Data Streaming Algorithms for Accurate and Efficient Measurement of Traffic and Flow Matrices

Qi (George) Zhao[†] Abhishek Kumar[†] Jia Wang[‡] Jun (Jim) Xu[†]

[†] College of Computing, Georgia Institute of Technology

[‡] AT&T Labs – Research

ABSTRACT

The traffic volume between origin/destination (OD) pairs in a network, known as traffic matrix, is essential for efficient network provisioning and traffic engineering. Existing approaches of estimating the traffic matrix, based on statistical inference and/or packet sampling, usually cannot achieve very high estimation accuracy. In this work, we take a brand new approach in attacking this problem. We propose a novel data streaming algorithm that can process traffic stream at very high speed (e.g., 40 Gbps) and produce traffic digests that are orders of magnitude smaller than the traffic stream. By correlating the digests collected at any OD pair using Bayesian statistics, the volume of traffic flowing between the OD pair can be accurately determined. We also establish principles and techniques for optimally combining this streaming method with sampling, when sampling is necessary due to stringent resource constraints. In addition, we propose another data streaming algorithm that estimates *flow matrix*, a finer-grained characterization than traffic matrix. Flow matrix is concerned with not only the total traffic between an OD pair (traffic matrix), but also how it splits into flows of various sizes. Through rigorous theoretical analysis and extensive synthetic experiments on real Internet traffic, we demonstrate that these two algorithms can produce very accurate estimation of traffic matrix and flow matrix respectively.

Categories and Subject Descriptors

C.2.3 [COMPUTER-COMMUNICATION NETWORKS]: Network Operations - Network Monitoring
E.1 [DATA STRUCTURES]

General Terms

Algorithms, Measurement, Theory

Keywords

Network Measurement, Traffic Matrix, Data Streaming, Sampling, Statistical Inference

SIGMETRICS'05, June 6–10, 2005, Banff, Alberta, Canada.

1. INTRODUCTION

A traffic matrix quantifies the volume of traffic between origin/destination (OD) pairs in a network. The problem of estimating traffic matrices has received considerable attention recently [2, 21, 22, 25, 26, 12, 20, 14, 8]. An accurate estimation of traffic matrices benefits a number of IP network management tasks such as capacity planning and forecasting, network fault/reliability diagnoses, provisioning for Service Level Agreements (SLAs), and routing configuration. For example, when a link fails, the network operators need to determine whether such an event will cause congestion based on the current traffic matrix and OSPF link weights, and re-optimize link weights if necessary.

The problem of measuring the traffic matrix TM can be formalized as follows. Assume that there are m ingress nodes[1] and n egress nodes in a network. We denote $TM_{i,j}$ as the total traffic volume traversing the network from the ingress node $i \in \{1, 2, \cdots, m\}$ to the egress node $j \in \{1, 2, \cdots, n\}$. The problem is to estimate the TM on a high-speed network in a specified measurement interval. One major contribution of this paper is an efficient and accurate scheme for measuring TM using data streaming (defined later) algorithms.

In addition, we notice that sometimes the traffic matrix is not yet fine-grained enough for some flow-oriented applications such as inferring the usage pattern of ISPs, detecting route-flapping, link failure, DDoS attacks, and Internet worms [5, 6, 11]. Traffic matrices only split total traffic volume among different OD pairs, not among different flows. In this paper we define a new term "flow matrix" which quantifies the traffic volume of flows between OD pairs in a network. Compared with traffic matrix, flow matrix is at a finer grained level and is more useful for the flow-oriented applications. Estimating flow matrices is the other important problem we are going to address in this work.

Existing approaches for estimating traffic matrix can be grouped into two categories: indirect and direct measurements. Conceptually, an indirect measurement approach infers the traffic matrix from related information that can be readily measured using existing network management infrastructure, such as link counts from SNMP. This approach typically relies on a traffic model and network status information. Techniques proposed in [2, 21, 22, 25, 26, 12, 14, 20] fall into this category. On the contrary, a direct approach, does not rely on any such model and state. Rather, the

[1]A node could be a link, router, PoP, or AS. Our schemes focus on estimating router-to-router matrices, but would also work at other levels of granularity. For example, instead of producing bitmaps at each router, we can get bitmaps produced at each link to infer the link-to-link TM or at backbone routers in each PoP to infer the PoP-to-PoP TM. For convenience, the notions of "router" and "node" are exchangeable in the rest of the paper.

estimation is based on direct observation of traffic at multiple nodes in the network. The trajectory sampling [4] approach and NetFlow-based approach [8][2] fall into this category. The direct approach is in general more accurate than the indirect approach, but incurs more measurement overhead. Our data streaming algorithms can be classified as a direct approach.

1.1 Main results and contributions

Our solution for estimating traffic and flow matrices is based on novel data streaming algorithms. Data streaming is concerned with processing a long stream of data items in one pass using a small working memory in order to answer a class of queries regarding the stream. The challenge is to use this small memory to "remember" as much information *pertinent to the queries* as possible. The main idea of our algorithms is to first perform data streaming on participating ingress and egress routers of the network. They generate streaming digests that are orders of magnitude smaller than the traffic stream. These digests are shipped to a central server on demand, when traffic/flow matrix needs to be estimated. We show that even with such small digests our schemes can obtain estimates with high accuracy.

Two data streaming algorithms are proposed, namely, *bitmap* based and *counter-array* based algorithms, for estimating traffic matrices and flow matrices, respectively. The data structure of the *bitmap* algorithm is extremely simple: an array of bits (bitmap) initialized to zero on each monitoring node. For each packet arrival at a participating node, the node simply sets the bit, indexed by the hash value of the part of this packet (described in Section 3.1), to 1 in that array. To estimate a traffic matrix element $TM_{i,j}$, two sets of bitmaps are collected from the corresponding nodes i and j, and are fed to a sophisticated estimator. Our key contribution here is the design and rigorous analysis of the accuracy of this estimator. The storage complexity of the algorithm is also reasonably low: 1$\sim$2 bits per packet[3]. We will show with a similar amount of storage complexity our scheme achieves better accuracy than the sampling-based schemes such as the work in [8]. This storage complexity can be further reduced through sampling. For maximizing the estimation accuracy when sampling is used, we also propose a technique for sampling packets consistently (instead of randomly independently) at multiple nodes and develop a theory that determines the optimal sampling rate under hard storage resource constraints.

As mentioned above, for estimating any matrix element $TM_{i,j}$, only the bitmaps from nodes i and j are needed. This allows us to estimate a submatrix using the minimum amount of information possible, namely, only the bitmaps from the rows and columns of the submatrix. This feature of our scheme is practically important in two aspects. First, in a large ISP network, most applications are often interested in only a portion of elements in the traffic matrix instead of the whole traffic matrix. Second, this feature allows for the incremental deployment of our scheme since the existence of non-participating nodes does not affect the estimation of the traffic submatrix between all participating ingress and egress nodes.

Our second streaming algorithm, namely *counter-array* scheme, is proposed for estimating the aforementioned flow matrix. With an array of counters as its data structure, its operation is also very simple. For each packet arrival, the counter indexed by the hash value of its flow label is incremented by 1. We show that the medium and large elements of the flow matrix can be inferred fairly accurately through correlating the counter arrays at all ingress and egress nodes. This algorithm can also be used for estimating the traffic matrix (weaker than flow matrix), though it is less cost effective than the bitmap scheme for this purpose.

[2]Actually this is a "semi-direct" approach which still needs routing information.

[3]Depending on the application that uses traffic matrix information, the bitmaps typically do not need to be stored for more than a day.

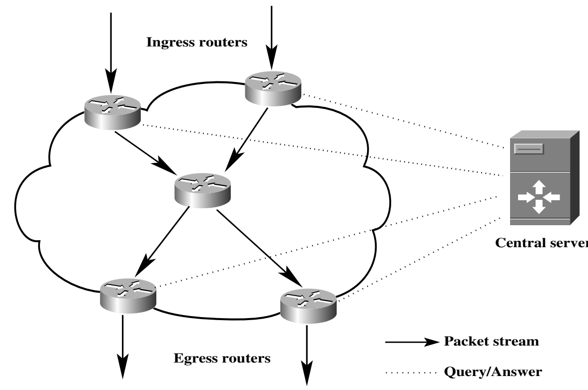


Figure 1: System model

1.2 Paper outline

The rest of this paper is organized as follows: we start in Section 2 with a system overview and performance requirements we need to achieve. Sections 3, 4, and 5 describe our main ideas and provide a theoretical analysis. This is followed by synthetic experimental evaluation on real-world Internet traffic and actual traffic matrices in Section 6. The related work is described in Section 7. Finally, we conclude the paper in Section 8.

2. SYSTEM OVERVIEW AND PERFORMANCE REQUIREMENTS

The common system architecture of both the bitmap scheme and the counter array scheme is shown in Figure 1. The participating nodes will run an online streaming module that produces digests that are thousands of times smaller than the raw traffic. These digests will be stored locally for a period of time, and will be shipped to a central server on demand. A data analysis module running at the central server obtains the digests needed for estimating the traffic matrix and flow matrix through queries. Sophisticated decoding algorithms are used to obtain a very accurate estimation from these digests.

Our work is the first effort that uses data streaming algorithms for traffic (and flow) matrix estimation. The key challenge in this effort is to satisfy the following four seemingly conflicting performance requirements simultaneously:

1. Low storage complexity. The amount of data digests generated on a single monitored node (per second) needs to be as small as possible since each node may be required to store such a digest for future inquiry for a certain period of time. Our (baseline) algorithms achieve more than three orders of magnitude reduction on the raw traffic volume. In the bitmap scheme, more reduction can be achieved using sampling (to be described in Section 4) and the inaccuracies of sampling is minimized through our consistent sampling method and parameter optimization theory.

2. Low memory (SRAM) complexity. As in other data streaming algorithms, our estimation accuracy becomes higher when the available memory (SRAM) becomes larger. However, we would also like to minimize this memory size since SRAM is an expensive and scarce resource on routers. We will show that both of our schemes provide very high accuracy using a reasonable amount of

```
Initialize
    B[k] := 0, k = 1, 2, ..., b

Update
    Upon the arrival of a packet pkt
        ind := h(φ(pkt));
        B[ind] := 1;
```

Figure 2: Algorithm for updating the online streaming module

SRAM (e.g., 512KB).

3. Low computational complexity. The online streaming module should be fast enough to keep up with high link speeds such as 10 Gbps (OC-192) or even 40 Gbps (OC-768). We will show that our schemes are designed to address this challenge: we use efficient hardware implementation of hash functions and for each packet the bitmap scheme needs only one memory write and counter-array scheme needs only one memory read and write (to the same location). When coupled with sampling in the bitmap scheme, even higher rates can be supported.

4. High accuracy of estimation. Despite the fact that the digests produced at each node are orders of magnitude smaller than the original traffic stream, we would like our estimation of the traffic matrix in a measurement interval to be as close to the actual values as possible. We propose sophisticated "decoding" algorithms that produce estimates much more accurate than existing approaches, using these small digests.

3. THE BITMAP BASED ALGORITHM

We first present the online streaming module and the data analysis module of our bitmap based scheme. Then, we address the issues of clock synchronization, measurement epoch alignment, and heterogeneity in router speeds that arise in the practical operation of our scheme.

3.1 Online streaming module

The operations of the online streaming module are shown in Figure 2. The data structure is very simple: a bitmap B indexed by a hash function h. When a packet p arrives, we extract the invariant portion of the packet (denoted as $\phi(p)$ and described later) and hash it using h (we will discuss the choice of the hash function later). The result of this hashing operation is an integer which is viewed as an index into B and the bit at the corresponding index is set to 1. The bitmap is set to all 0's initially and will be paged to disk when it is filled to a threshold percentage (discussed in Section 4). We define this time interval as a "bitmap epoch". This algorithm runs at all participating ingress and egress nodes, using the same hash function h and the same bitmap size b.

The invariant portion of a packet used as the input to the hash function must uniquely represent the packet and by definition should remain the same when it travels from one router to another. At the same time, it is desirable to make its size reasonably small to allow for fast hash processing. In our scheme, the invariant portion of a packet consists of the packet header, where the variant fields (e.g., TTL, ToS, and checksum) are marked as 0's, and the first 8 bytes of the payload if there is any. As shown in [19], these 28 bytes are sufficient to differentiate almost all non-identical packets.

For this application we need a uniform hash function that is amenable to hardware implementation[4] and can be computed very fast. The H_3 family of hash functions proposed by Carter and Wegman [3] satisfies this requirement. It can produce hash result in a few nanoseconds with straightforward hardware implementation [17]. The design of the H_3 family of hash functions is described in Appendix .4.

This online streaming module is extremely low in both computational and storage complexities:

Computational complexity. Each update only requires one hash function computation and one write to the memory. Using hardwired H_3 hash functions and $10ns$ SRAM, this would allow around 50 million packets per second, thereby supporting 50 Gbps traffic stream[5]. To support OC-192 speed (10 Gbps), we only need to use DRAM, since each packet has $100ns$ time budget.

Storage complexity. For each packet our scheme only produces a little more than one bit as its digest, which is three orders of magnitude reduction compared to the original traffic stream. For an OC-192 link, about 1∼2 MB of digests will be generated and stored every second. An hour's worth of digests are about 100 MB. This can be further reduced using sampling, at the cost of reduced accuracy (to be discussed in Section 4).

3.2 Data analysis module

When we would like to know $TM_{i,j}$ during a certain time interval, the bitmaps corresponding to the bitmap epochs contained or partly contained in that interval (Figure 3) will be requested from nodes i and j and shipped to the central server. Next, we present and analyze an estimator of $TM_{i,j}$ given these bitmaps. For simplicity of discussion, we assume an ideal case where both node i and node j produce exactly one bitmap during that time interval (a measurement interval) and defer other details of the general case to Section 3.3.

Our estimator is adapted from [23], which was proposed for a totally different application (in database). In addition, we also provide a rigorous analysis of its standard deviation/error (not studied in [23] and is very involved). This analysis not only quantifies the accuracy of the estimator, an important result by itself, but also is an important step in identifying the optimal sampling rate when the online streaming algorithm operates under hard resource (storage) constraints.

Let the set of packets arriving at the ingress node i during the measurement interval be T_i and the resulting bitmap be B_{T_i}. Let U_{T_i} denote the number of bits in B_{T_i} that are 0's. Recall that the size of the bitmap is b. A good estimator[6] of $|T_i|$, the number of elements (packets) in T_i, adapted from [23], is

$$D_{T_i} = b \ln \frac{b}{U_{T_i}} \tag{1}$$

$TM_{i,j}$, the quantity we would like to estimate, is simply $|T_i \bigcap T_j|$. An estimator for this quantity, also adapted from [23], is

$$\widehat{TM}_{i,j} = D_{T_i} + D_{T_j} - D_{T_i \cup T_j} \tag{2}$$

Here $D_{T_i \cup T_j}$ is defined as $b \ln \frac{b}{U_{T_i \bigcup T_j}}$, where $U_{T_i \bigcup T_j}$ denotes the number of 0's in $B_{T_i \bigcup T_j}$ (the result of hashing the set of pack-

[4]We do not use cryptographically strong hash functions such as MD5 or SHA, which are much more expensive to compute, since their security properties such as collision-resistance are not needed in this application.

[5]We assume a conservative average packet size of 1,000 bits, to our disadvantage. Measurements from real-world Internet traffic report much larger packet sizes.

[6]Note that, although D_{T_i} (as well as D_{T_j} and $D_{T_i \cup T_j}$) is an estimator, we do not put "hat" on top of it since it is just a component of the main estimator we are interested in.

ets $T_i \bigcup T_j$ into a single bitmap). The bitmap $B_{T_i \bigcup T_j}$ is computed as the bitwise-OR [7] of B_{T_i} and B_{T_j}. It can be shown that $D_{T_i}+D_{T_j}-D_{T_i \bigcup T_j}$ is a good estimator of $|T_i|+|T_j|-|T_i \bigcup T_j|$, which is exactly $|T_i \bigcap T_j|$. [8]

The computational complexity of estimating each element of the matrix is $O(b)$ for the bitwise operation of the two bitmaps. The overall complexity of estimating the entire $m \times n$ matrix is therefore $O(mnb)$. Note that the bitmaps from other nodes are not needed when we are only interested in estimating $TM_{i,j}$. This poses significant advantage in computational complexity over existing indirect measurement approaches, in which the whole traffic matrix needs to be estimated even if we are only interested in a small subset of the matrix elements due to the holistic nature of the inference method. This feature also makes our scheme incrementally deployable, as mentioned earlier.

While this estimator has been briefly mentioned in [23], there is no rigorous analysis of its standard deviation and error, which we perform in this work. These are characterized in the following theorem. Let t_{T_i}, t_{T_j}, $t_{T_i \bigcap T_j}$, and $t_{T_i \bigcup T_j}$ denote $\frac{|T_i|}{b}$, $\frac{|T_j|}{b}$, $\frac{|T_i \bigcap T_j|}{b}$, and $\frac{|T_i \bigcup T_j|}{b}$, respectively. They are the "load factors" of the array when the corresponding set of packets are hashed into the array (of size b). Its proof is provided in Appendix .2.

THEOREM 1. The variance of $\widehat{TM_{i,j}}$ is given by

$$Var[\widehat{TM_{i,j}}] = b(2e^{t_{T_i \cap T_j}} + e^{t_{T_i \cup T_j}} - e^{t_{T_i}} - e^{t_{T_j}} - t_{T_i \cap T_j} - 1).$$

The average (relative) error of the estimator $\widehat{TM_{i,j}}$, which is equal to the standard deviation of the ratio $\frac{\widehat{TM_{i,j}}}{TM_{i,j}}$ since this estimator is almost unbiased (discussed in Appendix .1), is given by

$$\frac{\sqrt{2e^{t_{T_i \cap T_j}} + e^{t_{T_i \cup T_j}} - e^{t_{T_i}} - e^{t_{T_j}} - t_{T_i \cap T_j} - 1}}{\sqrt{b}\, t_{T_i \cap T_j}}. \quad (3)$$

Equation 3 characterizes the tradeoff between memory complexity and estimation accuracy for the bitmap scheme as follows. The average error is scaled by the inverse of $\sqrt{b}$, which means the larger the page size the more accurate the result we get. Our experiments in Section 6 show that very high estimation accuracy can be achieved using a reasonable amount of SRAM (e.g., 512 KB).

Our estimator $\widehat{TM_{i,j}}$ actually estimates the number of distinct packets in $TM_{i,j}$. If two or more identical packets occur during *the same bitmap epoch*, only one will be counted in $\widehat{TM_{i,j}}$ since they are hashed to the same bit. TCP retransmissions are the dominant cause of identical packets. Since it is shown in [10] that TCP retransmissions represent about 2% of the TCP traffic, we should not ignore this portion of traffic. Therefore, in estimating the actual value of $TM_{i,j}$, our scheme compensates for such identical/retransmitted packets as follows. We maintain a counter C that counts the total number of packets on node i in a measurement epoch, including retransmitted ones. Recall that D_{T_i} estimates T_i, the number of distinct packets going through ingress node i to all egress nodes. Clearly $C - D_{T_i}$ is an estimator of the total number of retransmitted packets. Then we simply scale $\widehat{TM_{i,j}}$ by a factor of $\frac{C}{D_{T_i}}$ to obtain the final estimate of $TM_{i,j}$, assuming that retransmissions are proportionally distributed among traffic matrix elements $TM_{i,1}$, $TM_{i,2}$, ..., and $TM_{i,n}$.

3.3 Extension for operational issues

The above estimation procedure and its accuracy analysis are for the ideal case with the following three assumptions:

(i) *The measurement interval is exactly one bitmap epoch.* Practically some network management tasks such as capacity planning and routing configuration need the traffic matrices on the long time scales such as tens of minutes or a few hours. Each epoch in our measurements is typically much smaller especially for the high speed links. Therefore we need extend our scheme to support any time scales.

(ii) *The clocks on nodes i and j are perfectly synchronized.* Using GPS synchronization [18], or more cost-effective schemes [15], clocks at different nodes only differ by tens of microseconds. Since each bitmap epoch is typically one to several seconds with OC-192 or OC-768 speeds (even longer for lower link speeds), the effect of clock skew on our measurement is negligible.[9] Due to the high price of GPS cards today, the standard network time protocol (NTP) is most commonly used to synchronize clocks. As we will shown later in this section, our measurements still work accurately with relative large clock skews (e.g., tens of milliseconds as one may get from clocks synchronized by using NTP).

(iii) *The bitmap epochs between nodes i and j are well aligned.* Traffic going through different nodes can have rates orders of magnitude different from each other, resulting in some bitmaps being filled up very fast (hence short bitmap epoch) and some others filled up very slowly (hence long bitmap epoch). We refer to this phenomena as *heterogeneity*. Because of heterogeneity the bitmap epochs on different nodes may not well aligned.

Next, we solve for the general case in which these assumptions are eliminated. We assume that the measurement interval spans exactly bitmap epochs 1, 2, ..., k_1 at node i and bitmap epochs 1, 2, ..., k_2 at node j, respectively. Then the traffic matrix element $TM_{i,j}$ can be estimated as

$$\widehat{TM_{i,j}} = \sum_{q=1}^{k_1} \sum_{r=1}^{k_2} \widehat{N_{q,r}} \times overlap(q,r) \quad (4)$$

where $\widehat{N_{q,r}}$ is the estimation of the common traffic between the page[10] q at node i and the page r at node j, and $overlap(q,r)$ is 1, when the page q at node i overlaps temporally with page r at node j, and is 0 otherwise. To determine whether two bitmap epochs overlap with each other temporally, the timestamps of their starting times will be stored along with the pages. We name the above method "multipaging". We show that the multipaging actually completely eliminates the aforementioned three assumptions.

Clearly the assumption (i) is eliminated because the multipaging supports the measurements over multiple epochs. Now we further adapt the multipaging to the case that the measurement interval does not necessarily align with the epoch starting times (assumption (iii)). We illustrate this using an example shown in Figure 3. A measurement interval corresponds to the rear part of epoch 1, epochs 2 and 3, and the front part of epoch 4 at node i. It also corresponds to the rear part of epoch 1, epoch 2, and the front part of epoch 3 at node j. By Equation 4, we need to add up the terms $\widehat{N_{1,1}}, \widehat{N_{2,1}}, \widehat{N_{2,2}}, \widehat{N_{3,2}}, \widehat{N_{3,3}}$, and $\widehat{N_{4,3}}$ based on their tem-

[7] One can easily verify the correctness of the computation with respect to the semantics of $B_{T_i \bigcup T_j}$.

[8] Directly using Equation 1 based on the bitmap produced by bitwise-ANDing B_{T_i} and B_{T_j} to get an estimator is problematic due to random hashing.

[9] For the same reason the impact of clock resolution on our measurements is also negligible.

[10] The notions of "page" and "bitmap" are exchangeable in the rest of this paper.

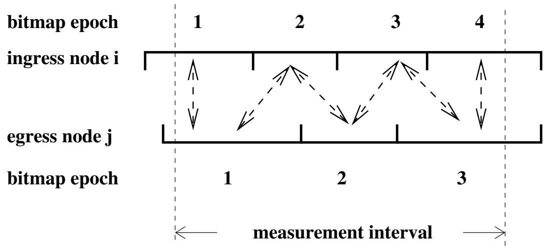


Figure 3: Example of timeline

poral overlap relationships. However, this would be more than $TM_{i,j}$ because the measurement interval only has the rear part of epoch 1 and the front part of epoch 4 at node i. Our solution is to adjust $\widehat{N_{1,1}}$ to the proportion of the epoch 1 that overlaps with the measurement interval. $\widehat{N_{4,3}}$ will also be adjusted proportionally accordingly. Since traffic matrix estimation is typically on the time scales of tens of minutes, which will span many pages, the inaccuracies resulting from this proportional rounding are negligible.

The assumption (ii) can be eliminated by combing the clock skew factor into definition of "temporal overlapping" in Equation 4. We still use an example shown in Figure 3. The epoch 1 at node i does not overlap temporally with the epoch 2 at node j visually. But if the interval between the end of the epoch 1 at node i and the start of the epoch 2 at node j is smaller than an upper bound T_1 of the clock skew (e.g., $50ms$ for an NTP enabled network), we still consider they are temporally overlapping.

By now there is still a remaining problem for the extension of Equation 4: the packets in transit. Let us come back to the example in Figure 3. If there are packets departing from i in epoch 1 (at node i) and arriving at j in epoch 2 (at node j) due to nontrivial traversal time from i to j, our measurement will miss these packets because only $\widehat{N_{1,1}}$ is computed. This can be easily fixed using the same method used above to eliminate the assumption (ii), i.e., combining another upper bound T_2 of the traversal time (e.g., $50ms$) to define "temporal overlapping". In other words if the interval between the end of epoch 1 at node i and the start of epoch 2 at node j is within $T_1 + T_2$, it should be labeled "temporal overlapping" ($overlap(1, 2) = 1$) and join the estimation.

4. SAMPLING

Sometimes the bitmaps need to be stored for a long period of time for later troubleshooting. This could result in huge storage complexity for very high speed links. Sampling can be used to reduce this requirement significantly. Also, if we would like to use DRAM to conduct online streaming for very high speed links (e.g., beyond OC-192), it is important to sample only a certain percentage of the packets so that the DRAM speed can keep up with the data stream speed. However, we need to bear in mind that sampling comes at the cost of reduced accuracy. In this section, we rigorously analyze the impact of sampling on accuracy, and address two challenging problems that arise in minimizing this impact: *sampling versus squeezing* and *consistent sampling*.

4.1 Sampling versus squeezing

Suppose there is a hard resource constraint on how much storage the online streaming algorithm can consume every second. For example, the constraint can be one bitmap of 4 Mbits per second. Suppose we have 40 million packets arriving within one second. One option is that we do no sampling and hash all these packets into the bitmap, referred to as "squeezing". But the resulting high load factor of approximately 10 would lead to high estimation error according to Equation 3. An alternative option is to sample only a certain percentage p of packets to be squeezed into the bitmap. We have many different p values to choose from. For example, we can sample 50% of the packets and thereby squeeze 20 million sampled packets into the bitmap, or we can sample and squeeze only 25% of them. This comes to the question which p is optimal. On the one extreme, if we sample at a very low rate, the bitmap will only be lightly loaded and the error of estimating the total *sampled* traffic as well as its common traffic with another node (a traffic matrix element) becomes lower. However, since the sampled traffic is only a small percentage of the total traffic, the overall error will be blown up by a large factor (discussed in Section 4.2). On the other extreme, if we sample with very high probability, the error from sampling becomes low but the error from estimating the sampled traffic becomes high. We establish the following principle for conducting sampling that aims at reaching a "sweet spot" between these two extremes.

PRINCIPLE 1. *If the expected traffic demand in a bitmap epoch does not make the resulting load factor exceed t^*, no sampling is needed. Otherwise, sampling rate p^* should be set so that the load factor of the sampled traffic on the bitmap is approximately t^*.*

We consider the following scenario throughout the rest of this section. We assume that each ingress and egress node will coordinate to use the same load factor t after sampling. This coordination will allow us to optimize t for estimating most of the traffic matrix elements accurately. We show how to minimize the error of an arbitrary $\widehat{N_{q,r}}$ term shown in Equation 4. Recall that $N_{q,r}$ is the amount of common traffic between two overlapping bitmap pages, page q at node i and page r at node j. We simply denote it and its estimator as X and $\hat{X}$, respectively. Also, let α and p_α be the aforementioned page q at node i and the corresponding sampling rate, respectively. Similarly, β and p_β denote page r at node j and the corresponding sampling rate, respectively. Note that each overlapping page pair may have its own optimal t^* to achieve the optimal accuracy of estimating its common traffic. Therefore it is impossible to adapt t^* to satisfy every other node, as their needs (t^* for optimal accuracy) conflict with each other. The goal of our following analysis is to identify a default t^* for every node such that the estimation accuracy for the most common cases is high.

We first study the optimal p^* and t^* between pages α and β given the expected traffic demand in a bitmap epoch. In fact, only one of them needs to be determined since the other follows from Principle 1. In Section 4.2 we will propose a sampling technique called *consistent sampling* to significantly reduce the estimation error. With consistent sampling, $\hat{X}$, the estimator of X, is given by $\frac{\hat{N}}{p}$, where $p = min(p_\alpha, p_\beta)$ and $\hat{N}$ is the estimation result (by Equation 2) on the sampled traffic. We denote the total sampled traffic volume squeezed into the page with sampling rate p by T. The following theorem characterizes the variance of $\hat{X}$. Its proof is provided in Appendix .3.

THEOREM 2. The variance of $\hat{X}$ is approximately given by

$$\frac{b}{p^2}\left((e^{\frac{Tp}{b}-\frac{Xp}{2b}} - e^{\frac{Xp}{2b}})^2 + e^{\frac{Xp}{b}} - \frac{Xp}{b} - 1\right) + \frac{X(1-p)}{p}.$$

Remark: The above formula consists of two terms. We show in Appendix .3 that the first term corresponds to the variance from estimating the sampled traffic (Equation 2) scaled by $\frac{1}{p^2}$ (to compensate for the sampling), and the second term corresponds to the

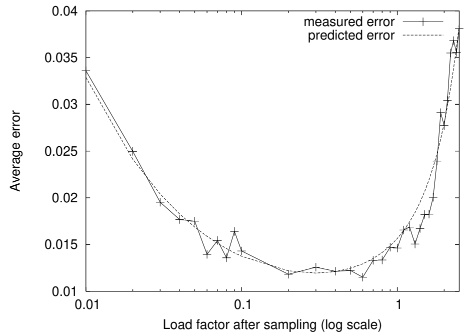


Figure 4: Measured average error from Monte-Carlo simulation vs. the value from Equation 5 (X = 1M packets, T = 10M packets, and b = 1 Mbits).

variance of the sampling process. Since these two errors are orthogonal to each other, their total variance is the sum of their individual variances as shown in Appendix .3.

The average error of $\hat{X}$, which is equal to the standard deviation of the ratio $\frac{\hat{X}}{X}$ since $\hat{X}$ is an almost unbiased estimator of X (discussed in Appendix .1), is given by

$$\frac{\sqrt{\frac{b}{p^2}\left((e^{\frac{Tp}{b}-\frac{Xp}{2b}} - e^{\frac{Xp}{2b}})^2 + e^{\frac{Xp}{b}} - \frac{Xp}{b} - 1\right) + \frac{X(1-p)}{p})}}{X}. \tag{5}$$

We perform Monte-Carlo simulations to verify the accuracy of the above formula since there is some approximation in its derivation (see Appendix .3). The following parameter settings are used in the simulations. The size b of the bitmap is 1M bits and both the ingress page and the egress page have the same load factor 10 without sampling. In other words, 10M packets each is to be squeezed, or sampled and then squeezed, into the bitmaps. Among both sets of 10M packets, 1M packets are common between both the ingress page and the egress page (i.e., $X = 1$M packets).

Figure 4 shows that the observed average error from Monte-Carlo simulation matches well with the value from Equation 5. The curve verifies our intuition above about the estimation error caused by sampling and squeezing. When the load factor is very low, the sampling error dominates; when the load factor becomes large the error caused by "excessive squeezing" is the main factor. At the optimal t^* (≈ 0.4) the scheme achieves the smallest average relative error (≈ 0.012).

Since the optimal t^* value is a function of T and X, setting it according to some global default value may not be optimal all the time. Fortunately we observe that t^*, the optimal load factor, does not vary much for different T and X values through our extensive experiments. In addition, we can observe from Figure 4 that the curve is quite flat in a large range around the optimal load factor. For example, the average errors corresponding to any load factor between 0.09 and 1.0 only fluctuate between around 0.012 to 0.015. Combining the above two observations, we conclude that by setting a global default load factor t^* according to some typical parameter settings, the average error will stay very close to optimal values. Throughout this work we set the default load factor to 0.7.

4.2 Consistent sampling

If the sampling is performed randomly and independently, only $p_\alpha p_\beta$ of the common traffic that comes from page α to page β is recorded by both nodes on the average. To estimate X, although it is possible to get its unbiased estimate by blowing the estimation of the sampled portion up by $\frac{1}{p_\alpha p_\beta}$, it also blows the error of our estimate up by $\frac{1}{p_\alpha p_\beta}$. To address this problem, we propose a consistent sampling scheme which has the following desirable property. When $p_\alpha \leq p_\beta$, among the set of packets that come from page α to page β, we ensure that those sampled and squeezed into page α are a subset of those sampled and squeezed into page β, and vice versa. In this way, $min(p_\alpha, p_\beta)$ of traffic between page α and page β will be sampled by both nodes and the error of our estimation will only be blown up by $\frac{1}{min(p_\alpha, p_\beta)}$ times.

Our consistent sampling scheme works as follows. We fix a hash function h' (different from the aforementioned h which is used to generate the bitmap), that maps the invariant portion of a packet to an l-bit binary number. The range of the hash function h' is $\{0, 1, \cdots, 2^l - 1\}$. If a node would like to sample packets with rate $p = \frac{c}{2^l}$, it simply samples the set of packets $\{pkt | h'(\phi(pkt)) < c\}$. We make l sufficiently large such that any desirable sampling rate p can be approximated by $\frac{c}{2^l}$ for some c between 1 and 2^l. When every node uses the same hash function h', the above property is clearly achieved as follows. When $p_\alpha = \frac{c_1}{2^l} \leq \frac{c_2}{2^l} = p_\beta$, the set of packets sampled and squeezed into page α, $\{pkt | h'(\phi(pkt)) < c_1\}$, is clearly a subset of those sampled and squeezed into page β, $\{pkt | h'(\phi(pkt)) < c_2\}$.

Initialize
    $C[i] := 0, i = 1, 2, ..., b$;

Update
    Upon the arrival of a packet pkt
        ind := h($pkt.flow\_label$);
        $C[ind] := C[ind] + 1$;

Figure 5: Algorithm for updating the online streaming module

5. THE COUNTER-ARRAY BASED SCHEME

In this section, we formally introduce the concept of flow matrix, which contains finer grained information than traffic matrix, and present our counter-array based scheme for estimating flow matrix. *Flow matrix* is defined as the traffic matrix combined with the information on how each OD element is splitted into flows of different sizes. Formally, a flow matrix element $FM_{i,j}$ is the set of sizes of flows that travel from node i to node j during a measurement interval. A traffic matrix element $TM_{i,j}$ is simply the summation of all the flow sizes in $FM_{i,j}$, that is, $TM_{i,j} = \sum_{s \in FM_{i,j}} s$. Thus our counter-array scheme also works for estimating traffic matrices.

5.1 Online streaming module

The online streaming algorithm (shown in Figure 5) uses a very simple data structure: an array of counters C. Upon arrival of a packet, its flow label[11] is hashed to generate an index into this array, and the counter at this index is incremented by 1. Similar to the bitmap scheme, all nodes employ an array of the same size b and the same hash function h. Since for each packet, this algorithm requires only one hash operation, one memory read and one memory write (to the same location), this allows our scheme to operate at

[11] Our design does not place any constraints on the definition of flow label. It can be any combination of fields from the packet header.

OC-768 (40 Gbps) speed with off-the-shelf $10ns$ SRAM and efficient hardware implementation of the H_3 family of hash functions.

Due to the delicate nature of the data analysis algorithm (discussed Section 5.2) for the counter array scheme, much more stringent yet still reasonable constraints are placed on the online streaming module. First, unlike in the bitmap scheme, the counter array scheme is holistic in the sense that all ingress and egress nodes have to participate. Second, unlike in the bitmap scheme, the *counter epochs* (defined next) in this scheme need to be aligned with each other, that is, all counter epochs in all ingress and egress nodes need to start and end at the approximately same time. The practical implication behind this is the counter array b needs to be large enough to accommodate the highest link speed among all nodes (i.e., the worst case). Similar to the definition of "bitmap epoch", we refer to the amount of time the highest-speed link takes to fill up the counter array to a threshold percentage as a "counter epoch", or epoch for abbreviation.

Next, we analyze the memory (SRAM) and storage complexities of the online streaming module.

Memory complexity. Depending on the maximum link speed among all the nodes, the counter epoch ranges from 1 to a few 10's of seconds. We show in Section 6 that, very accurate estimation can be achieved by setting the number of counters in the array to around the same order as the number of flows during an epoch. Therefore, for OC-192 or OC-768 link, one to a few million of counters need to be employed for a measurement interval of one to a few seconds. If each counter has a "safe" size of 64 bits to prevent the overflow[12], the memory requirement would be quite high. Fortunately, leveraging a technique invented in [16], this requirement is reduced to 9 bits per counter, an 85% reduction. For example, a million counters will only cost 1.1 MB SRAM. The key idea of this technique is to keep short counters in SRAM and long counters in DRAM. When a short counter in SRAM exceeds a certain threshold value due to increments, the value of this counter will be "flushed" to the corresponding long counter in DRAM.

Storage complexity. Perhaps surprisingly, at the same link speed, the storage complexity is even smaller than the bitmap scheme. In the bitmap scheme, each packet results in 1 to 2 bits of storage. Here, each flow results in 64 bits of storage. However, since most of the counter values are small, resulting in a lot of repetitions in small counter values, Huffman type of compression [9] can easily reduce the storage complexity to only a few bits per counter. Since the average flow length is about 10 packets (observed in our experiments), the average storage cost per packet is amortized to less than 1 bit.

5.2 Data analysis module

Once there is a need to estimate the flow matrix during a measurement interval, the counter arrays during that interval need to be shipped to the central server for analysis. If the measurement interval spans more than one epochs, digests in each epoch will be processed independently. In this section, we present our data analysis algorithm that infers the flow matrix from these counter arrays during a single epoch.

We first describe the intuition behind our algorithm. Let $I^k = \{C_{I_1}[k], C_{I_2}[k], \cdots, C_{I_m}[k]\}$ where $C_{I_i}[k], i = 1, \cdots, m$, is the value of the k^{th} counter at the ingress node I_i, and $E^k = \{C_{E_1}[k], C_{E_2}[k], \cdots, C_{E_n}[k]\}$ where $C_{E_j}[k], j = 1, \cdots, n$, is the value of the k^{th} counter at the egress node E_j. Since every node uses the same hash function, packets recorded in I^k have an approximate one-to-one correspondence with packets recorded in E^k.

$$\sum_{i=1}^{m} C_{I_i}[k] \approx \sum_{j=1}^{n} C_{E_j}[k] \qquad (6)$$

This approximation comes from the fact that the clock is not perfectly synchronized at all nodes and the packet traversal time from an ingress node to an egress node is non-zero. Similar to the discussion in Section 3.3, both factors only have marginal impact on the accuracy of our estimation. In addition, the impact of this approximation on the accuracy of our algorithm is further alleviated due to the "elephant matching" nature of our algorithm (discussed later).

Now, let us consider an ideal case in which there is no hash collision on index k in all counter arrays, and therefore the counter value represents a flow of this size. We further assume that: (i) the number of ingress nodes is the same as the egress nodes (i.e., $m = n$); (ii) the elements in I^k are all distinct; and (iii) the flows in I^k all go to distinct elements of E^k. Then the values in E^k are simply a permutation of the values in I^k. A straightforward "one-to-one matching" will allow us to infer this permutation.

In reality, none of the above assumptions is true. At an ingress node, multiple flows can collide into one counter. Flows from multiple ingress nodes can collide into the same counter at an egress node, and m is in general not equal to n. Therefore, a "naive" matching algorithm like above will not work in general. However, since there are only a small number of medium to large flows due to the Zipfian nature of the Internet traffic, matching kangaroos (medium flows) and elephants (large flows) between ingress and egress nodes turns out to work very well. As expected, it does not work well on small flows.

1. $FM^k_{i,j} := 0, i = 1, \cdots, m, j = 1, \cdots, n;$
2. **do**
3. $\quad max\_i := \underset{i}{argmax}\{C_{I_i}[k], i = 1, 2, \cdots, m\};$
4. $\quad max\_j := \underset{j}{argmax}\{C_{E_j}[k], j = 1, 2, \cdots, n\};$
5. $\quad$ **if** $C_{I_{max\_i}}[k] \geq C_{E_{max\_j}}[k]$
6. $\qquad FM^k_{max\_i,max\_j} := FM^k_{max\_i,max\_j} + C_{E_{max\_j}}[k];$
7. $\qquad C_{I_{max\_i}}[k] := C_{I_{max\_i}}[k] - C_{E_{max\_j}}[k];$
8. $\qquad C_{E_{max\_j}}[k] := 0;$
9. $\quad$ **else**
10. $\qquad FM^k_{max\_i,max\_j} := FM^k_{max\_i,max\_j} + C_{I_{max\_i}}[k];$
11. $\qquad C_{E_{max\_j}}[k] := C_{E_{max\_j}}[k] - C_{I_{max\_i}}[k];$
12. $\qquad C_{I_{max\_i}}[k] := 0;$
13. **while** $((C_{I_{max\_i}}[k] > 0) \&\& (C_{E_{max\_j}}[k] > 0))$

Figure 6: Generate matching of counter values at index k

Figure 6 shows a greedy algorithm for matching elephants and kangaroos. For each index k in all the counter arrays we perform the following iteration (lines 2 to 13). We first match the largest ingress counter value $C_{I_{max\_i}}[k]$ with the largest egress counter value $C_{E_{max\_j}}[k]$. The smaller value of the two is considered a flow from max_i to max_j (lines 6 and 10), and this value will be subtracted from both counter values (lines 7 and 11). This clearly reduces the smaller counter value to 0. The above procedure is repeated until either all ingress counters or all egress counters at index k become 0's[13]. When there is a tie on the maximum ingress or

[12] Due to the "Zipfian" nature of the Internet traffic, some "elephant" flows may account for the majority of Internet traffic during an epoch.

[13] They may not become all 0's simultaneously due to the approxi-

egress counter values, a random tie-breaking is performed. We will show that such a simple algorithm produces surprisingly accurate estimation of flow matrix for medium to large flows.

The computation complexity of the algorithm in Figure 6 is $O((m+n-1)(\log m + \log n))$ because the binary searching operation (lines 3 and 4) dominates the complexity of each iteration and there are at most $m+n-1$ iterations. Thus the overall complexity to estimate the flow matrix is $O(b(m+n-1)(\log m + \log n))$.

Recall that an exact flow matrix can be used to indicate intrusions such as DDoS attacks in Section 1. Unfortunately our estimation algorithm only offers accurate estimation on the medium and large flow matrix elements, as shown in Section 6; some typical intrusions (e.g., DDoS attacks) consist of a large number of *small* flows. To make our algorithm still be able to provide valuable information of intrusions we can adapt the flow definition in our scheme correspondingly. For example, to detect DDoS attacks, we can use the destination IP address of a packet as its flow label. Then the traffic of a DDoS attack becomes a large flow going through the network instead of a large number of small ones.

Sometimes besides obtaining a flow matrix during a time interval we need to know the identity (flow labels) of these flows, which is not provided by our scheme and could be useful in some applications. One possible method is to generate these flow labels using other sampling or streaming algorithms [7]. Since this is a challenging separate effort which is orthogonal to the problem we are solving, we do not explore it further.

Finally, the traffic matrix can also be obtained by adding up the sizes of all the flows that we determine going from node i to node j using the above algorithm. This is in fact a fairly accurate estimation of traffic matrix, as shown in Section 6, since our algorithm tracks kangaroos and elephants very accurately, and they account for the majority of traffic. However, for the purpose of estimating traffic matrix alone, the bitmap scheme provides much better accuracy and is clearly a better choice, also shown in Section 6.

6. EVALUATION

An ideal evaluation of our traffic matrix and flow matrix estimation mechanisms would require packet-level traces collected simultaneously at hundreds of ingress and egress routers in an ISP network for a certain period of time. However, it is very expensive if not impossible to collect, ship, and store raw packet-level traces at a large number of high-speed links (OC-192). Instead, we use two data sets in our evaluation: synthetic traffic matrices generated from publicly available packet-level traces from NLANR [13] and actual traffic matrices from a tier-1 ISP. The ISP traffic matrix was produced in [25] and corresponds to traffic during a one-hour interval in a tier-1 network.

6.1 NLANR trace-driven experiments

We adopt two performance metrics: Root Mean Squared Error (RMSE) and Root Mean Squared Relative Error (RMSRE), which were proposed and used in [25] for the same purpose of evaluating the accuracy of estimated traffic matrix.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2}$$

$$RMSRE = \sqrt{\frac{1}{N_T}\sum_{\substack{i=1\\x_i>T}}^{N}(\frac{\hat{x}_i - x_i}{x_i})^2}$$

mation mentioned above.

The RMSE provides an overall measure of the absolute errors in the estimates, while RMSRE provides a relative measure. Note that the relative errors for small matrix elements are usually not very important for network engineering. We take only matrix elements greater than some threshold T in the computation of RMSRE (properly normalized). In the above equation, N_T refers to the number of matrix elements greater than T, i.e., $N_T = |\{x_i|x_i > T, i = 1, 2, \cdots, N\}|$.

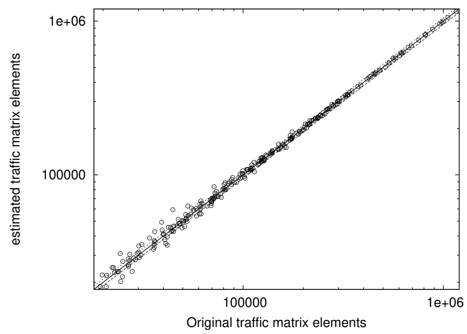
The set of traces we used consists of 16 publicly available packet header traces from NLANR. The number of flows in these traces varies from 170K to 320K and the number of packets varies from 1.8M to 3.5M. We piece together these traces to construct a synthetic scenario that *appears* as if these traces were collected simultaneously at all ingress nodes of a network. We set up the experimental scenario as follows: there are 16 ingress nodes and 16 egress nodes in the measurement domain. Each trace corresponds to the packet stream for one ingress node. The challenge in constructing this scenario lies in assigning the flows in the input stream at an ingress node to 16 different egress nodes such that the generated matrix will reflect some properties of real traffic matrices.

Recent work [1] shows that the Internet has "hot spot" behavior. A few OD pairs have very large traffic volume, while the majority of OD pairs have substantially less volume between them. Following the observed quantitative properties of real Internet traffic matrices [1], for each ingress node, we randomly divide the 16 ingress nodes into three categories: 2 nodes belonging to *large* category, 7 nodes belonging to *medium* category, and the rest 7 nodes belonging to *small* category. For each flow at an ingress node, we assign an egress node randomly, with nodes belonging to the large category twice as likely to be picked as medium category nodes which in turn are twice as likely to be picked as small category nodes. For simplicity, we configure the size of the bitmap and the counter array to fit the data set size without adopting the enhancement techniques (i.e., multipaging and sampling). Thus, we set the size of bitmap to 2,880K bits and the size of counter array to 320K counters which approximately occupy 2,880K bits of fast memory (SRAM).

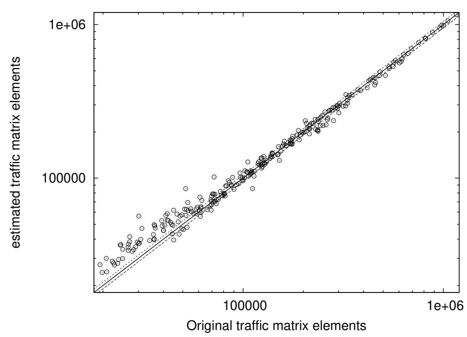
Figure 7 compares the estimated traffic matrix elements using the bitmap scheme and counter array scheme with the original traffic matrix elements. The solid diagonal lines denotes perfect estimation, while the dashed lines denote an estimation error of $\pm 5\%$. Clearly, the closer the points cluster around the diagonal, the more accurate the scheme is. We observe that both schemes are very accurate, and the bitmap scheme is more accurate than the counter array scheme.

Figure 8 shows the impact of varying T on RMSRE. We observe that both schemes produce very close estimates for the large and medium matrix elements. The traffic volume of the thresholded matrix elements decreases as the threshold increases, and the performance improves. For example, the RMSRE actually drops to below 0.05 for the top 70% of traffic for the counter array scheme. For the bitmap scheme, it drops even further to below 0.01 for the top 70% of traffic. In absolute terms, the RMSEs of the bitmap scheme and counter array scheme are equal to 4,136 packets and 11,918 packets, respectively, which are very small in comparison to the average traffic on a node. All of the above results confirm that the bitmap scheme achieves higher accuracy than the counter array scheme. The overall RMSRE of the bitmap scheme is below 6%, and that of the counter array scheme evolves from around 1% for large elements to 16% for the overall elements.

Note that our results reflect relative accuracy on a small time scale (one to several seconds for high speed routers), and they should not be directly compared with results reported in literature such as [20], which is on much larger time scales. Our schemes



(a) The bitmap based scheme



(b) The counter-array based scheme

Figure 7: Original vs. estimated traffic matrix elements. Both axes are on logscale

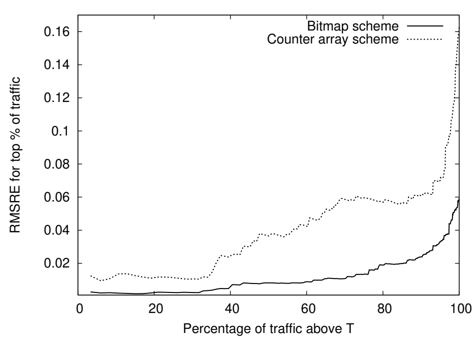


Figure 8: The RMSRE for various threshold T.

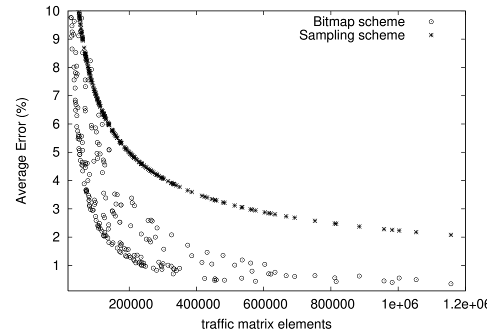


Figure 9: The average error of the bitmap scheme and the sampling based scheme.

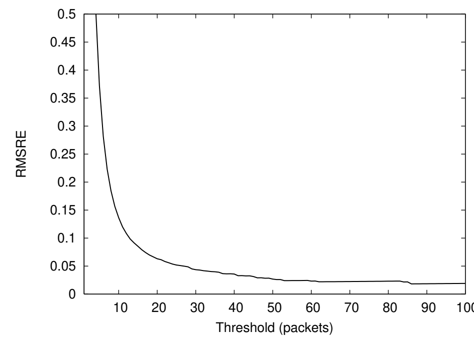


Figure 10: The RMSRE for various threshold T for flow matrix.

usually can achieve much higher relative accuracy on larger time scales (e.g., tens of minutes), as shown in Section 6.2.

We also compare the average (relative) error between our bitmap scheme and the existing sampling based schemes such as NetFlow. We adopt the similar method in [8] to infer the traffic matrix by collecting the packets in each ingress node with the typical NetFlow sampling rate of $\frac{1}{500}$ (which generates a similar amount of data per second as our bitmap scheme) and inferring the traffic matrix according to the egress nodes we assigned above for each sampled flows. Here, the variance of $\widehat{TM_{i,j}}$ is given by $\frac{TM_{i,j}(1-p)}{p}$ where p is the sampling rate $\frac{1}{500}$. Figure 9 plots the average error of each element of the traffic matrix in the trace-driven experiments for both our bitmap scheme and the sampling based scheme. We observe that our bitmap scheme achieves a consistently higher accuracy than the sampling based scheme.

Next, we evaluate the accuracy of flow matrix estimation. Note that our flow matrix estimation is counter-based and cannot distinguish the flows which are hashed to the same location with the same ingress node and egress node (we call this an *indistinguishable collision*). Our goal is to accurately estimate the medium and large flow matrix elements. We observe that the indistinguishable collisions happen rarely for medium and large flows. In our experiments, among the total 4 million flows with average size about 10 packets there are only 41 out of 71,345 medium and large flows ($>$ 10 packets) which suffer from this collision. Thus the impact of such indistinguishable collisions on the estimation accuracy is negligible. Figure 10 shows RMSREs for various threshold T. We observe a sharp downward trend in the value of RMSRE for increasing threshold values. When the threshold is equal to 10 packets, the error drops to below 15%. The accurate estimation of these flows is very important since, in this trace, flows of size 10 and above (71,345 of them) accounts for 87% of the total traffic.

6.2 Experiments based on ISP traffic matrices

We use a one-hour router-level traffic matrix from a tier-1 ISP network obtained in [25] to analytically evaluate the accuracy of the bitmap scheme. We assume that traffic volume between each pair

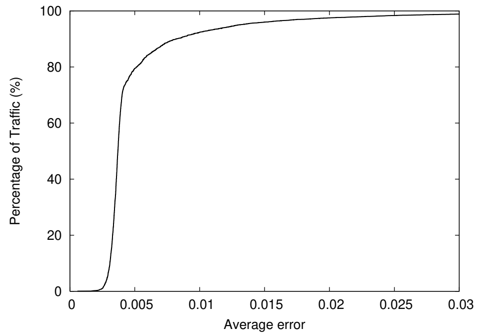


Figure 11: Cumulative distribution of traffic with certain average error.

of backbone routers is evenly distributed over the one hour time period[14]. Clearly an hour's traffic is too large (we assume a conservative average packet size of 200 bytes) to fit in a single bitmap, and therefore the aforementioned multipaging technique is used. Given a traffic matrix, we split the traffic on each ingress/egress node into multiple pages of 4Mbits (i.e., 512KB) with load factor 0.7 (the default load factor described in Section 4). Then, we compute the standard deviation for each pair of overlapped pages using Theorem 1. The sum of the standard deviation divided by the real matrix element value gives us the predicted error for the entire 1-hour interval. Figure 11 shows the cumulative distribution of traffic with the analytically predicted average error. We observe that our bitmap scheme provides very accurate results. Over 98% of the traffic has negligible average error (< 0.03) and the error for around 80% traffic is even below 0.005. Compared with the result in [26], our scheme improves the accuracy by more than an order of magnitude. For example, the error for around 80% traffic in [26], is about 20%. In addition, the average error across all traffic matrix elements in our estimation is around 0.5%, which is also more than an order of magnitude lower than that in [26] (i.e., 11.3%).

7. RELATED WORK

Previous work on estimating traffic matrices (e.g., [2, 21, 22, 25, 26, 12, 20, 14]) has mostly focused on statistically inferring them from SNMP link counts and/or the internal and external routing configurations (i.e., the routing matrix). The main advantage of these techniques is that little or no network instrumentation is needed since the information needed for inference is often readily available from the network management infrastructure (e.g., SNMP) already in place. However, since these techniques often need to make some assumptions on traffic model and/or network status, they usually cannot achieve very high accuracy (typically no better than 10% average error). Only the recent work by Soule et. al. [20] managed to reduce this error to 4%$\sim$5% using an active measurement technique (tuning routing matrix).

Feldmann et al. [8] proposed a semi-direct approach in which all ingress links collect flow-level statistics using *NetFlow*, then deduce the destination for each flow from the routing matrix. It is "semi-direct" in the sense it still infers information from the internal state (routing matrix) of the network. A direct approach called *trajectory sampling* is proposed by Duffield et al. [4]. Using a hashing technique similar to our consistent hashing[15], it ensures that all routers along the same OD path will sample the same set of packets. Data from all sampling points is collected and analyzed offline to construct a "snapshot" of the the flows of traffic through the network. The traffic matrix elements can be derived by summing all the traffic on the paths between the OD pairs. One slight drawback of this approach is that its accuracy is limited by the typically low sampling rate needed to make the sampling operation affordable.

[14]Our estimation accuracy will not be impacted significantly by the specific distribution of traffic during this one-hour interval, when multipaging is used.

[15]Our consistent hashing is an extension of the hashing semantics in [4].

8. CONCLUSION

The problem of estimating traffic matrices has received considerable attention recently. In this work, we attack this problem using a brand new approach: network data streaming. Our first main contribution is a novel data streaming algorithm that can produce traffic matrix estimation at least (depending on the amount of SRAM we use) an order of magnitude better than all existing approaches. We also establish principles and techniques for optimally combining this streaming method with sampling through rigorous analysis. In addition, we propose another data streaming algorithm that very accurately estimates *flow matrix*, a finer-grained characterization than traffic matrix. Both algorithms are designed to operate at very high link speeds (e.g., 40 Gbps) using only a small amount of SRAM (e.g., 512KB) and reasonable persistent storage. The accuracy of both algorithms is rigorously analyzed and verified through extensive experiments on synthetic and actual traffic/flow matrices.

Acknowledgements

We thank Dr. Yin Zhang for providing the third author with the traffic matrix obtained in their previous work. We also thank the anonymous reviewers for their constructive comments that help improve the quality and readability of the paper. The work of Zhao, Kumar and Xu was supported by NSF CAREER award ANI-0238315 and a grant from Georgia Tech Broadband Institute (GTBI). The authors Kumar, Wang, and Xu are listed in the alphabetical order.

9. REFERENCES

[1] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. Geographical and temporal characteristics of inter-pop flows: View from a single pop. *European Transactions on Telecommunications*, 2000.

[2] J. Cao, D. Davis, S. Vander Wiel, and B. Yu. Time-varing network tomography:router link data. *Journal of American Statistics Association*, pages 1063–1075, 2000.

[3] J. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, pages 143–154, 1979.

[4] N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE transaction of Networking*, pages 280–292, June 2001.

[5] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *Proc. of ACM/SIGCOMM IMW*, August 2002.

[6] N. Duffield, C. Lund, and M. Thorup. Estimating flow distribution from sampled flow statistics. In *Proc. of ACM SIGCOMM*, August 2003.

[7] C. Estan and G. Varghese. New directions in traffic measurement and a ccounting. In *Proc. of ACM SIGCOMM*, August 2002.

[8] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F.True. Deriving traffic demand for operational IP networks: Methodology and experience. In *Proc. of ACM SIGCOMM*, August 2000.
[9] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proc. of I.R.E*, pages 1098–1102, 1952.
[10] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. In *Proc. of IEEE INFOCOM*, March 2003.
[11] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *Proc. of ACM SIGMETRICS*, 2004.
[12] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation:existing techniques and new directions. In *Proc. of ACM SIGCOMM*, August 2002.
[13] http://pma.nlanr.net.
[14] A. Nucci, R. Cruz, N. Taft, and C. Diot. Design of IGP link weight changes for estimation of traffic matrices. In *Proc. of IEEE INFOCOM*, March 2004.
[15] A. Pasztor and D. Veitch. PC based precision timing without GPS. In *Proc. of ACM SIGMETRICS*, June 2002.
[16] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. In *Proc. of ACM SIGMETRICS*, June 2003.
[17] M. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient hardware hashing functions for high performance computers. *IEEE Transactions on Computers*, pages 1378–1381, 1997.
[18] http://www.ripe.net.
[19] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-based IP traceback. In *Proc. of ACM SIGCOMM*, August 2001.
[20] 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 *Proc. of ACM SIGMETRICS*, June 2004.
[21] C. Tebaldi and M. West. Bayesian inference on network traffic using link count data. *Journal of American Statistics Association*, pages 557–576, 1998.
[22] Y. Vardi. Internet tomography: estimating source-destination traffic intensities from link data. *Journal of American Statistics Association*, pages 365–377, 1996.
[23] K.Y. Whang, B.T. Vander-zanden, and H.M. Taylor. A linear-time probabilistic counting algorithm for database applications. *IEEE transaction of Database Systems*, pages 208–229, June 1990.
[24] J. Xu, M. Singhal, and Joanne Degroat. A novel cache architecture to support layer-four packet classification at memory access speeds. In *Proc. of IEEE INFOCOM*, March 2000.
[25] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale IP traffic matrices from link loads. In *Proc. of ACM SIGMETRICS*, June 2003.
[26] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An information-theoretic approach to traffic matrix estimation. In *Proc. of ACM SIGCOMM*, August 2003.

APPENDIX

.1 Preliminaries

Given a random variable X, we denote $X - E[X]$ as X^c. X^c is often referred to as a *centered random variable*. It can easily be verified that $E[X^c] = 0$ and $Var[X] = Var[X^c]$.

Next, we state without proof some lemmas and facts which will be used for proving Theorem 1. Here T is an arbitrary set of distinct packets hashed to a bitmap.

LEMMA 1. *(proved in [23])*

(I) $E[U_T] = be^{-t_T}$

(II) $Var(U_T) = be^{-t_T}(1 - (1 + t_T)e^{-t_T})$

(III) $E[D_T] = |T| + \dfrac{e^{t_T} - t_T - 1}{2}$

(IV) $Var(D_T) = b(e^{t_T} - t_T - 1)$

(V) $D_T^c \approx -e^{t_T} U_T^c$

Remark: We can see that from Lemma 1(III) that D_T is a biased estimator of $|T|$ with bias $\frac{e^{t_T} - t_T - 1}{2}$. However, this bias is very small compared to the parameter $|T|$, since $|T|$ is typically in millions and the t_T (load factor) we are working with is no more than 0.7, resulting in a bias no more than 0.16. We omit this bias from all following derivations since this omission results in negligible differences in the values of affected terms (confirmed by Monte-Carlo simulation), and the final result would be unnecessarily complicated without this omission. With this understanding, we will use equal signs instead of approximation signs when the only approximation involved is the omission of the bias.

Next, we prove a lemma that will be used extensively in the proof of Theorem 1.

LEMMA 2. *Let Y and Z be two sets of packets and $Y \bigcap Z = \emptyset$. Then we have (i) $E[U_Y^c U_Z^c] = 0$ and (ii) $E[U_{Y \bigcup Z}^c U_Y^c] \approx e^{-t_Z} Var[U_Y]$.*

PROOF. (i) Note that U_Y^c and U_Z^c are independent random variables since $Y \bigcap Z = \emptyset$. Therefore $E[U_Y^c U_Z^c] = E[U_Y^c]E[U_Z^c] = 0$.
(ii) Recall that B_T denotes the corresponding bitmap after "inserting" T into an empty array, and U_T is the number of 0's in the resulting array. Then U_T^c can be thought of as "extra" 0's in addition to the average (this "extra" can go negative). Then

$$\begin{aligned} E[U_{Y \bigcup Z}^c U_Y^c] &= \sum_{i=-\infty}^{\infty} E[U_{Y \bigcup Z}^c U_Y^c | U_Y^c = i] Pr[U_Y^c = i] \\ &= \sum_{i=-\infty}^{\infty} i E[U_{Y \bigcup Z}^c | U_Y^c = i] Pr[U_Y^c = i] \qquad (*) \end{aligned}$$

We would like to show that $E[U_{Y \bigcup Z}^c | U_Y^c = i] \approx ie^{-t_Z}$, denoted as (**). It suffices to prove that for any particular Y^* (a constant set) such that $U_{Y^*} - E[U_Y] = i$ and $Y^* \bigcap Z = \emptyset$, we have $E[U_{Y^* \bigcup Z}] = (i + be^{-t_Y})e^{-t_Z}$, denoted as (***). This is because, when (***) holds, we have

$$\begin{aligned} E[U_{Y \bigcup Z}^c | U_Y^c = i] &= E[U_{Y \bigcup Z} | U_Y^c = i] - E[U_{Y \bigcup Z}] \\ &= E[U_{Y^* \bigcup Z}] - E[U_{Y \bigcup Z}] \\ &\approx (i + be^{-t_Y})e^{-t_Z} - E[U_{Y \bigcup Z}] \end{aligned}$$

Since $E[U_{Y \bigcup Z}] = be^{-t_Y} e^{-t_Z}$, we have $E[U_{Y \bigcup Z}^c | U_Y^c = i] \approx ie^{-t_Z}$.

It remains to prove (***). Note that in the bitmap B_{Y^*} approximately $be^{-t_Y} + i$ entries (or $e^{-t_Y} + i/b$ percentage) are 0 since $U_{Y^*}^c = i$. Since in B_Z by definition U_Z entries (or U_Z/b percentage) are 0, in $B_{Y^* \bigcup Z}$ on the average $(e^{-t_Y} + i/b)(U_Z/b)$

percentage of entries is 0. In other words, $E[U_{Y^* \bigcup Z} | U_Z] \approx U_Z(e^{-t_Y} + i/b)$. Therefore,

$$E[U_{Y^* \bigcup Z}] = E[E[U_{Y^* \bigcup Z} | U_Z]] = E[U_Z(e^{-t_Y} + i/b)] = e^{-t_Z}(e^{-t_Y} + i/b)$$

which is exactly (***). Now applying (**) to (*) we have

$$E[U^c_{Y \bigcup Z} U^c_Y] = \sum_{i=-\infty}^{\infty} i^2 e^{-t_Z} Pr[U^c_Y = i] = e^{-t_Z} Var[U^c_Y] = e^{-t_Z} Var[Y]$$

□

Note however that $E[U^c_Y U^c_Z] = 0$ in general does not hold where Y and Z are not disjoint. The value of $E[U^c_Y U^c_Z]$ is characterized in the following lemma. We omit its proof since it is similar to that of lemma 2 (ii).

LEMMA 3. $E[U^c_Y U^c_Z] \approx e^{-t_{(Y-Z)} - t_{(Z-Y)}} Var[U_{Y \bigcap Z}]$

.2 Proof of Theorem 1

PROOF. To simplify the notations in the following proof, we use $R, S, N, \hat{N}$ to replace $T_i, T_j, TM_{i,j}$, and $\widehat{TM_{i,j}}$ respectively. Recall that our estimator becomes $\hat{N} = D_R + D_S - D_{R \cup S}$. We have

$$\begin{aligned} Var(\hat{N}) =& Var(\hat{N}^c) \\ =& E[(D^c_R + D^c_S - D^c_{R\cup S})^2] \\ =& E[(D^c_R)^2] + E[(D^c_S)^2] + E[(D^c_{R\cup S})^2] \\ &+ 2E[D^c_R D^c_S] - 2E[D^c_R D^c_{R\cup S}] \\ &- 2E[D^c_S D^c_{R\cup S}] \end{aligned}$$

These six terms are calculated as follows. By Lemma 1, we have

$$E[(D^c_R)^2] = Var(D_R) = b(e^{t_R} - t_R - 1) \quad (7)$$

$$E[(D^c_S)^2] = Var(D_S) = b(e^{t_S} - t_S - 1) \quad (8)$$

$$E[(D^c_{R\cup S})^2] = Var(D_{R\cup S}) = b(e^{t_{R\cup S}} - t_{R\cup S} - 1) \quad (9)$$

$$\begin{aligned} E[D^c_R D^c_S] &\approx E[e^{t_R} U^c_R e^{t_S} U^c_S] && \text{by Lemma 1} \\ &\approx e^{t_R + t_S - t_{R-S} - t_{S-R}} Var(U_{R\cap S}) && \text{by Lemma 3} \\ &= b(e^{t_{R\cap S}} - (1 + t_{R\cap S})) && (10) \end{aligned}$$

$$\begin{aligned} E[D^c_R D^c_{R\cup S}] &\approx E[e^{t_R} U^c_R e^{t_{R\cup S}} U^c_{R\cup S}] && \text{by Lemma 1} \\ &= e^{t_R + t_{R\cup S}} E[U^c_R U^c_{R\cup(S-R)}] \\ &\approx e^{t_R + t_{R\cup S}} e^{-t_{S-R}} Var[U_R] && \text{by Lemma 2} \\ &= b(e^{t_R} - (1 + t_R)) && (11) \end{aligned}$$

Similarly we have

$$E[D^c_S D^c_{R\cup S}] = b(e^{t_S} - (1 + t_S)) \quad (12)$$

Adding up these terms (Equation. 7-12), we get

$$Var(\hat{N}) = b(2e^{t_{R\cap S}} + e^{t_{R\cup S}} - e^{t_R} - e^{t_S} - t_{R\cap S} - 1)$$

□

.3 Proof of Theorem 2

PROOF. It is not hard to verify that our estimator $\hat{X} = \hat{N}/p$ is approximately unbiased, i.e., $E[\hat{X}] \approx X$, provided that we consider $\hat{N}$ as unbiased (discussed in Appendix .1). Therefore, we have

$$\begin{aligned} Var(\hat{X}) =& E[(\frac{\hat{N}}{p} - X)^2] = \frac{1}{p^2} E[(\hat{N} - pX)^2] \\ =& \frac{1}{p^2} E[(\hat{N} - N + N - pX)^2] \\ =& \frac{1}{p^2} E[(\hat{N} - N)^2] + \frac{1}{p^2} E[(N - pX)^2] \\ &+ \frac{2}{p^2} E[(\hat{N} - N)(N - pX)] \end{aligned}$$

We first prove that $E[(\hat{N} - N)(N - pX)] \approx 0$, i.e., $\hat{N} - N$ and $N - pX$ are approximately orthogonal[16]. To show this, we note that for any fixed value n, $\widehat{N(n)} - n$ and $n - pX$ are independent random variables because they represent errors from two independent measurements. Here $\widehat{N(n)}$ denotes the value of the estimator when the actual number is n. Therefore $E[(\widehat{N(n)} - n)(n - pX)] = E[\widehat{N(n)} - n]E[n - pX] = (E[\widehat{N(n)}] - n)(n - E[pX]) \approx 0$ since $\widehat{N(n)}$ is approximately unbiased (discussed in Appendix .1). Thus $E[(\hat{N} - N)(N - pX)] = E[E[(\hat{N} - N)(N - pX) | N = n]] \approx 0$. Therefore, we have $Var(\hat{X}) = \frac{1}{p^2} E[(\hat{N} - N)^2] + \frac{1}{p^2} E[(N - pX)^2]$. We claim that $E[(\hat{N} - N)^2] \approx E[(\hat{N} - pX)^2]$, denoted as (*). To prove this, note that with high probability, N does not fluctuate too far away its mean pX due to large deviation theory. Therefore we obtain $E[(\hat{N} - N)^2] \approx E[(\hat{N} - pX)^2]$ using mean value approximation. Finally, since N is a binomial random variable, we have $E[(N - pX)^2] = Var(N) = Xp(1-p)$, denoted as (**). Combining (*), (**), and Theorem 1, we obtain the variance of $\hat{X}$ as approximately

$$\frac{b}{p^2}\left((e^{\frac{T_p}{b} - \frac{X_p}{2b}} - e^{\frac{X_p}{2b}})^2 + e^{\frac{X_p}{b}} - \frac{Xp}{b} - 1\right) + \frac{X(1-p)}{p}$$

□

.4 H_3 hash function

Each hash function in H_3 class [3] is a linear transformation $B^T = QA^T$ that maps a w-bit binary string $A = a_1 a_2 \cdots a_w$ to an r-bit binary string $B = b_1 b_2 \cdots b_r$ as follows:

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1w} \\ q_{21} & q_{22} & \cdots & q_{2w} \\ \cdots & \cdots & \cdots & \cdots \\ q_{r1} & q_{r2} & \cdots & q_{rw} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_w \end{pmatrix}$$

Here a k-bit string is treated as a k-dimensional vector over GF(2) = {0,1} and T stands for transposition. Q is a $r \times w$ matrix defined over GF(2) and its value is fixed for each hash function in H_3. The multiplication and addition in GF(2) is boolean AND (denoted as $\circ$) and XOR (denoted as $\oplus$), respectively. Each bit of B is calculated as:

$b_i = (a_1 \circ q_{i1}) \oplus (a_2 \circ q_{i2}) \oplus \cdots \oplus (a_w \circ q_{iw}) \quad i = 1, 2, \cdots, r$

Since computation of each output bit goes through $\log_2 w$ stages of Boolean circuitry [24], and all output bits can be computed in paralle, it can finish well within 10ns.

[16]Note that they are dependent since N is a random variable.