Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.3, Jun. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.3.427

Count-Min HyperLogLog : 네트워크 빅데이터를 위한 카디널리티 추정 알고리즘

강 신 정,[1†] 양 대 헌[2‡]
[1,2]이화여자대학교 (대학원생, 교수)

Count-Min HyperLogLog : Cardinality Estimation Algorithm for Big Network Data

Sinjung Kang,[1†] DaeHun Nyang[2‡]
[1,2]Ewha Womans University (Graduate student, Professor)

요 약

카디널리티 추정은 실생활의 많은 곳에서 사용되며, 큰 범위의 데이터를 처리하는 데 근본적 문제이다. 인터넷이 빅데이터의 시대로 넘어가며 데이터의 크기는 점점 커지고 있지만, 작은 온칩 캐시 메모리만을 이용하여 카디널리티 추정이 이뤄진다. 메모리를 효율적으로 사용하기 위해서, 지금까지 많은 방법이 제안되었다. 그러나, 이러한 알고리즘에서는 estimator 간의 노이즈 발생으로 인해 정확도가 떨어지는 일이 발생한다. 이 논문에서는 노이즈를 최소화하는데 중점을 뒀다. 우리는 여러 개의 데이터 구조를 제안하여 각 estimator가 데이터 구조 수만큼의 추정값을 가지고, 이 중 가장 작은 값을 선택하여 노이즈를 최소화한다. 실험을 통해 이 방법이 이전의 가장 좋은 방법과 비교했을 때, 플로우당 1 bit와 같은 작은 메모리를 사용하면서 더 좋은 성능을 보이는 것을 확인했다.

ABSTRACT

Cardinality estimation is used in wide range of applications and a fundamental problem processing a large range of data. While the internet moves into the era of big data, the function addressing cardinality estimation use only on-chip cache memory. To use memory efficiently, there have been various methods proposed. However, because of the noises between estimator, which is data structure per flow, loss of accuracy occurs in these algorithms. In this paper, we focus on minimizing noises. We propose multiple data structure that each estimator has the number of estimated value as many as the number of structures and choose the minimum value, which is one with minimum noises, We discover that the proposed algorithm achieves better performance than the best existing work using the same tight memory, such as 1 bit per flow, through experiment.

Keywords: Big Network Data, Cardinality Estimation, Sketch

I. 서 론

카디널리티 추정은 네트워크 보안에서 Denial-of-Service (Dos) 공격, 링크 기반 스팸, 스캐닝 공격 등의 사이버 공격을 막기 위해서 근본적인 문제이다[2,4,6,10,12,14]. 카디널리티는 정해진 시간 동안 서로 다른 원소의 개수를 추정한 값으로 정의되는데, 여기서 원소는 destination IP 주소, source IP 주소, 또는 두 IP 주소의 쌍 등이 될 수 있다.

최근의 high-end 라우터들은 수백 GB/s, 심지어는 수 TB/s의 속도로 패킷을 보내고 있는데, 이

Received(02. 17. 2023), Modified(04. 19. 2023), Accepted(05. 12. 2023)
† 주저자, kangsin1110@ewhain.net
‡ 교신저자, nyang@ewha.ac.kr(Corresponding author)

결과 수억 개에 달하는 플로우가 코어 라우터를 거친다[1]. 한편, 라우터들이 스위치 패브릭을 통해 수신 포트에서 송신 포트로 패킷을 보낼 때 메인 메모리와 CPU를 우회하여 보낸다. 이때 일반적으로 네트워크 프로세서 칩에 사용되는 캐시 메모리는 SRAM인데, 크기가 보통 몇 MB 정도이며, 이는 측정, 성능과 같은 라우터의 다른 기능들과 공유하여 사용된다. 따라서, 카디널리티 추정에서 가장 중요한 문제는 메모리의 제한 하에 매우 큰 범위의 각 플로우의 개수를 세는 것이다.

메모리를 제한적으로 사용하면서 개수가 많은 플로우의 개수를 다루기 위해 지금까지 여러 개의 기법들이 제안되었는데, Multi-resolution Bitmap[2], MinCount[3], RRSE[19], PCSA[11], LogLog[5], HyperLogLog[13], 그리고 virtual HyperLogLog(vHLL)[7] 등이 제시안이다. 각 방법은 플로우당 개별의 데이터 구조를 할당하는데, 이를 estimator라 한다. 추정량은 여러 개의 비트맵, register, 또는 기초적인 데이터 구조로 이루어져 있다.

제시된 방안 중 특히 HyperLogLog 기반 알고리즘 vHLL은 앞에서 언급한 다른 방법들보다 더 정확하고 효율적이라는 것이 증명되었다. vHLL의 각 estimator는 메모리를 공유하는데, 비트맵과 PCSA 같은 이전에 나왔던 메모리를 공유하는 방식을 이용한 알고리즘들과는 다르게, vHLL은 register 측면에서 공유를 적용하였다. 결과적으로, vHLL은 공유로 인해 발생하는 각 estimator 간의 노이즈를 줄여 효과적으로 카디널리티를 측정한다.

기여:

vHLL이 이전의 시도들 중 가장 좋은 성능을 보이면서 플로우당 카디널리티 측정에서 노이즈 감소가 중요한 문제로 떠올랐다. 이 논문에서 우리는 노이즈 감소에 중점을 둔 vHLL 기반의 Count-Min HyperLogLog(CMHLL) 방안을 제시한다. 뒤의 단원에서 자세하게 다룰 CMHLL은 여러 개의 register로 이뤄진 데이터 구조 여러 layer를 사용하여, 작은 메모리를 사용하면서 노이즈 발생의 확률을 높인다.

해당 논문은 다음과 같이 구성된다: 단원 2에서는 알고리즘을 제시하게 된 동기를 다루고, 단원 3은 데이터 구조 및 encoding, decoding의 디자인 및 pseudo code와 함께 CMHLL에 대한 자세한 설명을 보인다. 단원 4에서는 실제 인터넷 traces들을 기반으로 진행한 실험을 통해 제시한 알고리즘의 성능을 평가하고, 단원 5는 related work를 소개한다. 마지막으로, 단원 6에서는 결론을 정리한다.

II. 동 기

최근 사이버 공격의 크기가 점차 증가하면서 심각한 문제로 떠올랐다. 이에 따라 다양한 사이버 공격 탐지에 이용될 수 있는 카디널리티 추정의 중요성이 강조되고 있다. 예를 들어, 같은 destination IP 주소에서 오는 모든 패킷을 하나의 플로우라 할 때, 각 플로우의 서로 다른 source IP 주소의 개수를 세는 카디널리티 추정 모듈이 있다. 특정 플로우의 카디널리티가 급증하게 되면, 해당 플로우는 DDos 공격으로 취급될 수 있다. 또 다른 예시로는, 방화벽이나 게이트웨이의 카디널리티 추정 모듈은 source IP 주소당 고유한 destination IP 주소의 개수를 세서 스캐닝 공격을 탐지한다.

또한, 사이버 공격 이외에도 카디널리티 추정은 다양하게 응용될 수 있다. 게이트웨이에서 각 URL에 대한 고유의 request수를 추적하여 해당 URL의 컨텐츠에 대한 인기를 측정하는데 사용되기도 하고, 수천 개의 컴퓨터로 이뤄진 서버 팜에서 각 파일에 접근하는 서로 다른 사용자들의 수를 추정해서 파일의 선호도를 알아낼 수 있다. Google과 같은 기업에서도 일 단위로 나오는 데이터셋에 대해 cardinality 추정 알고리즘을 사용하여 실험을 진행한다[10].

한편, 인터넷이 빅데이터의 시대로 이동하면서 아주 큰 범위($\sim 10^9$)의 카디널리티 추정이 필요해졌지만, 사용할 수 있는 메모리의 제한으로 인해 메모리가 효율적으로 사용되어야 한다. 이전의 제안되었던 방안 PCSA, vHLL이 각 estimator가 메모리를 공유하는 방법을 기반으로 메모리를 효율적으로 사용했는데, 메모리 공유는 estimator 간의 노이즈가 일어나기 때문에 정확도가 낮아진다는 단점이 있다. 이에, 우리는 estimator 간의 노이즈를 줄이는 데에 주목했다.

Count-Min HyperLogLog, CMHLL은 vHLL과 마찬가지로 register 측면의 메모리 공유를 기반으로 하지만, estimator 간의 공유가 일어나는 데이터 구조를 여러 개 사용한다. 각 구조는 메모리가 할당되어 있어, 구조 간의 메모리 공유는 일어나지 않는다. 각 데이터 구조에서 플로우당

estimator가 다른 register를 선택하여 값을 추정하므로, estimator는 데이터 구조 수만큼의 추정값을 가지게 되는데, 이때 값 중 가장 작은 값을 선택하여 노이즈 제거 가능성을 높인다.

III. CMHLL

이 단원에서는 여러 개의 layer로 구성된 multi-tenant 카디널리티 추정 알고리즘 CMHLL을 소개한다. 먼저 CMHLL의 데이터 구조를 소개하고, 간략한 공식을 통해 CMHLL이 노이즈를 어떻게 최소화하는지 설명한 후, encoding 및 decoding 과정을 알고리즘 코드를 사용하여 설명한다. Table 1.은 해당 논문에서 사용된 notations

Table 1. Notations

l	number of layers
$H()$	hash function that selects $M_n[]$ using layer number
$M_n[]$	register array
N	size of $M_n[]$
$M_{nf}[]$	virtual register array in $M_n[]$
s	size of $M_{nf}[]$
$hash_i()$	hash function that maps i_{th} element of $M_{nf}[]$ in $M_n[]$
n_f	real cardinality of f
$\hat{n_f}$	estimated cardinality of f

를 보여준다.

3.1 데이터 구조

Fig. 1.은 register array $M_n[]$들로 채워진 여러 개의 layer로 이루어진 CMHLL의 전체적인 구조를 보여준다. $M_n[]$의 크기는 N이고, 각 register는 5 bits이다. 플로우당 estimator는 각 layer당 hash 함수 $H()$를 통해 각각 다른 register array를 선택한다. 이후, 선택된 array 안에서 s개의 register를 사용하는데, 이는 virtual register array $M_{nf}[]$라 한다. Fig. 2.는 하나의 layer 내에서의 encoding 및 decoding 과정을 간략히 보여준다. Register array 내에서 플로우가 들어왔을 때, $hash()$ 함수를 통해 register index 및 플로우의 hash값을 얻는다. 해당 index에서 원래 값보다 큰 값이 들어오게 되면 update한다. 이후 decoding 과정에서는 register 안의 값을 가지고 카디널리티를 추정한다. 해당 과정은 3.2, 3.3에서 자세히 다룬다.

다음으로는 CMHLL의 데이터 구조와 함께 해당 알고리즘의 노이즈 최소화 방법에 대해 다룬다. CMHLL은 vHLL과 마찬가지로 카디널리티 크기에 따라 [5], [7] 또는 [8]의 방법을 이용하여 추정값을 도출한다. 그런데 실제 네트워크 trace와 같은 빅데이터에 대해 vHLL을 이용하여 추정하면 overestimation이 일어난다[19]. Count-Min HyperLogLog는 이러한 한계점을 개선하기 위해

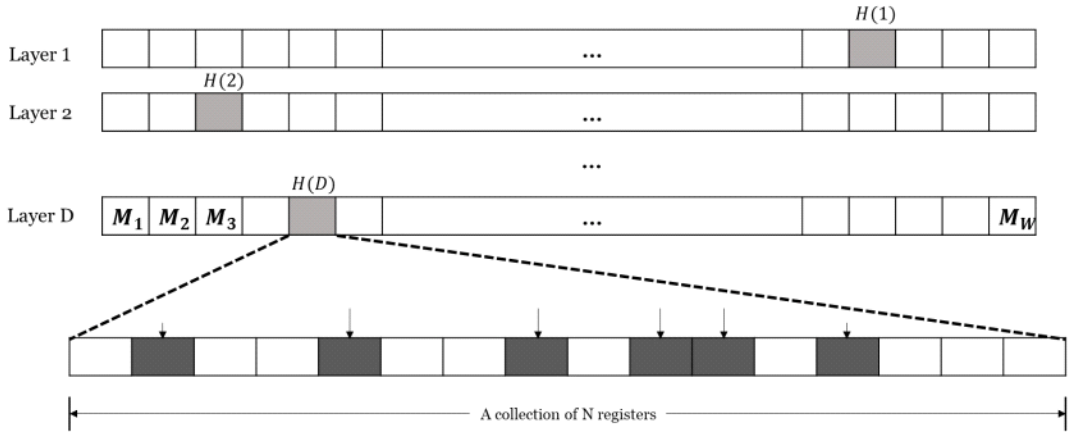

Fig. 1. The data structure of CMHLL

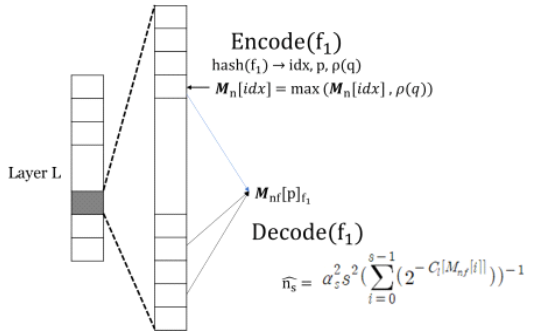

Fig. 2. Form of encoding & decoding of CMHLL

Fig 1.에 나온 것처럼 여러 개의 layer를 사용하는 Count Min Sketch[20]의 방법을 사용했다. 이는 각 layer에서 overestimation이 일어나더라도 그 중 가장 작은 값을 선택하여 overestimation으로 발생하는 노이즈를 최소화하는데, 해당 알고리즘도 동일한 방식으로 vHLL의 한계를 보완했다.

3.2 Encoding

Fig. 3.은 CMHLL의 encoding 과정을 보인다. 각 (scrIP,dstIP) 쌍에 대해 scrIP+dstIP 값을 먼저 hash한 후 패킷의 hash값 h_f는 Algorithm 1의 4번째 줄의 p, q로 나눈다. 이때 p는 h_f의 binary 형태의 앞의 b bits를, q는 뒤의 남은 bits이다. 이후 register array의 register를 선택하기 위해 dstIP를 hash한다. 여기까지 encoding 방식은 vHLL과 비슷하지만, CMHLL은 register array N개로 된 layer 여러 개로 이뤄져 있으므로, layer 번호와 scrIP를 더한 결과를 hash하여 각 layer에서 register array $M_n[]$를 선택한다. $\rho(q)$는 q의 앞의 0의 개수에 1을 더한 값(rank)으로, 예를 들어 q가 0001…라 하면, $\rho(q)$가 4가 된다. 그러므로 $\rho(q) = i$의 확률은 $(\frac{1}{2})^i (i > 0)$이 된다. array의 선택된 register($M_n[idx]$)의 값은 $\rho(q)$가 register값보다 클 때, update된다. 따라서 앞자리의 0의 개수가 가장 많은 원소의 rank값이 register에 저장된다.

Algorithm 1: Encoding

```
1  inputs : # of layer D, counter array C_l[M[]]
2  forall scrIP, dstIP ←pkt_f do
3      h_f ← hash(scrIP + dstIP);
4      p ← <x_1 x_2 ⋯ x_b>; q ← <x_{b+1} x_{b+2} ⋯ >;
5      idx ← hash(dstIP) % s;
6      for l = 0 to D−1 do
7          n ← H(scrIP + l) % N;
8          C_l[M_n[idx]] := max(C_l[M_n[idx]], ρ(q));
9      end
```

Fig. 3. Encoding algorithm

3.3 Decoding

Fig. 4.는 source IP(scrIP) 기반 decoding 과정을 묘사한다. register $M_{nf}[i]$, $0 \le i < s$에 대해 register 안의 값을 가지기 위해서 $\frac{1}{2^{M_{nf}[i]}}$의 확률을 가진다. E가 estimator $M_{nf}[]$에 의해 추정된 서로 다른 원소의 총 개수라 하면, 이 estimator만의 서로 다른 원소와 해당 estimator 안의 register를 공유하는 다른 estimator의 서로 다른 원소도 포함한다. E를 추정하기 위해서 조화평균을 이용하여 $M_{nf}[]$ 안의 모든 register의 추정값을 종합한다:

Algorithm 2: Decoding

```
1  Function Decoding(scrIP);
2      for l = 0 to D−1 do
3          n := H(scrIP + l) % N;
4          E := α_s^2 s^2 ( Σ_{i=0}^{s−1} (2^{−C_l[M_nf[i]]}) )^{−1};
5          if E < (5/2) s then
6              E := Linear_Counting(M_nf); [8] for details
7          end
8          if E > (1/30) 2^32 then
9              E := −2^32 log(1−E/2^32); [7] for details
10         end
11         return E
12     n̂_f := min(n̂_f, E);
13     return n̂_f
```

Fig. 4. Decoding algorithm

$$E = \alpha_s^2 s^2 (\sum_{i=0}^{s-1} (2^{-C_i[M_{i,j}[i]]}))^{-1} \quad (1)$$

여기서 α_s는 bias correction 상수로, 값은 수식 (2)로 결정된다.

$$\alpha_s = (s \int_0^{\infty} (\log_2 (\frac{2+u}{1+u}))^m du)^{-1} \quad (2)$$

이때, E가 너무 작을 때($E < \frac{5}{2}s$)는 Linear Counting[8]을, E가 너무 큰 값을 가질 때($E > \frac{2^{32}}{30}$)는 수식 (3)을 이용하여 값을 정정한다.

$$E = -2^{32} \log(1 - E/2^{32}) \quad (3)$$

여기서, CMHLL은 여러 개의 layer로 이뤄져 있어, 계산되는 카디널리티 추정값이 여러 개다. CMHLL은 이 중 가장 작은 값을 선택함으로써(12번째 줄), 노이즈를 최소화한다. 이는 sharing을 사용하여 카디널리티를 추정하는 알고리즘의 단점인 overestimation의 노이즈를 최소화하는 방법이다.

IV. 성능 평가

우리는 CMHLL과 해당 알고리즘과 관련이 가장 큰 vHLL[7]에 대해 실험을 진행했다. vHLL은 한 개의 데이터 구조에 대해 register 측면에서 메모리를 공유하고, CMHLL은 메모리를 여러 개의 데이터 구조에 나눠서 할당하여 각각의 구조에 대해 register 측면에서 공유한다. 우리는 CAIDA[9]에서 다운로드 받은 실제 네트워크 trace들을 사용한 실험을 통해 두 알고리즘의 성능을 비교했다. 사용한 trace 파일은 250초 동안 들어온 패킷을 포함한다. 우리는 각 source IP 주소에서 패킷을 보낸 서로 다른 destination IP 주소의 개수를 측정한다. 이는 scanner 공격 탐지 및 서버(source)당 정보를 보내는 사용자(destination)의 수를 추정하여 인터넷 호스트의 인기도를 측정 등 실용적으로 사용될 수 있어[17,18], 해당 실험 결과는 CMHLL이 카디널리티 추정의 정확도를 평가하여 알고리즘의 실용성을 보인다.

4.1 vHLL과의 비교

우리는 CMHLL과 vHLL의 카디널리티 추정의 정확도를 통해 두 알고리즘에 대한 평가를 진행하는데, 공정한 평가를 위해 실험 진행에 사용되는 메모리는 똑같이 할당한다. CMHLL은 layer의 개수 D, virtual register의 크기 N, 그리고 register array 개수 W의 변수가 있는데, 앞의 두 변수의 기본값은 D는 4, N은 32로 하고, 뒤의 실험에서 값을 변화시킨다. W는 실험에 할당된 메모리와 두 변수에 따라 값이 달라진다. 사용한 trace 파일 안의 플로우의 개수는 약 180만 개이고, 메모리는 플로우당 평균적으로 1bit, 2bit, 그리고 5bit를 할당할 수 있도록 1.8Mb, 3.6Mb, 9.0Mb를 사용하고, vHLL도 같은 조건에서 실험했다. vHLL의 변수 register 크기 s는 기본값을 512로 설정, array 크기 m은 할당 메모리에 따라 변한다.

4.1.1 Super Destinations 탐지

해당 실험에서 CMHLL과 vHLL은 super destinations의 탐지를 기반으로 비교했다. 여기서 측정 시간 동안 추정된 서로 다른 destination의 개수, 즉 추정 카디널리티가 500 이상인 경우를 super destinations로 취급한다. 이때 사용하는 지표는 recall, precision, f1-score로, 각 지표의 수식은 다음과 같다.

$$Recall = \frac{tp}{tp+fn} \quad (4)$$

$$\mathrm{Pr}ecision = \frac{tp}{tp+fp} \quad (5)$$

$$F1-score = \frac{2*Recall*\mathrm{Pr}ecision}{Recall+\mathrm{Pr}ecision} \quad (6)$$

여기서 실제값이 기준보다 클 때 추정값도 기준보다 크면 진양성(tp), 그렇지 않으면 거짓 음성(fn)이라 하고, 실제값이 기준보다 작을 때 추정값이 기준보다 크면 거짓 양성(fp), 그렇지 않으면 진음성(tn)라 한다. 각 지표가 1에 가까울수록 알고리즘의 추정 정확도가 높다. 표 2는 할당 메모리의 변화에 따른 CMHLL과 vHLL의 각 지표값을 나타낸다.

Table 2. Recall, Precision, and F1-score with variable per-flow memory

bit per flow	CMHLL			vHLL		
	Recall	Pr*ecision*	*F*1 − *score*	*Recall*	Pr*ecision*	*F*1 − *score*
1	1	0.888889	0.941176	0.875	1	0.933333
2	1	0.888889	0.941176	0.875	1	0.933333
5	1	0.941176	0.969697	0.4375	1	0.608696

플로우당 평균적으로 할당되는 bit에 상관없이 CMHLL은 recall값이 1, vHLL은 precision값이 1이였다. 해당 결과는 할당된 메모리에서 CMHLL은 fn인 플로우가 0, vHLL은 fp인 플로우가 0개임을 의미하는데, 이는 CMHLL은 기준보다 큰 값, 즉 super destinations 탐지율이, vHLL은 기준보다 작은 값에 대한 탐지율이 100% 임을 의미한다. 이를 통해 CMHLL이 큰 cardinality에 대한 탐지 성능이 더 좋다는 것을 알 수 있다. 전체적인 두 알고리즘의 성능 비교를 위해 f1-score를 기반으로 비교해보면, 플로우당 할당 bit가 1,2일 때에는 CMHLL의 f1-score가 0.941, vHLL의 f1-score가 0.933으로 CMHLL의 지표가 더 높았고, 5로 늘어났을 때에는 CMHLL은 0.970로 f1-score값이 증가한 반면, vHLL은 0.601로 값이 감소했다. 해당 실험 결과는 f1-score값을 통해 할당 메모리가 1,2, 그리고 5에서 지속적으로 CMHLL이 vHLL보다 super destinations 탐지를 포함한 전체적 성능에 대해 우수함을 확인할 수 있다.

4.2 Layer 개수의 영향

이 단원에서는 layer 개수에 따른 추정 정확도를 평가한다. 메모리는 각 플로우별 할당이 약 2bit를 할당되도록 설정하고, layer 개수는 2, 3, 그리고 5로 다르게 하여 Recall, Precision, 그리고 f1-score값을 비교했다. 표 3은 해당 실험 결과를 보인다. layer 개수가 늘어날 때, 노이즈 발생 확률이 낮아지므로, 추정 정확도를 보이는 지표가 더 높은 값을 가졌다. 개수에 상관없이 recall값은 1로, super destinations 탐지 성능이 뛰어남을 확인할 수 있다. 그러나 성능을 나타내는 다른 지표인 precision이 1보다 작았는데, 이는 layer 개수가 늘어남에 따라 f1-score값과 함께 증가했다. d가 2,3일 때는 precision값이 0.762, f1-score값이 0.865인 반면, 5일 때 precision값이 0.889, f1-score값이 0.941였다. 이러한 결과를 통해 layer 개수가 2,3일 때보다 5개일 때 노이즈가 적게 발생하여 탐지 성능이 개선되는 것을 확인할 수 있다.

Table 3. Recall, Precision, F1-score with different d

number of layers(d)	*Recall*	Pr*ecision*	*F*1 − *score*
2	1	0.761905	0.864865
3	1	0.761905	0.864865
5	1	0.888889	0.941176

V. Related work

카디널리티 추정은 flow-size 추정과는 다르다. 예를 들어 같은 source IP 주소에서 같은 destination IP 주소에 천 개의 패킷을 보내는 상황을 보자. 플로우의 크기는 패킷 개수이므로, 이 플로우의 크기가 1000인 반면, 카디널리티는 서로 다른 원소의 개수를 의미하기 때문에 1이다. 따라서 flow 크기 추정은 단순히 counter를 필요로 하는 반면, 카디널리티 추정은 같은 패킷을 제거해야 하므로 각 원소의 저장이 필요하여 더 복잡한 문제이다. 그러나 각 estimator가 모든 원소를 hash table에 저장하면 메모리가 지나치게 많이 소모된다. 이에 카디널리티를 추정하는 알고리즘들이 제안되었다.

5.1 Linear Counting

Linear Counting은 bitmap을 이용하여 카디널리티를 추정하는 방법인데, 초기에 모든 bit는 0으로 설정한다. 각 원소가 특정 bit에 hash되면, bit는 1로 바뀐다. 이때, 같은 패킷이 들어오면 같은 bit에 할당되므로, 중복은 세지 않는다. 카디널리티 추정 시 수식 (7)을 사용한다.

$$\hat{n} = -mln(V) \quad (7)$$

여기서 $\hat{n}$은 추정된 카디널리티, m은 bitmap의 크기, V는 bitmap 안에서 0의 비율이다. 이 방법의 단점은 $-mln(m)$이 최대값이라는 것이다. 즉, 큰 플로우가 들어올 때 카디널리티를 제대로 세기 위해서는 bitmap의 크기가 아주 커야 한다. 이외에도 bitmap을 사용하는 방법은 다양하게 제안되었다 [14-16].

5.2 Multi-resolution Bitmap과 PCSA

이에 큰 플로우를 다룰 수 있는 샘플링 기법이 사용된 방법들이 제시되었는데, Multi-resolution Bitmap과 PCSA이다. 첫 번째 방안은 여러 개의 bitmap을 사용하여 각각의 bitmap이 샘플링 확률을 가지게 한다. 여기서 bitmap의 크기를 최소화하면 PCSA에서 사용하는 register가 되는데, bitmap의 개수를 w개라 할 때, register의 크기는 w가 되고, 추정값은 최대 2^w까지 될 수 있다. 그러나 register가 1개일 때 PCSA의 성능은 매우 좋지 않아서 많은 수의 register를 필요로 한다.

5.3 LogLog, HyperLogLog, 그리고 vHLL

LogLog와 HyperLogLog는 최대 추정값이 2^w일 때 register의 크기가 $\log_2(w)$로 축소된 방법이다. 따라서 두 기법과 PCSA 모두 register를 사용하지만, 두 기법이 더 좋은 정확도를 보인다. 두 기법의 차이는 카디널리티 추정 시에 LogLog는 정규평균을 이용하는 반면, HyperLogLog은 조화 평균을 사용하여 LogLog보다 정확하다.

vHLL은 HyperLogLog를 기반으로 하는 알고리즘으로, 기존의 제안된 방법들보다 좋은 성능을 보인다. 이는 각 estimator가 virtual register를 사용하면서 estimator 간의 register를 공유한다. 이전에 나온 방법 중 가장 좋은 방법인 HyperLogLog를 기반으로 하여, 메모리 효율적이고 높은 정확도를 보이는 알고리즘이다.

VI. 결 론

해당 논문에서는 효율적으로 카디널리티를 추정하는 알고리즘 Count-Min HyperLogLog (CMHLL)을 제시하고, framework를 통해 이 방안의 추정 및 노이즈 최소화 방법에 대해 자세히 설명한다. 실험을 통해 vHLL과 성능을 비교했고, vHLL과 마찬가지로 작은 메모리를 사용하지만 좋은 성능을 보인다는 것을 확인했다. 이때 vHLL보다 super destinations 탐지율이 높다는 것을 통해 빅데이터 추정 시 적합한 알고리즘임을 확인했지만, underestimation이 일어나는 한계점을 발견했다. 제시된 방안은 프로세서 칩의 메모리 할당량 정도로 현대 라우터의 빠른 패킷 처리 속도에 뒤처지지 않게 카디널리티를 추정하여 빅데이터를 효과적으로 처리하는데 사용될 수 있을 것이다. 향후 연구에서는 해당 연구에서 사용한 추정 방법과 여러 개의 layer를 이용하여 추정값을 도출하는 방식을 이용하되, 가장 작은 값이 아닌 중간값이나 평균값을 사용함으로써, 해당 알고리즘의 underestimation 문제를 해결하고자 한다.

References

[1] FOSCO, "Researchers transmit optical data at 16.4 Tbps," http://www.informationweek.com/researchers-transmit-optical-dataat-164/206900603, Dec. 2022

[2] C. Estan, G. Varghese, and M. Fish, "Bitmap algorithms for counting active flows on high-speed links," Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, pp. 153 - 166, Oct. 2003.

[3] Z. Bar-yossef, T. S. Jayram, R. Kumar, D. Sivakumar and L. Trevisan, "Counting distinct elements in a data stream," Lecture Notes in Computer Science, RANDOM 2002, LNCS, vol. 2483, pp. 1 - 10, Jan. 2002.

[4] C. Estan and G. Varghese. "New directions in traffic measurement and accounting," Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 323

- 336, Aug. 2002.
[5] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," ESA: European Symposia on Algorithms, LNCS, vol. 2832, pages 605 - 617, Apr. 2003.
[6] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," Proc. of IEEE INFOCOM, pages 1 - 9, Mar. 2010.
[7] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 417 - 428, Jun. 2015,
[8] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," ACM Transactions on Database Systems, vol. 15, no. 2, pp. 208 - 229, Jun. 1990.
[9] CAIDA, "dataset:passive_2013_pcap," http://www.caida.org/data/passive/passive_2013_dataset.xml, Jan. 2023
[10] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: Algorithmic engineering of a state-of-the-art cardinality estimation algorithm," Proceedings of the 16th International Conference on Extending Database Technology, pp. 683 - 692, Mar. 2013.
[11] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for database applications," Journal of computer and system sciences, vol. 31, no. 2, pp. 182-209, Apr. 1985.
[12] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," IEEE INFOCOM 2009, pp. 504-512, Apr. 2009.
[13] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," Proc. of AOFA: International Conference on Analysis Of Algorithms, pp.137-156, Jun. 2007.
[14] Q. Zhao, J. Xu, and A. Kumar, "Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation," IEEE Journal on Selected Areas in Communications, vol. 24, no. 10, pp. 1840-1852, Oct. 2006.
[15] Q. Xiao, Y. Qiao, M. Zhen, and S. Chen, "Estimating the persistent spreads in high-speed networks," 2014 IEEE 22nd International Conference on Network Protocols, pp. 131-142, Oct. 2014.
[16] Q. Xiao, B. Xiao, and S. Chen, "Differential estimation in dynamic RFID systems," Proceedings IEEE INFOCOM, pp. 295 - 299, Apr. 2013.
[17] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-end containment of internet worms," Proceedings of the twentieth ACM symposium on Operating systems principles, vol. 39, no. 5, pp. 133-147, Oct. 2005.
[18] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for internet worms," Proceedings of the 10th ACM conference on Computer and communications security, pp. 190 - 199, Oct. 2003.
[19] Dinh Nguyen Dao, Rhongho Jang, Changhun Jung, David Mohaisen, DaeHun Nyang, "Minimizing Noise in HyperLogLog-Based Spread Estimation of Multiple Flows," 2022 52nd Annual IEEE/IFIP International

Conference on Dependable Systems and Networks (DSN), pp. 331-342, Jun. 2022

〔20〕 G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," Journal of Algorithms, vol. 55, no. 1, pp. 58 - 75, 2005.

〈저자소개〉

강 신 정 (Sinjung Kang) 학생회원
2021년 8월: 이화여자대학교 물리학과 졸업
2021년 9월~현재: 이화여자대학교 인공지능 소프트웨어 학부 사이버보안 전공 석사과정
〈관심분야〉 정보 보호, 네트워크 보안, AI 보안

양 대 헌 (DaeHun Nyang) 종신회원
1994년 2월: KAIST 전자공학과 졸업
1996년 2월: 연세대학교 컴퓨터과학과 석사
2000년 8월: 연세대학교 컴퓨터과학과 박사
2003년 3월~2020년 2월: 인하대학교 컴퓨터공학과 교수
2020년 3월~현재: 이화여자대학교 소프트웨어학부 교수
〈관심분야〉 AI 보안, 네트워크 보안, 시스템 보안, 소프트웨어 보안