Super spreaders estimator in compact memory

다오딩응웬* · 양대헌*

*인하대학교 정보보호연구실

DinhNguyen Dao · DaeHun Nyang

*Department of Computer Science & Information Technology, Inha University

Abstract

Estimating the number of distinct values is well-known problem in network data measurement and many effective algorithms are suggested. Recent works have built upon technique called Linear Counting to solve the estimation problem for massive sets or spreaders in small memory. But it has a drawback when having limited upper bound estimation. In this paper, we propose a per-flow cardinality estimation technique that is using adaptive sampling with randomness sharing counter. Our per-flow measurement scheme called ASSE has some improvements over existing counting schemes such as faster decoding, huge range of estimation. In this dissertation, we presented mathematical analysis, conducted experiment with real data, and compared the results of these previous schemes.

I. Introduction

In network traffic measurements area, counting distinct value plays a vital role in many applications. The values can be source addresses, destination addresses, port number or a protocol. In this paper, we focus on the spreader which has contact with numerous destination addresses. High spreader can be used as an indicator of a DDoS attack, a worm propagation, a port or address scanner. Thus, it is necessary to find out the spreads or the number of distinct values. Certainly, the spread estimation algorithm can be extended to other fields that need to count the distinct value.

There have been many researches for distinct counting or spreader estimation. Some of them count for one flow only such as [1-2]. The others works only classify the high spreaders which are greater than a defined threshold and skip the small ones. Recently, Yoon [3] and Li [4] have a decent approach to estimate multi-spreaders. All the data is stored in a single memory by a random mapping process using hash functions. However, the limitation of this approach is their small range of estimation. Thus, we came with new solution that adopted from HyperLogLog (HLL) algorithm [2]. Our per-flow measurement scheme, called Adaptive Sampling Spreader Estimator (ASSE), has some improvement over existing counting schemes such as faster decoding, huge range of estimation.

This paper is organized into 5 sections. The proposed scheme is discussed in Section II. Experimental results are presented in Section III. Section IV summarizes the results and draws the conclusion.

II. Adaptive Sampling Spreader Estimation (ASSE)

2.1 Package Encoding

The idea of our encoding algorithm is that

we use an array of s registers for each source spreader estimation like in HLL. But instead of storing the registers individually which requires enormous memory space for all spreaders, we store on virtual vector which will be mapped to big array of size m. The position of registers k of spreader $d \in D$ are mapped to position in memory M by hash function H which has following character. $H: D \times [0,s-1] \rightarrow [0,m-1]$.

The encoding algorithm works as follow. Wherever a packet come in, we get the hash value of destination IP. the virtual register index i is set by the first b bits of hash value. The virtual vector mapped to the memory by hashing source and index i $H(src,i)$. While the value of this register is set to maximum leading zeros of last c bits of hash value, denoted as $\rho(w)$ when old value is smaller than the new one that is adapted from HLL. The number of counters which have the same value is stored in C_M for decoding. The algorithm is summarized in Alg. 1.

Algorithm 1: The Encoding Process

```
1  Let m is the total number of registers, each register store value in range of [0..c].
2  Let Cm is the array counter for all registers in memory.
3  Let number of registers for 1 spreader s = 2^b, b ∈ [4..16]
4  Let Cs is the array counter for registers of one spreader.
5  Let Hs : S → {0,1}^32 hash data from D to 32-bit words.
6  Let Hd : D × [0,s − 1] → [0,m − 1]
7  Initialize M[0..m − 1] to 0.
8  Initialize Cm[0]..Cm[c] to 0.
9  Initialize Cs[0]..Cs[c] to 0.
10 function Encode(src, dest)
11   for each src do
12     x := Hd(dest)
13     i :=< x1x2...xb >2  //register index
14     w :=< xb+1xb+2...xb+c >2  //register value
15     idx := H(src, i)
16     if M[idx]<ρ(w) then // update counter
17       CM[M[idx]] − −
18       CM[ρ(w)] + +
19       M[idx] := ρ(w)
20     end if
21   end for
22 end function
```

The encoding procedure works identically fast. Only 2 hash functions, 1 leading zeros bit finding function and compare function are definitely called. Some read, write memory to update the register and its counters are only called when registers updated.

3.2 Spreader Decoding

The spreader estimation for the given source IP is calculated from the value get from its registers group. This decoding algorithm is based on the decoding process in HLL. But before do the estimation, the RegisterNormalize function is called to reduce the noise caused by sharing register by multi spreaders. The decoding process is described in Alg. 2.

In RegisterNormalize function, let $C_S[]$ and $C_M[]$ is the counter array of s registers

Algorithm 2: The Decoding Process

```
1  Define α16=0.673; α32=0.697; α64 =0.709;
2  αm=0.7213 = (1+1.079/m) for m >= 128
3  function Decode(src)
4    for i = 0 to s-1 do //count the number of counter which has same value
5      index = H(src,i)
6      CM[M[index]]++
7    end for
8    RegisterNormalize(CS[],CM[])
9    E := αm.m^2. • (Σ_{i=1}^{c} 2^{-i})^{-1}
10   if E<= 5m/2 then
11     Let V be the number of counter = 0
12     if V ≠ 0 then
13       E* := log(m=V)
14     else
15       E* := E
16     end if
17   else if E <= 2^32/30
18     E* := E
19   else
20     E* := -2^32.log(1 - E/2^32)
21   end if
22   return E*
23 end function
```

Algorithm 3: Auxiliary procedure for removing noise from registers

```
1  function RegisterNormalize(CS[],CM[])
2    Let c is maximum value of counters
3    Set CS0[0...c] to 0
4    for i=0 to c do
5      dec := 0
6      inc := 0
7      for h=0 to i-1 do
8        inc = inc + CS0[h]
9      end for
10     for j=i+1 to c do
11       dec = dec + CM[j]
12     end for
13     CS0[i] := (m.CS[i] - inc.CM[i])/(m-dec)
14   end for
15   return CS0[]
16 end function
```

value from virtual vector s and m registers from all memory. $C_S[i]$ is the number of register has value i in s register. Let $C_{S0}[]$ is the idea counter array of s register which is set only by current spreader. And the primary function of RegisterNormalize is recovering $C_{S0}[]$ from $C_S[]$. The analysis for finding the original register value is based on probability theory. When one of the original register has value is i, the probability this register is overwrite by higher value j from other spreader is $C_M[j]/m$, so if there is $C_S[i]$ register has value is i then the number of registry change from i to j is $C_{S0}[i].C_M[j]/m$. Therefor, the counter value at a certain index i, $C_S[i]$ equals $C_{S0}[i]$ plus a number of counter change from $h<i$ to i, minus a number of register change from i to $j>i$ like below equation

$$i] = C_S[i] + \sum_{h=0}^{i-1} C_{S0}[h]\frac{C_M[i]}{m} - \sum_{j=i+1}^{c} C_{S0}[i]\frac{C_M[j]}{m}$$
$$= C_{S0}[i] + C_M[i]\sum_{h=0}^{i-1}\frac{C_{S0}[h]}{m} - C_{S0}[i]\sum_{j=i+1}^{c}\frac{C_M[j]}{m}$$

Then, we have

$$C_{S0}[i] = \frac{C_S[i] - C_M[i]\sum_{h=0}^{i-1}\frac{C_{S0}[h]}{m}}{1-\sum_{j=i+1}^{c}\frac{C_M[j]}{m}} = \frac{m \cdot C_S[i] - C_M[i]\sum_{h=0}^{i-1} C_{S0}[h]}{m - \sum_{j=i+1}^{c} C_M[j]}$$

The final procedure of function RegisterNormalize list in Alg. 3

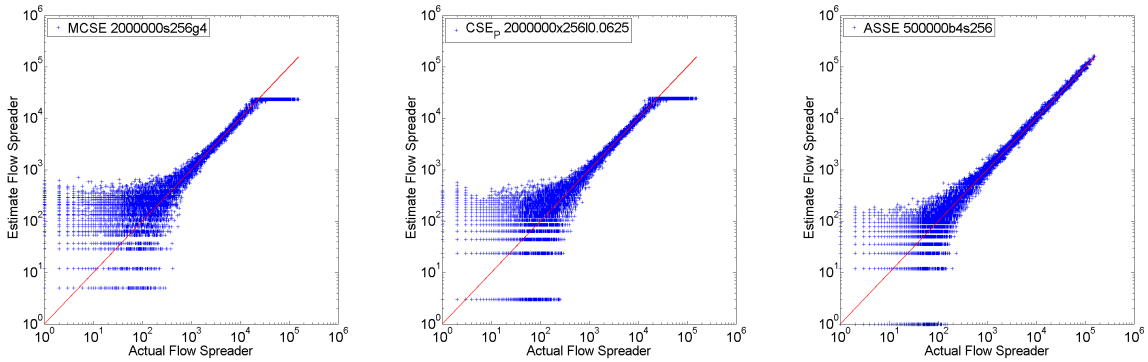
III. Experiment

In our experiment, to estimate the high spreader, we used 1 hour traffic when the network is attacked by Witty worm from CAIDA [5]. The number of spreaders is 12,379, the average and maximum size of the spreader is 3,287 and 156.432 respectively. We evaluate the performance of 3 algorithms: MCSE [3], CSE with sampling (CSES) [4] and our ASSE in terms of processing time and estimation accuracy. The same amount of memory and comparative parameters are used for fair comparison. These values are listed in Table 1.

Table 1: The parameters of 3 schemes uesd for experiments

Memory	1 MB
MCSE	m=2M; s=256; g=4;
CSES	m=2M; s=256; p=0.0625
ASSE	m=0.5M; s=256; r=4bit

We draw the figure based on the real spreader size in x-axis and the estimated spreader size. in y-axis. Both are in logarithm scale. Each point is drawn by one spreader, and there is the reference line y = x in these figure. The more closer a point of a spreader to the reference line, the more precise estimation is. If a point is under the line, its estimation is under-estimated. Otherwise, it is over-estimated. As can be seen in Fig. 1 the MCSE and CSES show the upper bound limitation in estimation which is about 20,000 while the ASSE deliver accurate estimation in all range up to 150,000.



(a) MCSE (b) CSES (c)ASSE

Figure 1: Estimation result with MCSE (m=2M; s=256; g=4). CSES (m=2M; s=256; p=0.0625) and ASSE (m=0.5M; s=256; r=4bit) with 0.25MB memory. Each point (in cross) stands for each spreader, and y = x line is the reference line. The closer the point is to y = x, the more accurate the estimation is. Note that both axes are in log scale

Moreover, ASSE shows better accuracy in all range.

In order to look into the accuracy more clearly, we divide spreaders into 3 groups: small (1-63), normal (64-1023) and large (1024-100,000) size. Afterwards, we do the descriptive statistics of the error */real - estimated/* over the real spreader to show the accuracy of the estimation. The results are listed in Table 2. It has been found that ASSE delivers from 1.5 to 2 times better accuracy than MCSE and CSES.

Table 2: The average ratio of the estimated error and real spreader in all ranges.

Range	MCSE	CSES	ASSE
Small (1-63)	13.886	12.416	5.951
Normal (64-1023)	0.4725	0.4318	0.2704
Large (1024-100,000)	0.1554	0.1462	0.0958

ASSE also has its limitation. With normal traffic which has more spreader and the spreaders size are relatively small or the memory is so small, CSE and CSES come with better since they have more number of bits than number of registers in ASSE.

IV. Conclusion

In summarize, our proposed scheme ASSE provide better accuracy with nearly unlimited bound in estimating high spreader when other algorithms fail. With simple encoding and decoding, it can be easily implemented but still needed to be statically analyzed to fully used for network applications.

References

[1] K. Hwang, B. Vander-Zanden, and H. Taylor, “A linear-time probabilistic counting algorithm for database applications,” Trans. Database Syst., vol. 15, no. 2, pp. 208 - 229, Jun. 1990

[2] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier. “HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm”. DMTCS Proceedings, 2008.

[3] M. Yoon, T. Li, S. Chen, and J. Peir. “Fit a spread estimator in small memory”. IEEE INFOCOM 2009.

[4] T. Li, S. Chen, and W. Luo. “Spreader classification based on optimal dynamic bit sharing”. Networking, IEEE/ACM Transactions on, pp. 817-830, 2013.

[5] C. cooperation http://www.caida.org, “The cooperative association for internet data analysis.”