)는 하이브리드 구조를 기반으로 하여, 큰 Flow에 대해서는 FM Sketch[18]를 이용해 측정하고 작은 Flow은 HitCounter[9]를 이용해 측정하는 기법이다. 그 결과로 PMC는 MRSCBF보다 좋은 측정 정확도를 보여주었다.

D. H. Nyang등은 Linear Counting[9]과 CSE[10] 바탕으로 한 RCC를 소개했다[11]. RCC는 여러 Linear Counter를 이용해 트래픽을 Flow 별로 측정한다. 그리고 메모리 공간을 절약하기 위해서 Randomized Counter Sharing 기법을 사용했으며 Counter가 포화되었을 때 큰 Flow를 추가적인 테이블에 누적함으로서 Counter가 사용하는 블록을 재활용하는 방법을 사용했다. RCC는 CSE와 PMC보다 높은 정확도로 트래픽을 측정할 수 있을 뿐만 아니라 Counter를 1 워드 내에 감금함으로서 한 Flow에 대해서 메모리 액세스를 한 번만 하는 방법으로 빠른 측정 속도를 보여주었다.

RCC에서는 Quadratic Probing 기반의 테이블을 이용해 큰 Flow를 누적한다. Quadratic Probing은 메모리 사용률이 낮을 때는 좋은 성능을 보여주지만, 메모리 사용률이 높을 때는 삽입, 갱신, 조회할 때에 많은 연산량과 메모리 접근이 필요하기 때문에, 전체 성능을 저하시키는 원인이 될 수 있다.

트래픽 측정에서는 갱신 또는 실시간 조회가 자주 발생하는 특성이 있다. 따라서 이 논문에서는 갱신 또는 조회에 좋은 성능을 보여주는 Cuckoo Hashing[12]을 이용해 RCC가 더 작은 메모리 공간 또는 메모리 사용량이 높은 상황에서 효율적으로 동작하는 것에 기여한다.

Cuckoo Hashing을 이용하여 삽입 연산을 할 때에

는 Quadratic Probing보다 연산량이 많으나, 갱신 또는 조회할 때에는 $O(1)$에 완료할 수 있다. [12]에 따르면 Cuckoo Hashing이 두 개의 테이블과 해시함수를 사용했을 때 Load Factor는 0.5에 불과 했다. 하지만 이 논문에서는 테이블과 해시함수의 개수를 추가하므로 Cuckoo Hashing의 Load Factor가 향상되는 것을 실험으로 증명하려고 한다. 따라서 Cuckoo Hashing을 이용한 RCC가 네트워크 측정 또는 실시간 조회에 대하여 좋은 성능을 보여주는 것을 기대한다.

Ⅲ. Cuckoo Hashing을 이용한 RCC

일반적으로 트래픽 측정 기법의 정확도를 평가하는 척도는 두 가지가 있다. 하나는 메모리 블록의 포화(Saturation) 속도이고 나머지 하나는 한 Flow를 저장하는데 사용하는 메모리 블록(Memory Block)의 크기이다. 여기서의 메모리 블록은 Flow별로 측정하기 위해 각 Flow별로 할당한 메모리 공간을 뜻한다. 제한적인 메모리 공간에서 많은 Flow들을 측정하기 위해서는 각 Flow가 다른 Flow와 메모리를 공유하는 방법을 이용할 수 있으며, 이 때 메모리 공간을 랜덤하게 공유하는 블록을 가상 매트릭스(Virtual Matrix) 혹은 가상 벡터(Virtual Vector)라고 한다.

3.1 Recyclable Counter with Confinement

RCC[11]는 고속 라우터와 같은 장치에서 작은 메모리 공간을 이용해 네트워크 트래픽을 Flow별로 측정하는 카운팅 기법이다. 이는 CSE[10] 기반에서 Small-but-recyclable Counter와 메모리 블록을 1 워드 안에 감금하는 것을 이용하여 Flow별 측정 정확도와 속도를 높였다.

RCC에서는 각 Flow에 대해 한 가상 벡터를 할당하고 Linear Counting 이용해 측정하고 있으며, 정확한 측정을 위해 작은 크기의 가상 벡터를 사용한다. 그리고 가상 벡터의 포화를 방지하기 위해서 사용량이 70%가 되었을 때, 측정된 Flow의 예상 값을 계산한다. 그 다음에는 가상 벡터의 공유된 부분에 대한 노이즈를 제거하고, Quadratic Probing 해시 테이블에 계산한 예상 값을 누적함으로서 가상 벡터를 재활용(Recycle)한다.

그림 1은 가상 벡터의 구조와 한 가상 벡터가 포화되었을 때 재활용하는 과정을 보여주고 있다. 먼저 가상 벡터의 크기 s를 8-bit라고 가정하고, 이것은 그림 1의 Counter A에서 8개의 칸(a1-a8)을 의미한다. 이 8-bit 가상 벡터의 위치(Bit Position)는 Flow의 유일한

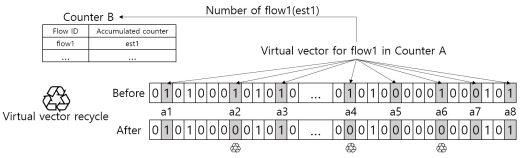

그림 1. 가상 벡터가 포화 발생할 때 재활용하는 과정
Fig. 1. Process of virtual vector recycling when saturation occurs

(Unique) 정보를 이용해 해시(Hash) 연산으로 계산할 수 있다. 새로운 패킷이 들어왔을 때 Randomized Hashing을 이용해 무작위로 8자리 중에 한자리를 1로 바꾼다. 이러한 과정을 반복하면 가상 벡터의 사용률이 70% 이상이 될 때가 있는데, Linear Counting의 특성상 메모리 공간의 사용률이 70%가 넘으면 정확한 측정이 불가능하기 때문에[9], 메모리의 사용률을 낮추어야 한다. RCC에서는 사용률이 70% 이상인 가상 벡터에 측정된 Flow의 예상 값을 계산하여 Counter B(Quadratic Probing 해시 테이블)에 누적시킨다. 예상 값을 계산하는 데에는 K. Y. Whang등이 제시한 Linear Counting에서 메모리 공간의 0의 개수에 따른 전체 개수의 예상 값 식(1)을 사용한다. ([9] 부록 A 참조).

$$\begin{aligned} E(U_n) &= \sum_{j=1}^{m} P(A_j) \\ &= m(1-\frac{1}{m})^n \cong me^{-n/m}, \quad as \;\; n,m\to\infty \end{aligned} \tag{1}$$

여기서 U_n은 메모리상의 0의 개수를 의미하며 j번째 bit가 0일 확률 $P(A_j)$는 $(1-1/m)^n$로 나타낼 수 있다. m은 메모리 크기고 n이 U_n개의 0이 존재 할 때의 패킷의 예상 양이다. RCC에서 더 정확한 측정을 위해서는 CSE에서 사용하는 근사 값인 $me^{-n/m}$을 사용하지 않고, 식(1)로부터 메모리상의 0의 비율 V_m을

$$V_m = \frac{E(U_n)}{m} = (1-\frac{1}{m})^{\hat{n}} \tag{2}$$

로 표현한다. 그 다음 (2)로부터

$$\ln(V_m) = \hat{n} \bullet \ln(1-\frac{1}{m}) \tag{3}$$

을 얻을 수 있다. 따라서 메모리상의 0의 비율에 따른 패킷의 개수는 다음과 같이 예상할 수 있다.

$$\hat{n}=\frac{\ln(V_m)}{\ln(1-1/m)} \quad (4)$$

그러므로 포화된 가상 벡터의 0의 비율을 V_m에 대입하고, 가상 벡터의 크기를 m에 대입하면 가상 벡터에 저장된 패킷의 양(k_2)의 예상 값을 계산할 수 있다.

RCC에서는 가상 벡터들이 메모리를 공유하고 있으므로 노이즈 k_1가 발생 한다. 전체 메모리의 평균 노이즈 k_1은 $\frac{s}{m} \bullet \hat{n}$로 표현할 수 있다[11]. 노이즈를 계산할 때는 m이 Counter A의 전체 메모리 공간이고 V_m은 전체 메모리상의 0의 비율이다. 따라서 Counter B에 누적해야할 Flow의 패킷 개수는

$$est=E(\hat{k})=k_2-k_1 \quad (5)$$

이다. 가상 벡터의 0의 비율이 0인 경우 가 0이 되는 경우가 있는데, 이러한 상황에서는 쿠폰 수집 문제(Coupon Collector's Problem)를 사용하여 k_2를 계산한다. 즉, $k_2=s\ln(s)+\gamma \bullet s+0.5$이다. 여기서 $\gamma \approx 0.5772156649$이다.

가상 벡터를 재활용하는 방법은 가상 벡터의 0의 비율을 $\frac{s}{m} \bullet Z$만큼에 맞추는 것이다. 여기서 Z와 m은 전체메모리 상의 0의 개수와 전체 메모리 공간이다. 가상 벡터에서 0으로 전환되는 1의 자리는 랜덤하게 선택된다.

메모리 블록을 1 워드 안에 감금하는 것은 Counter A를 1 워드씩 나눈 다음에 한 Flow를 저장하는 가상 벡터의 Bit Position을 1 워드 안에서 할당하는 방법이다(그림 2). Flow를 어떤 워드 블록에 할당 할지는 Flow의 해시 값을 이용하여 결정한다. 서로 다른 두 Flow가 같은 1 워드를 공유 할 수 있지만 Bit Position이 다르기 때문에 서로 메모리를 공유하더라도 식(5)와 같은 노이즈 제거 방법으로 가상 벡터에 저장된 Flow의 예상 값을 계산할 수 있다. 그리고 하나의 가상 벡터를 워드 하나로 제한하면 한 번의 메모리 접근만으로 모든 가상 벡터의 Bit Position을 읽고 쓸 수 있으며 측정의 속도도 빠르게 할 수 있다.

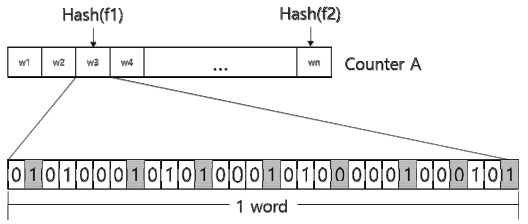

그림 2. 가상 벡터의 감금의 대한 설명
Fig. 2. Description of virtual vector confinement

3.2 Quadratic Probing

RCC에서는 Counter A중에 포화된 가상 벡터를 저장된 Flow의 패킷 개수를 Qudratic Probing 해시 테이블(Counter B)에 누적하고 재활용한다. Quadratic Probing은 테이블 충돌을 방지하는 목적으로 널리 사용되어지고 있다. 그러나 Quadratic Probing이 모든 경우에 좋은 성능을 보여주는 것이 아니다. Quadratic은 다음과 같은 수식으로 표현할 수 있다.

$$H(k,i)=(h(k)+c_1 i+c_2 i^2)\bmod(m) \quad (6)$$

먼저 Quadratic Probing에서 m의 선택이 가장 중요하다. 을 2보다 큰 소수로 선택하면 대부분 c_1, c_2의 선택에 대해 $h(k,i)$에서 i에 의한 차별(Distinct) 범위는 $[0,(m-1)/2]$로 작아진다. 따라서 Load Factor가 1/2보다 클 때에는 삽입의 성공률을 보장 할 수 없다. 그리고 키를 해시 테이블에 삽입을 할 때에는 충돌(Collision)이 발생하지 않을 때 까지 식(6)을 통해 테이블의 위치를 찾아야 한다. 따라서 충돌이 발생할 때마다 식(6) 연산, 메모리 액세스 그리고 키 비교연산이 한 번씩 발생하게 된다. RCC는 특성상 Counter A에서 크기가 작은 가상 벡터를 사용하기 때문에 Counter B에 대한 누적이 자주 발생 할 수밖에 없다. 만약 어떤 Flow가 N번의 연산을 통해 찾은 위치에 저장되었다면, 그 Flow에 대한 갱신 혹은 조회가 발생 할 때 마다 N번의 식(6) 연산, 메모리 액세스 그리고 키 비교연산을 해야 한다. 그래서 Counter B는 작은 메모리 공간에서 혹은 메모리 사용률이 높은 상황에서 충돌이 자주 발생해 많은 연산 또는 메모리 액세스가 필요하다. 따라서 Quadratic Probing이 모든 상황에서 효율적으로 적용되지는 않는다.

3.3 Quadratic Probing

이 논문에서는 RCC에서 Counter B로 누적을 할 때 Quadratic Probing 대신 Cuckoo Hashing을 이용하여 Counter B의 성능을 향상시키려고 한다. Cuckoo Hashing은 뻐꾸기의 습성과 유사한 형태를 이용한 테이블 충돌 방지기법이다. 뻐꾸기는 다른 종류의 새둥지에 알을 낳으며 먼저 깨어난 뻐꾸기 다른 알을 밀어낸다. 유사한 형태로 Cuckoo Hashing에서는 키가 삽입

되면 이전에 있었던 키는 자리에서 밀려 나간다. 밀려나온 키는 다른 자리를 찾아가게 된다. 그러므로 Cuckoo Hashing은 각 키에 대해 한 개 이상의 자리(Index)를 가지고 있다 $(h_1(key), h_2(key), \ldots, h_n(key))$. 따라서 key_1을 테이블에 삽입할 때 먼저 Greedy 알고리즘으로 비어있는 자리를 찾아가고, 만약 비어 있는 자리가 없다면 $h_1(key_1), h_2(key_1), \ldots, h_n(key_1)$중에 한 자리의 key_2를 밀어내고(Kickout) 그 자리로 들어간다. 밀려나온 key_2도 key_1과 같은 과정을 거쳐 자리를 찾는다.

R. Pagh 등은 Cuckoo Hashing을 소개할 때, 두 개의 테이블과 두 개의 해시함수를(One Hash per Table) 사용하는 모델로 설명하였고, Load Factor는 0.5에 불과하다고 주장했다[12]. 이 논문에서는 실험을 통하여 기존 모델에서 테이블과 해시함수를 한 개씩 추가하여 Load Factor가 0.9까지 올라가는 것을 증명했다. 그리고 테이블과 해시함수를 세 개 이상 사용하지 않는 이유는 Cuckoo Hashing에서 조회가 발생할 때 최악의 경우에는 테이블 개수만큼 메모리 액세스를 하기 때문에, 많은 조회가 발생하는 실시간 트래픽 측정환경에서는 테이블 개수를 최소로 사용하는 것이 좋은 성능을 기대할 수 있으므로 테이블을 세 개만 사용을 하였다. 우리는 이를 RCC에 적용하여 성능을 개선하려고 한다.

Cuckoo Hashing에서 삽입 연산을 할 때, 충돌이 발생하면 그때 마다 다른 자리를 찾아가기 위해 키에 대한 해시 연산을 한번 해야 한다. Quadratic Probing에서 식(6)의 더하기 두 번, 제곱 한 번과 비교했을 때 비싼 연산이라고 할 수 없다. 그렇다면 두 기법에서 충돌로 발생하는 연산량이 같다고 가정했을 때, 성능을 좌우하는 것이 충돌 횟수라고 할 수 있으며, 이는 메모리 액세스 횟수라고 볼 수 있다.

Quadratic Probing에서 조회 또는 갱신 연산을 할 때의 최악의 경우 $O(n)$이며, Cuckoo Hashing은 테이블 또는 해시함수의 개수($O(1)$)만큼만 하면 된다. RCC에서 크기가 작은 가상 벡터를 사용하는 점과 메모리의 사용량이 많은 경우를 고려하면 Cuckoo Hashing은 Quadratic Probing보다 좋은 성능을 보여줄 수 있다.

Ⅳ. 실험 및 분석

이 장에서는 RCC에서 Quadratic Probing과 Cuckoo Hashing을 사용할 때의 성능차이에 대하여 비교 및 분석을 한다. 또한 Cuckoo Hashing에서 세 개의 테이블과 세 개의 해시함수를 사용할 때의 Load Factor를 실험을 통해 알아보려고 한다. 실험에서는 CAIDA 데이터 셋[19]을 사용했다. 이 데이터 셋은 2013년 11월 21일 13:10-13:11사이에 Equinix Chicago Data Center에서 기록된 트래픽이며 264K개의 Flow를 담고 있다.

4.1 실험 환경 및 방법

이 실험에서는 데이터 셋의 패킷이 기록된 순서대로 하나씩 읽어서 발송지 주소(Source IP)의 해시 값을 이용해 Counter A에서 해당 Flow의 가상 벡터를 찾아 누적한다. 그리고 Counter A의 가상 벡터가 포화되면 Counter B에 누적한다. 우리는 Counter A가 포화되었을 때 Counter B에 누적되는 연산에 대하여 성능을 향상시키려고 하는 것이 목적이기 때문에, Counter A의 성능은 이 논문에서 고려하지 않는다. 그리고 Quadratic Probing을 이용하는 Counter B와 Cuckoo Hashing을 이용하는 Counter B를 같은 실험 조건에서 평가하기 위해 이 둘을 하나의 Counter A와 병렬로 동작하도록 구현했다. 따라서 Counter A에서 포화된 가상 벡터가 있을 때 동시에 두 Counter B에 누적되며 두 Counter B에 저장된 Flow들의 정보는 서로 같다. 마지막으로 모든 측정이 끝난 다음 데이터 셋의 모든 Flow에 대해 한번씩(중복 없이) 조회한다([11]의 IV.B 참조). Quadratic Probing과 Cuckoo Hashing는 모두 .NET에서 제공하는 SHA256를 사용한다(표 1).

앞서 3장에서 제시한 것처럼 Quadratic Probing 또는 Cuckoo Hashing은 충돌이 발생할 때 마다 연산이 발생하기 때문에, 테이블에서 충돌이 발생하는 횟수가 성능을 좌우한다고 볼 수 있다. 따라서 이 실험에서는 두 Counter B를 조작(삽입, 갱신, 조회)하는 과정에서 발생하는 메모리 접근 횟수를 각각 기록하여 Quadratic Probing과 Cuckoo Hashing의 성능을 비교 및 분석하려고 한다. 또한 데이터 셋의 모든 Flow를 조회하여 측정된 값과 실제 Flow의 양을 비교하여 Counter A

표 1. RCC 알고리즘 탑재 장치의 사양
Table 1. Specification of station for RCC algorithm

	Description
CPU	Intel Core i5-4460
RAM	16GB
OS	Windows System
Program Language	C# in Visual Studio
Hash function	SHA256(System.Security.Cryptography)

또는 두 Counter B의 구현 정확도를 확인한다.

RCC의 구현은 [11]에서 제시한 Encode과 Decode 알고리즘 기준으로 C#을 이용하여 구현하였다. 여기서 RCC를 구현하기 위해서는 추가적인 라이브러리가 필요 없기 때문에 C#이 아닌 다른 언어를 사용하여도 구현상의 어려움은 없다. 또한 메모리 접근 횟수로 Quadratic Probing과 Cuckoo Hashing의 성능을 비교하려고 하는 것이기 때문에 C#의 성능이 실험 결과에 영향을 주지 않는다. Quadratic Probing을 이용한 Counter B에서는 식(6)에서 $c_1 = 0$, $c_2 = 1$ 로 설정하여 테이블 충돌을 해결하게 하였다. Cuckoo Hashing을 이용한 Counter B는 세 개의 테이블로 나누어져 있기 때문에 각 테이블은 Counter B의 메모리 공간의 1/3이 할당된다. 이 실험에서 사용하는 Cuckoo Hashing은 [12]에서 제시한 알고리즘을 바탕으로 테이블과 해시함수를 각각 하나씩 추가한 형태로 구현되어 있으며 기존 모델과 같은 방식으로 동작한다(IIV.3). 세 개의 해시함수는 SHA256로부터 얻은 256bit 값을 이용해서 여러 32bit해시 값을 파생할 수 있기 때문에 여러 해시함수를 사용하는 것과 같은 효과를 얻을 수 있다. 이 실험에서는 다소 무거운 SHA256을 사용했지만 Counter B에서 사용하는 해시함수는 더 가벼운 해시를 사용해도 된다.

4.2 실험의 파라미터

이 실험에서의 Counter A는 RCC[11]에서 제시한 32-bit 가상 벡터를 사용할 때의 파라미터를 사용하고 있다(표 2). 8-bit 가상 벡터를 사용하지 않는 이유는 가상 벡터가 작을수록 큰 Flow를 Counter B에 반복 누적하는 횟수가 증가하기 때문에 Cuckoo Hashing이 더 유리하기 때문이다. Counter B에서는 테이블의 Bucket Size가 55bit이며 그 중에 키가 32bit를 사용하고 누적한 패킷의 양(Value)은 22bit에 저장한다. Quadratic Probing은 테이블과 해시함수를 한 개씩 사용하는 반면 Cuckoo Hashing은 세 개의 테이블과 세 개의 해시함수를 사용한다. Counter B의 크기는 Load Factor의 값에 따라 2.4Mb부터 4.4Mb까지 할당한다. 2.4Mb를 할당 했을 때 Counter B의 Load Factor가 약 0.9가되며 Quadratic Probing과 Cuckoo Hashing의 메모리 사용률이 높을 때의 성능을 분석할 수 있다. 이 실험에서는 주어진 데이터 셋[19]에 대해 Counter B의 Load Factor가 0.5부터 0.9까지 되도록 메모리공간을 할당하여 비교 및 분석한다.

표 2. RCC 및 Counter B에 대한 파라미터 설정
Table 2. Parameters of RCC and Counter B

		Quadratic probing	Cuckoo hashing
Counter A	Memory size	3.5 Mb	
	Virtual vector size	32 bit	
	Confinement size	32 bit	
Counter B	Memory size	2.4 Mb - 4.4 Mb	
	Bucket size	55 bit	
	Key size	32 bit	
	Value size	22 bit	
	Num. of table	1	3
	Hash func. per table	1	1

4.3 결과 및 분석

[그림 3]은 Quadratic Probing을 이용한 Counter B

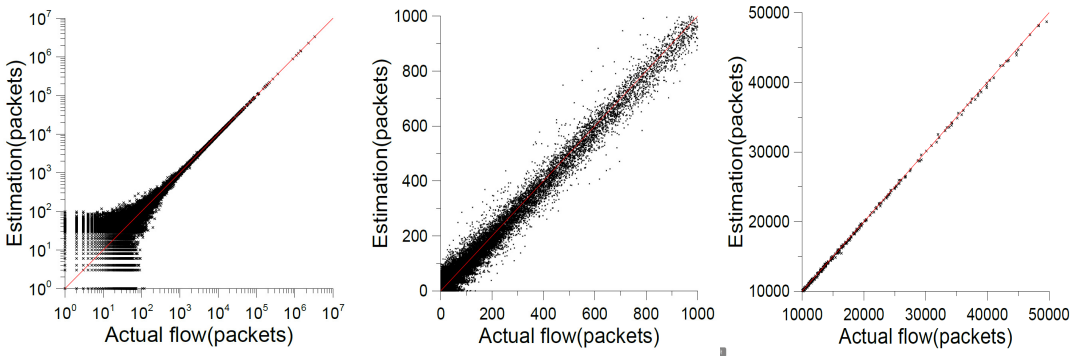

그림 3. Quadratic probing을 이용한 RCC의 측정결과(Counter B 2.4Mb). 그림 상의 한 점은 한 Flow에 대한 측정결과를 의미하며 x축은 Flow의 정확한 패킷의 양이고 y축은 측정된 Flow의 양이다. 따라서 선 y=x에 가까울수록 측정이 정확하다.
Fig. 3. Estimation result of RCC with quadratic probing(Counter B 2.4 Mb). Each point in figure points estimation result of one flow. X-Axis means actual packet number of one flow and Y-Axis means estimated packet number. More closer point to y=x means more actual flow.

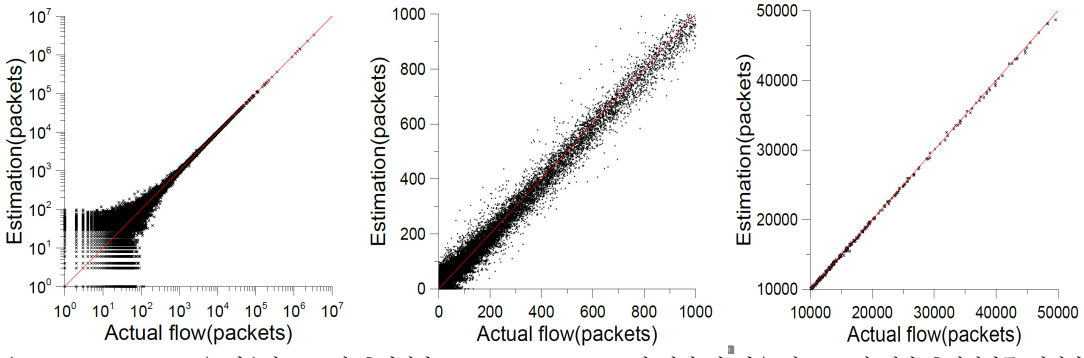

그림 4. Cuckoo hashing을 이용한 RCC의 측정결과(Counter B 2.4Mb). 그림 상의 한 점은 한 Flow에 대한 측정결과를 의미하며 x축은 Flow의 정확한 패킷의 양이고 y축은 측정된 Flow의 양이다. 따라서 선 y=x에 가까울수록 측정이 정확하다. Cuckoo hashing은 세 개의 테이블과 세 개의 해시함수를 사용한다.
Fig. 4. Estimation result of RCC with cuckoo hashing(Counter B 2.4 Mb). Each point in figure points estimation result of one flow. X-Axis means actual packet number of one flow and Y-Axis means estimated packet number. More closer point to y=x means more actual flow.

표 3. Counter B의 load factor에 따른 실험 결과
Table 3. Results of experiments according to load factor of Counter B

Load factor	0.7		0.8		0.9	
Memory	3.1 Mb		2.7 Mb		2.4 Mb	
Counter B	Quadratic	Cuckoo	Quadratic	Cuckoo	Quadratic	Cuckoo
Total	2,233,228	2,052,133	2,729,719	2,250,475	4,162,448	2,387,360
Insertion fail	0	0	0	0	0	0
Average access per operation	1.51	1.39	1.85	1.53	2.82	1.62
Max access for one insertion	36	23	60	29	124	317

에 2.4Mb 메모리 공간을 할당 했을 때의 측정결과를 보여주고 있다. 이때 주어진 데이터 셋에 의해 Counter B의 Load Factor은 0.9에 달했지만 삽입의 실패율을 0으로 기록 했으며 높은 측정 정확도를 보여주었다. [그림 4]는 Cuckoo Hashing을 사용했을 때의 측정결과를 보여주고 있다. Quadratic Probing과 같은 측정 정확도

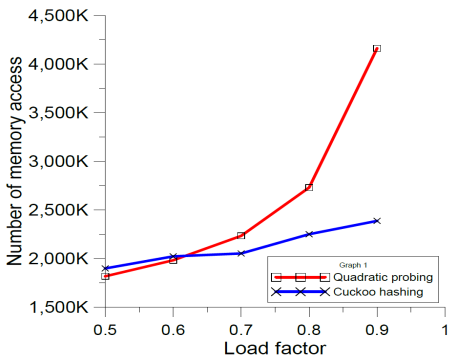

그림 5. Counter B의 load factor에 따른 메모리 총 액세스 횟수
Fig. 5. Total amount of memory access according to load factor of Counter B

를 보여주고 있으며 Load Factor가 0.9인 상황에서도 실패율을 0으로 기록했다. 따라서 두 기법이 Load Factor 0.9까지 정확하게 동작한다는 것을 알 수 있었으며, Cuckoo Hashing이 세 개의 테이블과 세 개의 해시함수를 사용해도 메모리의 90%까지 사용할 수 있다는 것을 알 수 있었다.

[그림 5]는 주어진 데이터 셋이 Counter B에 누적되며 Load Factor가 0.5부터 0.9 사이가 되도록 Counter B의 메모리 할당 크기를 2.4Mb부터 4.4Mb 사이에서 조절하며 진행한 실험결과를 보여준다. Quadratic Probing과 Cuckoo Hashing은 Load Factor 0.6(3.6 Mb)까지 비슷한 메모리 액세스 횟수로 모든 측정을 할 수 있었다. 그러나 Counter B에 3.1Mb(0.7)의 메모리 공간이 주어졌을 때 Quadratic Probing은 Cuckoo Hashing보다 181,095번의 메모리 액세스를 더 했으며 2.4Mb(0.9)의 메모리 공간이 주어졌을 때는, 메모리 액세스 횟수 차이가 1,775,088로 급증했다([표 3]). Quadratic Probing에서 메모리 액세스 할 때마다 식(6) 연산과 키의 비교연산을 해야 되기 때문에 특히 Load Factor가 큰 상황에서는 많은 오버헤드(Overhead)가

발생한다. 또한 [표 3]에 따르면 Counter B의 메모리를 줄여도 Cuckoo Hashing의 메모리 액세스 횟수는 선형적으로 증가하며 Load Factor가 0.9인 상황에서도 좋은 성능을 보여준다. 따라서 Cuckoo Hashing은 메모리 사용량이 높을수록 Quadratic Probing보다 더 적은 연산으로 트래픽을 측정할 수 있다는 것을 보여준다.

V. 결 론

이 논문에서는 RCC(recyclble counter with confinement)의 특성에 대해 분석 및 재현하고 실험을 통해 Quadratic Probing기법을 이용했을 때 RCC의 성능이 저하되는 것을 확인하였다. 그리고 이 문제를 해결하기 위해서 Cuckoo Hashing을 RCC에 적용했다. 그리하여 Quadratic Probing과의 비교를 통해 Cuckoo Hashing이 작은 메모리 공간 또는 메모리 사용률이 높은 상황에서 더 적은 연산 또는 메모리 액세스로 트래픽을 측정할 수 있다는 것을 증명했다. 이를 통해 Flow에 대한 누적 또는 실시간 조회가 자주 발생하는 네트워크 트래픽 측정에서는 RCC에 Quadratic Probing을 사용하는 것보다 Cuckoo Hashing을 사용하는 것이 더 적합하다는 것을 알 수 있었다.

References

[1] Cisco, *Cisco Visual Networking Index: Forecast and methodology, 2012–2017*, Retrieved May, 29, 2013, from http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html.

[2] Cisco, *NetFlow systems*, from http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home. html.

[3] InMon Corp, *sFlow accuracy & billing*, Retrieved 2003, from http://www.sflow.org/sFlowOverview.pdf.

[4] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," in *Proc. IEEE INFOCOM*, pp. 1-9, San Diego, USA, Apr. 2010.

[5] A. Kumar, J. JimXu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," in *Proc. IEEE INFOCOM*, pp. 1762-1773, Hong Kong, China, Mar. 2004.

[6] 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*, pp. 121-132, Annapolis, USA, Jun. 2008.

[7] Y. Lu and B. Prabhakar, "Robust counting via counter braids: An error-resilient network measurement architecture," in *Proc. IEEE INFOCOM*, pp. 522-530, Rio de Janeiro, Brazil, Apr. 2009.

[8] T. Li, S. Chen, and Y. Ling, "Fast and compact per-flow traffic measurement through randomized counter sharing," in *Proc. IEEE INFOCOM*, pp. 1799-1807, Shang Hai, China, Apr. 2011.

[9] K. Y. Whang, B. Vander-Zanden, and H. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208-229, Jun. 1990.

[10] M. Yoon, T. Li, S. Chen, and J. K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Trans. Netw.*, vol. 19, no. 5, pp. 1253-1264, Oct. 2011.

[11] D. H. Nyang and D. O. Shin. "Recyclable counter with confinement for real-time per-flow measurement," *IEEE/ACM Trans. Netw.*, Jan. 2016.

[12] R. Pagh and F. F. Rodler. "Cuckoo hashing" *J. Algorithms*, vol. 51, no. 2, pp. 122-144, May 2004.

[13] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, pp. 323-336, Pittsburgh, USA, Aug. 2002.

[14] R. Karp, S. Shenker, and C. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, no. 1, pp. 51-55, Mar. 2003.

[15] N. Kamiyama and T. Mori, "Simple and accurate identification of high rate flows by packet sampling," in *Proc. IEEE INFOCOM*, pp. 1-13, Barcelona, Spain, Apr. 2006.

[16] X. Dimitropoulos, P. Hurley, and A. Kind,

"Probabilistic lossy counting: An efficient algorithm for finding heavy hitters," *Comput. Commun. Rev.*, vol. 38, no. 1, pp. 7-16, Jan. 2008.

[17] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proc. ACM SIGMOD*, pp. 241-252, San Diego, USA, Jun. 2003.

[18] P. Flajolet and G. Nigel Martin, "Probabilistic counting algorithms for database applications," *J. Comput. Syst. Sci.*, vol. 31, pp. 182-209, Apr. 1985.

[19] The Cooperative Association for Internet Data Analysis, *Equinix Chicago data center*, Retrieved Nov. 21, 2013, from http://www.caida.org.

장 룡 호 (Rhong-ho Jang)

2013년 8월 : 인하대학교 컴퓨터정보공학과 졸업
2015년 8월 : 인하대학교 컴퓨터정보공학과 석사
2015년 9월~현재 : 인하대학교 컴퓨터정보공학 박사과정
<관심분야> 네트워크 보안, 정보보호, 무선 인터넷 보안

정 창 훈 (Chang-hun Jung)

2014년 9월~현재 : 인하대학교 컴퓨터정보공학 석사과정
<관심분야> 정보보호, 인증 프로토콜, 금융 보안, IoT

김 근 영 (Keun-young Kim)

2015년 2월 : 인하대학교 컴퓨터정보공학과 학사
2015년 3월~현재 : 인하대학교 컴퓨터정보공학 석사과정
<관심분야> 정보보호, 네트워크 보안, 암호이론

양 대 헌 (Dae-hun Nyang)

1994년 2월 : 한국과학기술원 과학기술 대학 전기 및 전자공학과 졸업
1996년 2월 : 연세대학교 컴퓨터과학과 석사
2000년 8월 : 연세대학교 컴퓨터과학과 박사
2000년 9월~2003년 2월 : 한국전자통신연구원 정보보호연구본부 선임연구원
2003년 2월~현재 : 인하대학교 컴퓨터정보공학과 교수
<관심분야> 암호이론, 암호프로토콜, 인증프로토콜 무선 인터넷 보안, 네트워크 보안

이 경 희 (Kyung-hee Lee)

1993년 2월 : 연세대학교 컴퓨터과학과 학사
1998년 8월 : 연세대학교 컴퓨터과학과 석사
2004년 2월 : 연세대학교 컴퓨터과학과 박사
1993년 1월~1996년 5월 : LG 소프트(주) 연구원
2000년 12월~2005년 2월 : 한국전자통신연구원 선임연구원
2005년 3월~현재 : 수원대학교 전기공학과 부교수
<관심분야> 바이오인식, 정보보호, 컴퓨터비전, 인공지능, 패턴인식