Fast and Accurate Traffic Matrix Measurement Using Adaptive Cardinality Counting

Min Cai[†] Jianping Pan[‡] Yu-Kwong Kwok[†] Kai Hwang[†]

[†]University of Southern California [‡]NTT MCL
{mincai, yukwong, kaihwang}@usc.edu panjianping@acm.org

Categories and Subject Descriptors:
C.2.3 [Computer Communication Networks]: Network Operations
General Terms: Measurement, Algorithms, Security
Keywords: Traffic Matrix, Cardinality Counting

1. INTRODUCTION

The spatial and temporal properties of traffic matrix (TM) can be used to diagnose various network anomalies [6]. To identify anomalies quickly, TM needs to be measured in a fast and accurate manner. Besides byte-level TM (BTM) and packet-level TM (PTM), flow-level TM (FTM) is also important for anomaly detection since deliberate anomalies often start with an unusual increase in small flows, such as flooding attacks and scanning worms [2, 5].

Existing TM measurement techniques are often not sufficient for anomaly detection since most of them operate on a hourly basis and have limited accuracy. Additionally, most TM schemes only offer BTM or PTM while FTM is often missing. Flow statistics are not available in ordinary SNMP MIB for inference. Sampled flow statistics may be available through NetFlow, but small flows are often under-counted since they are less likely to be sampled [2].

We propose a *cardinality-based TM measurement* (CBTM) approach with an *adaptive counting* algorithm. The *cardinality* of a set of packets or flows is the number of *distinct* elements in the set. Our CBTM scheme uses a small memory, *cardinality summary*, to digest packets or flows observed at each OD router. Both PTM and FTM are measured by estimating overlapping packets or flows at OD routers. In addition, the number of packets or flows might change dramatically during flooding attacks or worm outbreaks. The adaptive counting algorithm can adapt itself to varying cardinalities without imposing any extra processing or memory costs.

The CBTM scheme estimates both PTM and FTM in almost real-time (once every 10 seconds) with low average relative errors (less than 5%). It has low processing, storage and communication costs. For example, each router only needs 640 KB memory to achieve 5% average relative error for 40 Gbps (OC-768) line speed. Further, our approach can also be implemented in a passive mode and deployed incrementally without changing ISP's existing infrastructures.

2. ADAPTIVE CARDINALITY COUNTING

Several probabilistic algorithms [8, 4, 3] have been proposed to estimate large set cardinalities using a small memory. The LogLog algorithm by Durand and Flajolet counts large cardinalities very efficiently in terms of space complexity and accuracy [3]. It first uses a uniform hash function to eliminate duplicates and generate uniformly distributed hash values. The hash values are separated into m groups, or *buckets*. For a hash value x in binary format, let $\rho(x)$ denote the position of its first most significant bit 1. It uses m counters in memory denoted by $M^{(j)}$ to record the largest ρ of hash values in bucket j. The arithmetic mean $\frac{1}{m}\sum_{j=1}^{m} M^{(j)}$ can be expected to approximate $\log_2(n/m)$ with an additive bias.

However, LogLog counting is asymptotically unbiased *only* when n is much greater than m. Since many buckets are empty when load factor $t = n/m$ is relatively small, sampling error will introduce significant estimation bias. For a set of n distinct elements, the probability that a bucket is empty is $p_e = (1 - 1/m)^n \approx (1/e)^{n/m} = e^{-t}$, i.e. the number of empty buckets will increase exponentially as t decreases. Although LogLog counting is not suitable when t is small, the expected number of empty buckets can give us a much better estimation on n. Suppose the number of empty buckets is b_e, the expected value of b_e after digesting n distinct elements is $\mathrm{E}[b_e] = mp_e \approx e^{-n/m}$, and n can be estimated as $\hat{n} = -m\ln(b_e/m)$. This is similar to the *linear counting* algorithm that counts b_e using a bitmap of m bits instead of m buckets [8]. Our observation is that there is a constant *switching load factor*, denoted by t_s, at which the standard error of linear counting becomes greater than that of LogLog counting. We calculate t_s analytically and we have $t_s \approx 2.89$, which is independent of the number of buckets m.

The *adaptive counting* algorithm uses the same data structure as LogLog counting, i.e. an array of m counters, which is referred to as *cardinality summary* in our context. Suppose the ratio of empty buckets is β, we have $\beta_s = b_e/m = e^{-t_s} = 0.051$, where β_s is the ratio of empty buckets when $t = t_s$. Thus, n can be estimated as follows

$$\hat{n} = \begin{cases} \alpha_m m 2^{\frac{1}{m}\sum_{j=1}^{m} M^{(j)}} & \text{if} \quad 0 \le \beta < 0.051 \\ -m\ln(\beta) & \text{if} \quad 0.051 \le \beta \le 1 \end{cases}. \qquad (1)$$

3. TRAFFIC MATRIX ESTIMATION

We now present a *cardinality-based TM measurement* (CBTM) approach using the adaptive counting algorithm. Consider a network with n routers $\{R_1, R_2, ...R_n\}$. We denote $X(t_k)$ as the TM for the k-th time slot t_k, and $X_{i,j}(t_k)$ represents the amount of traffic from ingress router R_i to egress router R_j during t_k. The traffic is counted in packets for PTM (denoted as $X^p(t_k)$), or in flows for FTM ($X^f(t_k)$). Let $\mathbf{S}_i^+(t_k)$ be the set of traffic entering the network from R_i during t_k, and $\mathbf{S}_j^-(t_k)$ be the set of traffic leaving from R_j during t_k. Obviously, the same packet or flow traveling from R_i to R_j should appear in both $\mathbf{S}_i^+(t_k)$ and $\mathbf{S}_j^-(t_k)$. According to the definition of TM, we have $X_{i,j}(t_k) = |\mathbf{S}_i^+(t_k) \cap \mathbf{S}_j^-(t_k)|$. One of our insight is that the intersection operation can be transformed into a union

SIGCOMM'05 Workshops, August 22–26, 2005, Philadelphia, PA, USA.
ACM 1-59593-026-4/05/0008.

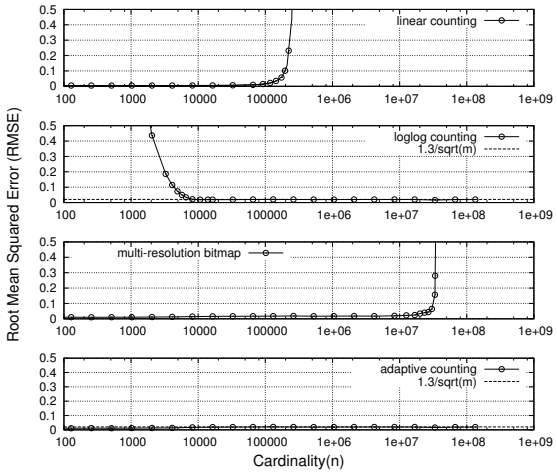

Figure 1: Comparison of four counting algorithms with n scaling from $0.1K$ to 16M, m=4096.

operation, i.e. $X_{i,j}(t_k) = |\mathbf{S}_i^+(t_k)| + |\mathbf{S}_j^-(t_k)| - |\mathbf{S}_i^+(t_k) \cup \mathbf{S}_j^-(t_k)|$.

We digest $\mathbf{S}_i^+(t_k)$ and $\mathbf{S}_j^-(t_k)$ into small cardinality summaries, denoted as $\mathbb{S}_i^+(t_k)$ and $\mathbb{S}_j^-(t_k)$, at R_i and R_j. $|\mathbf{S}_i^+(t_k)|$ and $|\mathbf{S}_j^-(t_k)|$ can be estimated using adaptive counting on $\mathbb{S}_i^+(t_k)$ and $\mathbb{S}_j^-(t_k)$. In addition, $|\mathbf{S}_i^+(t_k) \cup \mathbf{S}_j^-(t_k)|$ can be estimated by *max-merging* $\mathbb{S}_i^+(t_k)$ and $\mathbb{S}_j^-(t_k)$ (instead of $\mathbf{S}_i^+(t_k)$ and $\mathbf{S}_j^-(t_k)$!). Recall that the cardinality summary consists of an array of counters $\mathbb{S}[j] = M^{(j)}$, where $1 \le j \le m$. The *max-merge* operation, or $\mathbb{U}$, is defined as follows: $\mathbb{S} = \mathbb{S}_1 \mathbb{U} \mathbb{S}_2$ iff $\forall j \in [1, m], \mathbb{S}[j] = \max\{\mathbb{S}_1[j], \mathbb{S}_2[j]\}$.

The CBTM approach can be applied to both PTM and FTM. For PTM, we digest the packets at ingress or egress routers into a cardinality summary using packet identities that consist of invariant IP header fields as well as a few bytes of payload [7]. While for FTM, we use 5-tuple flow identities as the elements of flow sets. Here, we assume flows are terminated by inter-packet timeout T_f, and its default value is typically 30 seconds. When the duration of t_k is less than T_f, the number of flows is equal to the distinct flow identities seen in packets, i.e. the cardinality of flow identities of all packets.

4. PERFORMANCE EVALUATION

We evaluated the efficacy of our CBTM-based approach with adaptive counting for both PTM and FTM. Figure 1 compares adaptive counting algorithm with linear counting [8], LogLog counting [3] and multi-resolution bitmap [4] by scaling cardinalities in about six orders of magnitude. The accuracy is measured by *Root Mean Squared Error* (RMSE) of the ratio $\hat{n}/n$. All four algorithms use the same 20 Kbit memory and the multi-resolution bitmap is configured with 2% relative error for up to 10M. Figure 1 shows that linear counting can estimate small cardinalities (less than 100K) more accurately than other three algorithms, but cannot estimate cardinalities larger than 200K. In contrast, LogLog counting performs very well as n is greater than 10K. However its RMSE increases dramatically when n scales from 10K down to 0.1K. The multi-resolution bitmap and adaptive counting have almost the same RMSE when n is less than 10M. But when n is greater than 10M, the multi-resolution bitmap becomes full very quickly and cannot estimate n correctly. Adaptive counting is capable of scaling both up to very large cardinalities (more than 130M) and down to small ones (less than 0.1K).

We also evaluated the accuracy of our CBTM scheme using real

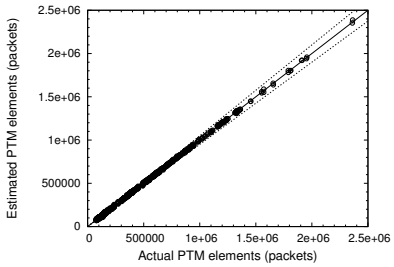

Figure 2: Actual vs. estimated PTM elements, m=1024K.

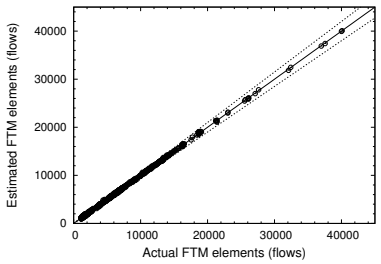

Figure 3: Actual vs. estimated FTM elements, m=1024K.

OC192 NLANR packet traces [1]. We simulated an ISP network with 16 OD routers and measured both PTM and FTM once every 10 seconds using 1024K buckets. Figure 2 illustrates the estimated vs. actual PTM elements in 5 runs. The solid diagonal line shows equality and the doted line shows $\pm 5\%$. As most points are clustered around the diagonal, our CBTM scheme achieves a very accurate estimation of PTM. The average relative error is 2.4% while the largest relative error is 19% for a small PTM element of actual $137,584$ packets. When PTM elements become larger, the estimation tends to be much more accurate. Similarly, Figure 2 shows the estimated vs. actual FTM elements. Our scheme also estimates FTM very accurately although FTM elements are much smaller than those of PTM. The average relative error is 2.78% for all elements while the largest relative error is 27% for an small FTM element of actual $1,450$ flows.

In summary, the CBTM scheme can estimate both PTM and FTM in almost real-time (once every 10 seconds) with low average relative errors (less than 5%). Our performance study also shows that each packet only needs about 1040 CPU cycles on average to process in software, and a commodity PC with 3 GHz CPU can process more than 2.8 million packets per second.

5. REFERENCES

[1] NLANR Moat PMA trace archive. See http://pma.nlanr.net/Traces.

[2] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *SIGCOMM '03*, pages 325–336, 2003.

[3] M. Durand and P. Flajolet. Loglog counting of large cardinalities. In *11th Annual European Symposium on Algorithms*, Sept. 2003.

[4] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. In *IMC'03*, Oct. 2003.

[5] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *SIGMETRICS '04*, pages 177–188, 2004.

[6] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM*, pages 219–230, 2004.

[7] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer. Single-packet ip traceback. *IEEE/ACM Transaction on Networking*, 10(6):721–734, 2002.

[8] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Trans. Database Syst.*, 15(2):208–229, 1990.