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,kaihwang*}*@usc.edu* *panjianping@acm.org*

Abstract

Traffic matrix (TM) can be used to detect, identify, and trace network anomaly caused by DDoS attacks and worm outbreaks. To detect network anomaly as early as possible, we need to obtain TM in a fast and accurate manner. Many existing TM estimation techniques are found not sufficient for this purpose due to their high overhead or low accuracy. We propose a cardinality-based TM measurement approach with an adaptive counting algorithm to produce both packet-level and flow-level TM, which is well-suited for TM-based anomaly detection on a network basis. Our results show that the approach can obtain TM in almost real-time (once very 10 seconds) with low average relative error (less than 5%). Our approach has low processing, storage and communication overhead, e.g. software implementation can support OC-192 line speed. It can also be implemented in a passive mode and deployed incrementally without changing current routing infrastructure.

1 Introduction

A network traffic matrix (TM) represents the amount of traffic in bytes, packets, or flows between origin and destination (OD) pairs of a network in a time slot [10]. TM is an important building block for many operation tasks in large-scale networks: its spatial property can guide network planning and traffic engineering, and its temporal behavior is a good indicator of traffic growth and network anomaly, where the latter is often caused by hardware failure, software bugs, or security attacks. With regard to OD pairs and time slots, a TM can offer various levels of insight into a real network.

We use TM to detect, identify, and trace deliberate network anomaly caused by DDoS flooding attacks and scanning worms, and we want to do so at their earliest stage and before they become epidemic. Compared to other anomaly detection techniques focusing on individual hosts, links, and routers, our approach is well-suited for backbone ISPs that peer with other ISPs, serve many customers, and want to protect their assets on a network basis. To identify network anomaly as early as possible, we need to obtain TM in a fast and accurate manner. Besides byte-level TM (BTM) and packet-level TM (PTM), we are more interested in flow-level TM (FTM), since deliberate anomaly often starts with an increase in small flows, e.g., flooding attacks with spoofed source addresses and scanning worms with random destination addresses.

TM, as a network-wide metric, is intrinsically difficult to measure and estimate in large-scale networks. TM techniques usually collect local information at OD routers and other locations first, and then manipulate the information globally to produce a TM for the entire network. Depending on how to balance overhead, existing TM techniques belong to two major categories: statistical inference and direct measurement. For the former [15, 6, 5, 11, 18, 19, 14], TM is inferred through SNMP link counts, routing matrix, and intelligent priors, i.e., simplified information collection followed by sophisticated manipulation. Direct measurement on the other hand collects much more fine-grained information to simplify its manipulation. To keep the process affordable, Papagiannaki et al. [13] measure TM from sampled flow statistics to reduce communication, storage, and processing overhead.

However, existing TM techniques are still not sufficient for our purpose. First, they often operate on a hourly basis to balance accuracy and overhead, which is not fast enough for anomaly detection preferably in every 10 seconds. Second, existing techniques only achieve limited accuracy with relative error much higher than 10%. Slow and inaccurate TM can lead to higher than expected false positives and negatives, especially when we want to early detect network anomaly. There is one exception: Soule et al. [14] are able to achieve 4 to 5% relative error by deliberately changing network routing to relax TM constraints. But in reality, it is hard to update routing as fast as generating TM, and ISPs are unlikely to alter their routing regularly just for TM purpose.

Further, due to information availability and overhead constraint, existing TM techniques often can only offer BTM

and PTM; FTM, which is more important for anomaly detection, is largely missing. For example, flow statistics are not readily available in ordinary SNMP MIB for statistical inference. Indeed, counting active flows in high-speed links itself still imposes great challenges [8]. Sampled flow statistics may be available on some occasions through NetFlow, but small flows are often greatly under-counted, since they are much less likely to be sampled. Therefore, we need develop new information collection and manipulation techniques to support TM-based early network anomaly detection.

In this paper, we propose a *cardinality-based TM measurement* (CBTM) approach with an *adaptive counting* algorithm. Our approach can estimate PTM and FTM in almost real-time (once every 10 seconds) with very low overhead and relatively low error (less then 10%). The *cardinality* of a set, consisting of packets or flows in our context, is the number of its *distinct* elements, while its *size* counts *all* elements. Existing probabilistic counting algorithms are typically designed on purpose for a predetermined range of set size and cardinality. However, the number of packets or flows can change dramatically in the order of magnitude during DDoS flooding attacks or worm outbreaks, and most existing algorithms cannot scale either up to large cardinalities [16], or down to small cardinalities [7]. Our adaptive counting algorithm, on the other hand, can intelligently adapt itself to the cardinality of the set it counts with different algorithm variants and a common data structure, which is well-suited for anomaly detection.

Our CBTM scheme uses a very small summary to digest the set of packets or flows observed at an OD router, and estimates PTM or FTM by merging these summaries from different OD routers and by taking advantage of our adaptive counting algorithm. Here, the light-weight summary serves for a dual purpose: it estimates the cardinality of the digested set; it also enables meaningful set operations to be conducted over summaries from different routers or in different time slots. Our approach strikes a better balance between information collection and manipulation; i.e., cardinality summary contains much more information than counters, but it only imposes a little bit additional overhead than the latter.

Our approach has very low processing, storage, and communications overhead. Each packet only needs a few hundred cycles to process in software implementation, and a commodity PC with 3 GHz CPU can process more than 3 million packets per second (or 12 Gbps with average packet size of 500 bytes). With hardware acceleration and 10 ns SRAM, it can process 50 million packets per second. To achieve 10% relative error for line speed up to 40 Gbps (OC-768), we only need 640 KB memory in each router, and the communication cost to send cardinality summaries to network operation center is about 512 Kbps. Also, our approach does not need any routing information and is independent of routing changes. PTM and FTM can be partially estimated for a subset of OD pairs by only collecting their packet and flow information. Further, our approach can be implemented in passive measurement mode and be incrementally deployed without forcing ISPs to replace their current routing infrastructures.

The remainder of the paper is organized as follows. In Sec. II, we outline the system model and main building blocks. We present our adaptive counting algorithm in Sec. III and our CBTM approach in Sec. IV. Section V gives experimental results with real Internet traffic, and Sec. VI offers further discussion, followed by conclusions in Sec. VII.

2 System Model

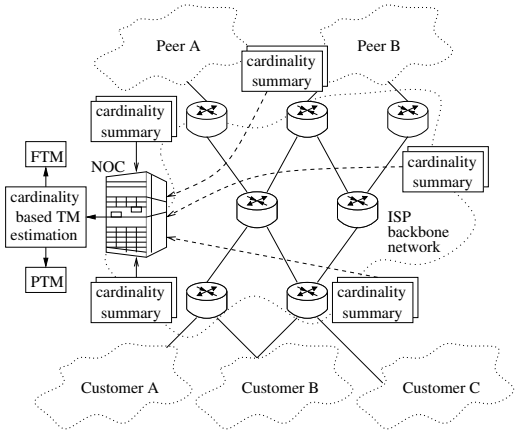

Figure 1: The system model of cardinality based TM estimation

Figure 1 shows the system model of our approach. Assume an ISP backbone network consists of many OD routers to other ISPs or its customers and a large number of internal routers. At each OD router, we can observe traffic incoming to and outgoing from the network by collocating passive measurement facilities with it or its adjacent links. With specialized network monitoring interfaces in promiscuous mode such Endace DAG cards, these facilities can capture all packets of concerned traffic appear on the replicate port of OD routers or shared medium in adjacent links. Due to customer multi-homing and ISP multi-peering, packets with the same source and destination addresses may have different entry/exit points to/from the network and can travel through different OD pairs.

With ever-increasing link speed, it becomes much less likely that we can keep all captured packets for a long time. Instead of keeping these packets, we propose to summarize their identity into a small digest. Snoeren et al show that the first 28 bytes of a packet, excluding 4 bytes in its 20-byte IP header and including 12 bytes of its payload, are sufficient for identification purpose with collision probability

less then 0.1%% for wide-area traffic. We adopt this notation as packet identity. Packet identities are randomized by a common hash function, and the same packet should have the same hash value throughout the network. OD flows, on the other hand, are identified by their 5-tuple consisting of source and destination IP addresses and port numbers, as well as protocol identifiers.

Measurement facilities periodically send the digests they produce to the network operation center, where these digests are further processed to estimate the cardinality of summarized set of packets or flows, and to further estimate the PTM or FTM for the entire network. Due to the persistence of packet or flow identity, the same packet or flow identity should appear in the digests produced by its both ingress and egress routers. We will show that even though we cannot recover these summarized packets or flows, the operation center still can estimate the number of packets or flows travel through a particular OD router, or an OD pair, which is PTM and FTM.

PTM and FTM have many applications. We are particularly interested in using them to detect, identify, and trace network anomalies such as those caused by DDoS flooding attacks and scanning worms. These anomalies usually will create many small flows (i.e., flows with a very few packets) initially. Since we summarize all packets into digests without sampling, these small flows will be all captured, and the increase of small flows will be reflected by the increase of cardinality of digests as well. The spatial and temporal behaviors of PTM and FTM can indicate the behavior and progress of attacks. For example, if a customer network suddenly has many small flows to all other customer networks and peering ISPs, it is very likely some hosts in this customer network have been compromised by scanning worms. On the other hand, if all OD routers suddenly have many small flows to a particular customer network, it is very likely that some hosts in this customer network are undergoing DDoS flooding attacks.

3 Cardinality Estimation

Counting large cardinalities is a very important problem in many areas such as database query optimization and network traffic analysis. Several probabilistic algorithms have been proposed since middle 1980s, e.g. probabilistic counting [9], linear counting [16], multi-resolution bitmap [8] and LogLog counting [7]. These algorithms can estimate the cardinality of a considerably large set by digesting the set into a small memory referred to as *cardinality summary*.

3.1 LogLog Counting

The LogLog algorithm designed by Durand and Flajolet counts sets of large cardinality very efficiently in terms of space complexity and accuracy [7]. Similar to other approaches, the LogLog algorithm first applies a uniform hash function to all elements in set $\mathbf{S}$ to eliminate duplicates and to generate hash values that closely resemble a random uniform distribution. For a hash value x in binary string format, i.e. $x \in \{0,1\}^\infty$, let $\rho(x)$ denote the position of its first most significant bit 1, e.g. $\rho(1\cdots) = 1$ and $\rho(001\cdots) = 3$. Durand and Flajolet showed the largest ρ for a set of randomly distributed hash values can provide a reasonable indication on $\log_2 n$, where n is the set cardinality. To obtain a better estimate, the LogLog algorithm separates hash values into m groups, or *buckets*, by using the least significant k bits of x as bucket index, where $m = 2^k$. Suppose counter $M^{(j)}$ records the largest ρ for hash values in bucket j, then 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. Therefore, the cardinality n can be estimated by

$$\hat{n} = \alpha_m m 2^{\frac{1}{m}\sum_{j=1}^{m} M^{(j)}}, \tag{1}$$

where α_m is a correction factor approximated by $\alpha_\infty = e^{-\gamma\sqrt{2}/2} \doteq 0.39701$ (γ is Euler's constant) when $m \geq 64$.

The estimator $\hat{n}$ is asymptotically unbiased as $n \to \infty$. The bias (B) of $\hat{n}/n$ is derived from its expectation (E), or

$$\mathrm{B}(\hat{n}/n) = \mathrm{E}(\hat{n}/n) - 1 = \theta_{1,n} + o(1), \tag{2}$$

where $|\theta_{1,n}| < 10^{-6}$ as $n \to \infty$ [7]. The standard error (SE) of $\hat{n}/n$ is derived from the variance (Var), or

$$\mathrm{SE}(\hat{n}/n) = \sqrt{\mathrm{Var}(\hat{n})}/n \approx 1.30/\sqrt{m}. \tag{3}$$

3.2 Adaptive Counting

The LogLog algorithm has very low space-complexity and it only requires m counters in memory with $\log_2(\log_2(n/m)+3)$ bits for each counter. For example, with 1 million 5-bit counters (640 KB), it can estimate cardinality up to 10^{14} with standard error around 0.13%.

Although the LogLog algorithm can scale up to very large cardinalities, it is asymptotically unbiased *only* when the cardinality n is much greater than the number of buckets m. Figure 2 plotted the empirically measured bias, in log scale, of the ratio $\hat{n}/n$ by varying the load factor $t = n/m$. The number of buckets m is configured as 64K, 256K and 1024K, respectively. The experiment shows that the bias increases exponentially when the load factor t is less than 3. From (3), we know m should be large enough to ensure accuracy: e.g., we need at least 1 million buckets for an estimator of 0.13% error, and cardinalities less than 3 millions cannot be accurately estimated due to large bias.

The large bias of the LogLog algorithm for comparatively small cardinalities is mainly due to sampling error, since many buckets are empty when t is relatively small. The probability that an element is hashed into a given bucket is

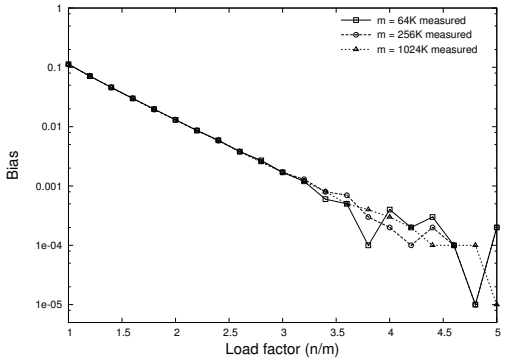

Figure 2: The bias of LogLog algorithm for small cardinalities with load factor from 1 to 5. Note that the bias is in log scale.

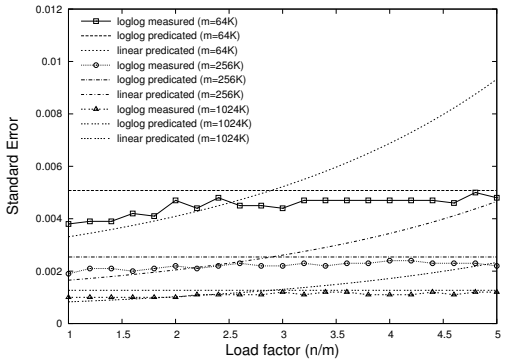

Figure 3: The standard error of LogLog algorithm for small cardinalities with load factor less than 5.

$p = 1/m$. For a set of n distinct elements, the probability that a bucket is empty is $p_e = (1-p)^n \approx (1/e)^{n/m} = e^{-t}$; i.e., the number of empty buckets will increase exponentially when t decreases. For example, when $t = 1$, there are 36% empty buckets.

Although the LogLog algorithm is not suitable for estimating cardinalities when load factor 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 number of empty buckets after digesting n distinct elements is $\mathrm{E}[b_e] = mp_e \approx e^{-n/m}$, and n can be estimated as

$$\hat{n} = -m \ln(b_e/m). \quad (4)$$

Whang et al. also show that this is the maximum likelihood estimator of n [16]. In their *linear counting* algorithm, they count b_e using a bitmap of m bits instead of m buckets. The bias and standard error of $\hat{n}/n$ of their scheme are

$$\mathrm{B}(\hat{n}/n) = (e^t - t - 1)/(2n) \quad (5)$$

$$\mathrm{SE}(\hat{n}/n) = \sqrt{e^t - t - 1}/(t\sqrt{m}). \quad (6)$$

In order to achieve a specified accuracy, load factor t in linear counting cannot be too large. Figure 3 shows the predicted standard errors of linear counting and LogLog counting as well as the measured standard error of LogLog counting as t varying from 1 to 5. The standard error of LogLog counting is almost constant as predicated by (3), while that of linear counting increases significantly when t changes from 1 to 5. However, there is a certain load factor for different numbers of buckets at which the standard error of linear counting becomes greater than that of LogLog counting. We refer to this particular load factor as *switching load factor*, denoted by t_s. From (3) and (6), we have

$$\mathrm{SE}(\hat{n}/n) = \sqrt{e^{t_s} - t_s - 1}/(t_s\sqrt{m}) = 1.30/\sqrt{m}. \quad (7)$$

Solving (7), we have $t_s \approx 2.89$, which is independent of the number of buckets m. As shown in Fig. 2, the maximum bias of LogLog counting is below 0.17% when $t > t_s$, which is sufficient for most cardinality estimation applications.

Therefore, to better estimate both small and large cardinalities, we propose an *adaptive counting* algorithm that estimates small cardinalities using linear counting when $t \leq t_s$ and estimates large ones using LogLog counting when $t > t_s$. Our adaptive counting algorithm uses the same data structure as LogLog counting, i.e. an array of m counters $M^{(j)}$ to record the largest ρ for hash values in each bucket. Since $\rho(x)$ is always greater than zero for all x ($\rho(x) = L+1$ for $x = 0$, where L is the bit length of x), a bucket j is empty *iff* $M^{(j)} = 0$. Suppose the ratio of empty buckets is β, we have $\beta_s = b_e/m = e^{-t_s} = 0.051$ from (4), where β_s is the ratio of empty buckets when $t = t_s$. Therefore, the adaptive counting algorithm estimates n as follows

$$\hat{n} = \begin{cases} \alpha_m m 2^{\frac{1}{m}\sum_{j=1}^{m} M^{(j)}} & \text{if} \quad 0 \leq \beta < 0.051 \\ -m \ln(\beta) & \text{if} \quad 0.051 \leq \beta \leq 1 \end{cases}. \quad (8)$$

Our adaptive counting algorithm uses only one data structure for two estimation algorithms, i.e. supporting both linear counting and LogLog counting without introducing any extra processing and storage overhead, when compared with the ordinary LogLog algorithm. Also, as supported by our results, the algorithm can scale down to small cardinalities and scale up to large ones simultaneously, which cannot be accomplished by either LogLog counting or linear counting. The dual-scalability of our adaptive counting algorithm makes it well-suited for measuring highly dynamic PTM and FTM to detect, identify, and trace network anomaly.

4 Traffic Matrix Estimation

We now present a *cardinality-based TM measurement* (CBTM) approach using our adaptive counting scheme. Consider a network represented by a set of n nodes $\{R_1, R_2, ...R_n\}$. The nodes can be links, routers, or Points-of-Presence (PoPs). In this paper, we focus on measuring

router-to-router traffic matrix, and our scheme can be applied at other granularities as well. 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 the time slot t_k, and $\mathbf{S}_j^-(t_k)$ be the set of traffic leaving from R_j during t_k.

As we discussed in Sec. **??**, the same packet or flow traveling from R_i to R_j should appear in both $\mathbf{S}_i^+(t_k)$ and $\mathbf{S}_j^-(t_k)$, if every router is perfectly time-synchronized and there is no network latency. We will consider the case with time skew and network latency later in this section. If there are no duplicate packets (we will discuss the effect of duplicate packets in Sec. 6), according to the definition of TM, we have

$$X_{i,j}(t_k) = |\mathbf{S}_i^+(t_k) \cap \mathbf{S}_j^-(t_k)|. \quad (9)$$

Obviously, it is impractical to collect all original sets of traffic from routers and perform intersection operations to obtain the TM. One of our insight is that the intersection operation can be transformed into a union operation, i.e.

$$X_{i,j}(t_k) = |\mathbf{S}_i^+(t_k)| + |\mathbf{S}_j^-(t_k)| - |\mathbf{S}_i^+(t_k) \cup \mathbf{S}_j^-(t_k)|. \quad (10)$$

To do so, 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 then be estimated by using our adaptive counting algorithm on $\mathbb{S}_i^+(t_k)$ and $\mathbb{S}_j^-(t_k)$. To estimate $|\mathbf{S}_i^+(t_k) \cup \mathbf{S}_j^-(t_k)|$, we only need to merge $\mathbb{S}_i^+(t_k)$ and $\mathbb{S}_j^-(t_k)$ (instead of $\mathbf{S}_i^+(t_k)$ and $\mathbf{S}_j^-(t_k)$!), and estimate the merged cardinality. Recall that the cardinality summary consists of an array of counters $\mathbb{S}[j] = M^{(j)}$, where $1 \le j \le m$. We can define the *merge* operation, or $\Cup$, as follows

Definition 1 $\mathbb{S} = \mathbb{S}_1 \Cup \mathbb{S}_2$ *iff* $\forall j \in [1,m], \mathbb{S}[j] = \max\{\mathbb{S}_1[j], \mathbb{S}_2[j]\}$

As we discussed in Sec. III, the duplicate elements of a set are eliminated after applying the uniform hash function. Also, the cardinality summary of a set is independent of the processing sequence of its elements. Suppose $\mathbf{S}_{i,j}(t_k) = \mathbf{S}_i^+(t_k) \cup \mathbf{S}_j^-(t_k)$, we have $\mathbb{S}_{i,j}(t_k) = \mathbb{S}_i^+(t_k) \Cup \mathbb{S}_j^-(t_k)$. Therefore, $|\mathbf{S}_i^+(t_k) \cup \mathbf{S}_j^-(t_k)|$ can be estimated from the merged cardinality summary $\mathbb{S}_i^+(t_k) \Cup \mathbb{S}_j^-(t_k)$.

PTM and FTM — 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 by using packet identities defined in Sec. **??** as set elements. While for FTM, we use 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, each flow identity represents a unique flow in time slot t_k, and all packets that share the same flow identity belong to a single flow. Hence, the number of flows is equal to the distinct flow identities seen in the packets, i.e. the cardinality of flow identities of all packets.

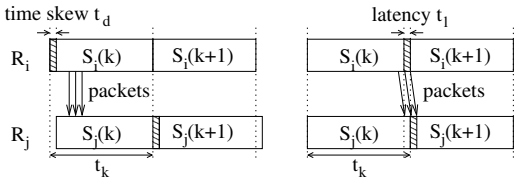

Figure 4: The effect of time skew and network latency.

Time Slot Alignment — In the discussion above, we assume that the clock of each router is perfectly synchronized and there is no network latency. Actually, routers only need to be coarsely time-synchronized. For fast TM estimation, we typically set the duration of a time slot be 10 seconds. Suppose routers are synchronized using NTP, of which time skews t_d are often within 10 ms in the wide area [12], there will be approximately 0.1% packets in $\mathbf{S}_i^+(t_k)$ missing in $\mathbf{S}_j^-(t_k)$ as shown in Fig. 4(a). The error caused by time skew and clock drift is insignificant when compared with the TM estimation error that is often around 5% as shown in Sec. 5. The error can also be reduced by using the time slot realignment technique discussed below.

Further, suppose the average latency of packets traveling from R_i to R_j is t_l, there will be on average $t_l/(t_{k+1} - t_k)$ fraction of packets in $\mathbf{S}_i^+(t_k)$ digested into $\mathbb{S}_j^-(t_{k+1})$ as shown in Fig. 4. We can solve this problem by using so-called *slot realignment*; i.e., we intersect $\mathbf{S}_i^+(t_k)$ with $\mathbf{S}_j^-(t_k) \cup \mathbf{S}_j^-(t_{k+1})$, instead of just $\mathbf{S}_j^-(t_k)$. Even without latency, this technique is accurate since a packet should not appear in both summaries for t_k and t_{k+1} with high possibility. The same is true for flow summary if its duration is less than $T_f/2$.

5 Performance Evaluation

We evaluated the efficacy of our CBTM-based approach with adaptive counting for both PTM and FTM. We compared our adaptive counting algorithm with linear counting and LogLog counting by scaling cardinality in four orders of magnitude. We also evaluated the accuracy of our CBTM scheme to measure PTM and FTM by using real OC192 NLANR packet traces [2], and benchmarked the performance of packet/flow digesting in CBTM.

5.1 Scalability of Adaptive Counting

The efficacy of our CBTM scheme largely depends on the space complexity and accuracy of the adaptive counting algorithm. In this experiment, we compared adaptive counting with linear counting and LogLog counting on sets of varying

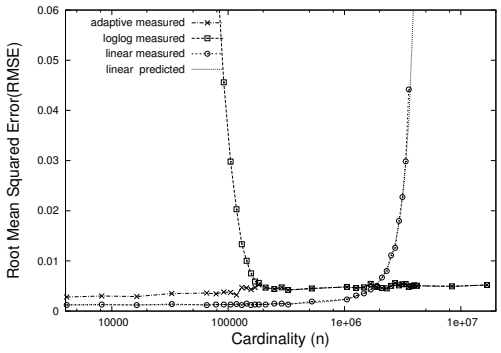

Figure 5: Comparison of three counting algorithms with n scaling from 4K to 16M, m=64K.

cardinalities. The estimation accuracy is measured by *Root Mean Squared Error* (RMSE) [18] of the ratio $\hat{n}/n$, where n is the cardinality and $\hat{n}$ is its estimator. The RMSE reflects both the bias and the standard error in a single metrics since

$$\mathrm{RMSE}(\hat{n}/n) = \sqrt{(B(\hat{n}/n))^2 + (SE(\hat{n}/n))^2}. \quad (11)$$

We randomly generated a set of n distinct elements, and varied n from 4K to 16M to test the scalability of three counting algorithms. $\mathrm{RMSE}(\hat{n}/n)$ are calculated from 100 runs. Figure 5 shows the comparison of three algorithms with the same amount memory allocated. Both LogLog counting and adaptive counting algorithms use 64K buckets and 5 bits for each bucket, i.e. 320 Kbit memory. Linear counting uses a bitmap of the same memory size. The results show that linear counting performs quite well when n is less than 2M, but its RMSE increases dramatically from 0.0067 to 0.044 when n only increases from 2M to 3.4M. After n is greater than 4M, the bitmap becomes full and cannot estimate n correctly. The dotted line shows the predicated RMSE error of linear counting derived from Eq. (5) and (6). In contrast, the LogLog algorithm perform very well when n is larger than 192K, which corresponds to the *switching load factor*, 2.98, of our adaptive counting algorithm. LogLog counting can scale to very large cardinalities such as 16M with almost constant RMSE of 0.0052. However, its RMSE increases significantly from 0.0052 to 0.46 when n scales from 192K down to 32K, and further increases to 5.91 when n is 4K. The adaptive counting algorithm has the same scale up property as LogLog counting while it can scale down to small cardinalities as well. Its RMSE slightly decreased from 0.0052 to 0.0028 when n scales from 192K down to 4K, which is much better than the LogLog counting algorithm.

5.2 Trace-driven Simulation

We evaluated our CBTM scheme by using 16 IP header traces collected at an OC-192 link in Abilene network by the NLANR PMA project [2]. Each of these traces consists of 3.9M to 10.2M packets and 330.2K to 993.5K flows.

We simulated an ISP network with 16 OD routers and each of them is both ingress and egress routers. Since there are no synchronized network traces publicly available, we used the Abilene traces as incoming traffic at ingress routers and simulated the outgoing traffic at egress routers. We used a similar fanout distribution as measured in a Tier-1 ISP [3]. For each ingress router, we divided its 15 egress routers into three groups, i.e. large, medium and small, with 2, 6, and 7 routers in each group, respectively. Each group received approximately twice the traffic of the group behind it. We also simulated a random latency with mean t_l. We measured both PTM and FTM once every 10 seconds using 256K or 1024K buckets.

Figure 6 illustrates the estimated vs. actual PTM elements in 5 runs with 1024K buckets. 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 smallest PTM element is $77,909$ packets with an estimate of $80,128$ packets. The largest one is $2,363,136$ packets with an estimate of $2,387,547$ packets. 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 as we discussed later. Similarly, Fig. 6 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 smallest element is $1,098$ flows with an estimate of $1,079$ flows, while the largest one is 40,092 flows with an estimate of $40,081$ flows. 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.

To better understand the error distribution among TM elements, we adopted the *Root Mean Squared Relative Error* (RMSRE) metrics proposed by Zhang et al. [18]. $\mathrm{RMSRE}(T)$ is defined as

$$RMSRE(T) = \sqrt{\sum_{i=1, x_i > T}^{N} ((\hat{x_i} - x_i)/x_i)^2 / N_T} \quad (12)$$

where x_i is the i-th TM element and N is the total number of elements in TM. The $\mathrm{RMSRE}(T)$ only takes into account the error of TM elements greater than some threshold T, and compute the mean normalized by the number of elements greater than T, i.e., $N_T = \sum_{i=1, x_i > T}^{N} 1$.

In Fig. 8, we vary threshold T and compute $\mathrm{RMSRE}(T)$ compared with the percentage of packets or flows in TM elements above T. Fig. 8 shows large TM elements can be estimated much more accurately than small ones. As T increases, the percentage of traffic above T decreases and the accuracy of PTM and FTM estimation improves significantly. For example, when all traffic is taken into account,

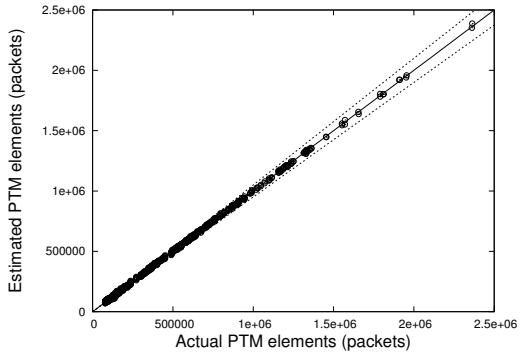

Figure 6: Actual vs. estimated PTM elements in the number of packets, m=1024K.

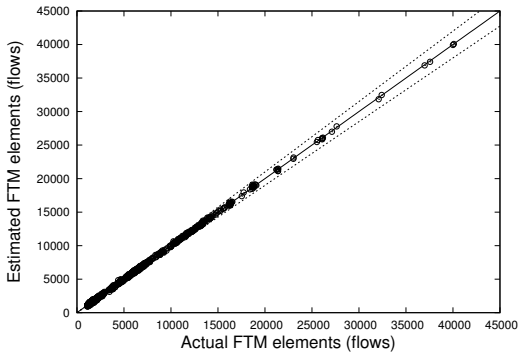

Figure 7: Actual vs. estimated FTM elements in the number of flows, m=1024K.

the RMSREs of PTM and FTM with 1024K buckets are 0.038 and 0.042, respectively. However, when we only consider 90% of the traffic in largest elements, they both decrease to 0.021 and 0.022 respectively. Further, 80% of the traffic only sees RMSRE of 0.012 and 0.013 for PTM and FTM, respectively. The similar trends hold for the estimation with 256K buckets. However, the overall error increases since less memory is used when compared with 1024K buckets.

We also evaluated the performance of *slot realignment* by varying average latency t_s from 0 to 1000 ms. Fig. 9 compares the RMSRE of estimated FTM with or without slot realignment. As we discussed in Sec. 4, the error will increase almost proportionally with the latency as demonstrated by the dashed line. The error bar shows the standard deviation of 20 runs. When slot realignment is used, the latency effect can be eliminated if t_l is less than t_k, which is expectable in practice since the latter is typically 10 second. The dotted line in Fig. 9 shows the RMSRE of PTM is almost constant when t_l increases from 0 to 1000 ms.

5.3 Performance of Packet Digesting

The performance of digesting packets into cardinality summary is very critical for our CBTM scheme to support high-speed links such as OC192 or OC468. We did a benchmark test on our prototype CBTM implementation in C on a DELL server with 3 GHz Xeon CPU and 1 GB memory. We use the hash function in *UMAC* [4] as our uniform hash function.

Figure 10 shows the cumulative distribution function of the CPU cycles used to digest each packet into cardinality summary with 256K or 1024K buckets. Each packet uses 1040 cycles on average for 256K buckets and 1041 cycles on average for 1024K buckets. Therefore, the software implementation of CBTM on a server with 3 GHz CPU can process more than 2.8 million packets per second (pps), i.e. 11 Gbps with packets of 500 bytes on average, which is sufficient for OC192. Moreover, current state-of-the-art hardware implementation of UMAC hash can yield a throughput of 79 Gbps [17]. Since our CBTM scheme digests each packet with one memory read and one write, it should be able to process 50 million pps with 10 ns SRAM and hardware acceleration.

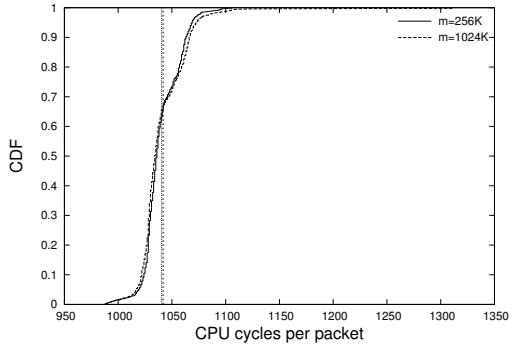

Figure 10: Performance of packet processing in CBTM at each router.

6 Further Discussion

Our CBTM approach focuses on distinct elements (packets or flows) observed by an OD router. Due to the hash function, duplicate elements will be eliminated before being digested into a cardinality summary. Therefore, the TM produced by CBTM may underestimate duplicate traffic between OD pairs. Traffic can get duplicated by network, or by intentional attacks. For the former, the impact is negligible, since network protocols rarely duplicate traffic excessively according to our definition of packet and flow identity. Attackers can duplicate identical packets or flows arbitrarily, but such behaviors can be easily detected by other means, e.g. correlating the increase in SNMP link count and PTM.

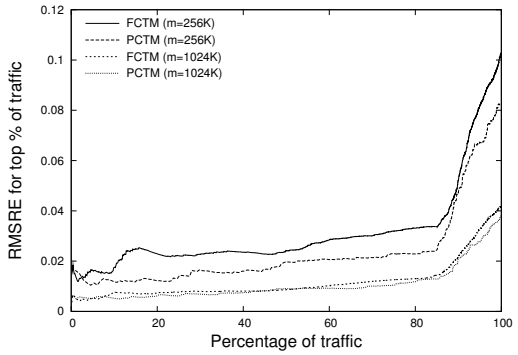

Figure 8: RMSRE of PTM and FTM elements with varying threshold T

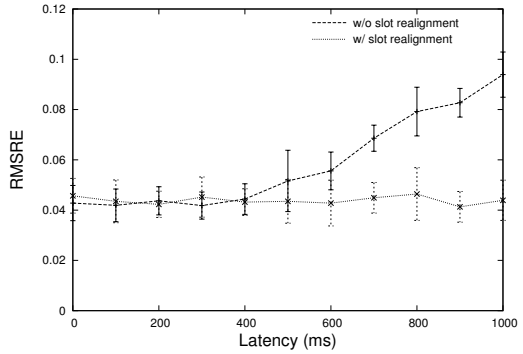

Figure 9: RMSRE of PTM elements with latency varying from 0 to 1000 ms, m=1024K.

Moreover, the TM estimation error may become noticeable as the traffic between an OD pair is extremely low. However, this is an unavoidable trade-off between accuracy and overhead. Our results show that the error is acceptable for most OD pairs with moderate traffic, which are of more interest to anomaly detection.

Further, with fast and accurate PTM and FTM offered by CBTM, as well as the readily available SNMP MIB, we can detect network anomaly by observing not only the spatial and temporal properties of PTM and FTM separately, but also their correlation. For example, in addition to excessive duplicates, a steep increase in PTM with insignificant one in FTM indicates more *elephant* flows, while a sudden increase in FTM with a moderate one in PTM indicates many small flows. The application of PTM and FTM is very broad, although anomaly detection usually has the most stringent requirement.

7 Conclusions

We have proposed a cardinality-based TM measurement approach with an adaptive counting algorithm to digest traffic into summary, estimate summary cardinality, and measure both PTM and FTM from summaries in a fast and accurate manner. As the experiment results suggest, our approach is well-suited for detecting network anomaly caused by DDoS flooding attacks and scanning worms, and is very useful for backbone ISPs to protect their assets on a network basis.

Our future work will focus on applying our fast CBTM scheme to detect, identify and trace network anomalies by correlating PTM, FTM and SNMP MIB information. We plan to evaluate the TM-based anomaly detection scheme by emulating large-scale DDoS flooding attacks and worm outbreaks on the DETER testbed [1].

References

[1] DETER Testbed. See http://www.isi.edu/deter/.

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

[3] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. POP-Level and Access-Link-Level Traffic Dynamics in a Tier-1 POP. In *ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2001.

[4] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. Umac: Fast and secure message authentication. *CRYPTO '99*, 1999.

[5] J. Cao, D. Davis, S. V. Weil, and B. Yu. Time-varying network tomography: Router link data. *J. of the American Statistical Assoc.*, 95(452), 2000.

[6] C.Tebaldi and M.West. Bayesian inference of network traffic using link count data. *J. of the American Statistical Assoc.*, 93(442), June 1998.

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

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

[9] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. of Comp. and Syst. Sci.*, 31(2):182–209, Oct. 1985.

[10] A. Medina, C. Fraleigh, N. Taft, S. Bhattacharyya, and C. Diot. A taxonomy of ip traffic matrices. In *SPIE ITCOM*, August 2002.

[11] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques compared and new directions. In *ACM SIGCOMM '02*, Aug. 2002.

[12] D. L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. on Netw.*, 3(3):245–254, 1995.

[13] K. Papagiannaki, N. Taft, and A. Lakhina. A distributed approach to measure ip traffic matrices. In *IMC'04*, pages 161–174, 2004.

[14] A. Soule, A. Nucci, R. Cruz, E. Leonardi, and N. Taft. How to identify and estimate the largest traffic matrix elements in a dynamic environment. In *SIGMETRICS'04*, 2004.

[15] Y. Vardi. Estimating source-destination traffic intensities fromlink data. *J. of the American Statistical Assoc.*, 91(433), March 1996.

[16] 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.

[17] B. Yang, R. Karri, and D. A. Mcgrew. An 80gbps fpga implementation of a universal hash function based message authentication code. Third Place Winner, 2004 DAC/ISSCC Student Design Contest, 2004.

[18] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *SIGMETRICS '03*, pages 206–217, 2003.

[19] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An information-theoretic approach to traffic matrix estimation. In *SIGCOMM '03*, pages 301–312, 2003.