A Distributed Approach to Measure IP Traffic Matrices*

Konstantina Papagiannaki§, Nina Taft‡, Anukool Lakhina†

§Intel Research
Cambridge, UK
dina.papagiannaki@intel.com

‡Intel Research
Berkeley, CA, USA
nina.taft@intel.com

†Computer Science Department,
Boston University, MA, USA
anukool@cs.bu.edu

ABSTRACT

The traffic matrix of a telecommunications network is an essential input for any kind of network design and capacity planning decision. In this paper we address a debate surrounding traffic matrix estimation, namely whether or not the costs of direct measurement are too prohibitive to be practical. We examine the feasibility of direct measurement by outlining the computation, communication and storage overheads, for traffic matrices defined at different granularity levels. We illustrate that today's technology, that necessitates a centralized solution, does indeed incur prohibitive costs. We explain what steps are necessary to move towards fully distributed solutions, that would drastically reduce many overheads. However, we illustrate that the basic distributed solution, in which flow monitors are on all the time, is excessive and unnecessary. By discovering and taking advantage of a key stability property underlying traffic matrices, we are able to propose a new scheme that is distributed and relies only on a limited use of flow measurement data. Our approach is simple, accurate and scalable. Furthermore, it significantly reduces the overheads above and beyond the basic distributed solution. Our results imply that direct measurement of traffic matrices should become feasible in the near future.

Categories and Subject Descriptors

C.2.3 [Computer-Communication Networks]: Network Operations - Network Monitoring

General Terms

Algorithms, Management, Measurement, Design

Keywords

Traffic matrix, Internet measurement, Distributed algorithm

---

1. INTRODUCTION

The traffic matrix (TM) of a telecommunications network measures the total amount of traffic entering the network from any ingress point and destined to any egress point. The knowledge captured in the traffic matrix constitutes an essential input for optimal network design, traffic engineering and capacity planning. Despite its importance, however, the traffic matrix for an IP network is a quantity that has remained elusive to capture via direct measurement. The reasons for this are multiple. First, the computation of the traffic matrix requires the collection of flow statistics across the entire edge of the network, which may not be supported by all the network elements. Second, these statistics need to be shipped to a central location for appropriate processing. The shipping costs coupled with the frequency with which such data would be shipped translate to communications overhead, while the processing cost at the central location translates to computational overhead. Lastly, given the granularity at which flow statistics are collected with today's technology on a router, the construction of the traffic matrix requires explicit information on the state of the routing protocols as well as the configuration of the network elements [5]. The storage overhead at the central location thus includes routing state and configuration information. It has been widely believed that these overheads would be so significant as to render computation of backbone traffic matrices, through measurement alone, not viable using today's flow monitors.

This assumption has been one of the main motivations behind recent research targeted toward estimation techniques that can infer the traffic matrix from readily available SNMP link counts [13, 14, 8, 11, 1, 6]. The SNMP link counts constitute only partial information, and thus basic inference methods are limited in how low they can drive the error rates. Hence these previous efforts have explored different avenues for extracting additional information from the network. Research efforts, such as [1, 14, 13, 8], usually postulate some underlying model for the TM elements and then use an optimization procedure to produce estimates that are consistent with the link counts. In [11] the authors propose changing the IGP link weights in order to obtain more information to reduce the uncertainty in the estimates. While this technique is powerful in collapsing errors, it requires carriers to alter their routing in order to obtain a traffic matrix. It is not clear that carriers are willing to do this. In [6] they recognize that some of the optimization approaches may not scale to networks with large numbers of nodes (such as traffic matrices at the link-to-link granularity level) and

hence they propose a method for partitioning the optimization problem into multiple subproblems. This improves the scalability but at the expense of some accuracy. Most of these studies have come from carriers whose interest lies in backbone traffic matrices at larger time scales for the purposes of improving network traffic engineering. We consider a similar context in this paper.

There has been some debate as to whether or not inference techniques are really needed. Some researchers believe that the communication, storage and processing overheads of direct measurement are prohibitive thus rendering it impractical. Other researchers believe that the traffic matrix problem is an implementation issue, and can be solved by advances in flow monitoring technology. In this paper, we address this debate directly. While many of the inference techniques perform quite well, monitoring capabilities on the network elements have made noticeable progress, and technologies for the collection of flow statistics have been made available on a wide variety of router platforms. Due to such advances, we believe it is time to revisit the issue of direct measurement of an IP traffic matrix.

If direct measurement can be made practical, then there are many reasons why it would be attractive. Direct measurement could lead to very small errors, and would remove the need for modeling and optimization procedures. Perhaps the most salient reason is that it has the potential to enable distributed solutions to traffic matrix estimation; all of today's inference techniques necessitate a centralized solution. We are aware of carriers that are currently evaluating the potential of enabling monitors such as Netflow on a widescale basis. Before they incur such dramatic costs, hidden overheads need to be exposed and understood. We hope this paper will contribute to such understanding.

Our contributions in this paper are as follows. First, we articulate what the overheads are for both centralized and distributed measurement solutions. Although this is a straightforward exercise, it is important to do because (1) it has never been spelled out before; (2) having this understanding enables one to know where improvements are possible; and (3) it allows us to evaluate the feasibility of direct measurement. Second, we identify the steps that are needed to move from a centralized solution (today's state of the art) to a distributed one (tomorrow's state of the art). This takes the form of specific recommendations to designers of flow statistics monitors, such as Netflow [2]. Recent advances in Netflow indicate that these monitors are moving in the right direction, however some additional steps are still needed. Third, we use Netflow data collected from the entire edge of the European Sprint IP backbone to compute traffic matrices at three levels of granularity, namely that of link-to-link, router-to-router, and PoP-to-PoP. The fanout is a vector describing the fraction of total traffic sourced at one node and destined to each of the other egress nodes. We find that node fanouts across all three traffic matrix aggregation levels are remarkably predictable across time. We propose a new scheme for the computation of the traffic matrix that relies on this observation. The key idea is that measurements are not needed frequently. We show that by updating the flow measurements once every few days, we can maintain fairly accurate traffic matrices at the hourly time scale. The improvement in computational overhead at the collection station is dramatic while the reduction in communications overhead ranges from 70-85% on average.

Finally, we also include a detailed discussion of errors. When a traffic matrix estimation method is validated against real data, it generates errors in both time and space. Some previous studies select one metric to summarize errors or decide to focus on the errors experienced by flows at a randomly selected time instant. In this work, we discuss some different views of the errors. An attractive feature of our scheme is that it allows the operator to tune a knob to control the error rate. The errors can be pushed very low by increasing the number of measurements taken. Inference techniques do not enable the operator to control this tradeoff between frequency of measurement and estimation accuracy. We will show that the number of measurements taken can be drastically reduced with reasonable sacrifice in terms of errors.

The implication of our work is that distributed direct measurement of traffic matrices is feasible as long as the recommendations we give are implemented. We believe that this is achievable as router manufacturers are poised to move along the needed path.

The paper is structured as follows. In Section 2 we present our data and define traffic matrices at three different granularities. In Section 3 we describe the state of the art in the computation of the traffic matrix. We also describe the advancements needed in today's flow monitors to move toward a distributed solution, and identify the overheads involved in direct measurement. Section 4 illustrates the predictability of fanouts and Section 5 describes our trigger-based algorithm for gathering only the needed measurements. Section 6 contains our performance metrics, the evaluation of our scheme, and a discussion of errors. The impact of our approach on the overheads is explained in Section 7. We conclude in Section 8.

2. TRAFFIC MATRIX DATA

In this work we analyze three weeks of traffic matrix data obtained using today's available technology, that of a centralized direct measurement approach. In this section we describe the collected data, the architecture of Sprint's European backbone network, as well as the three types of traffic matrices we study in this work.

2.1 The backbone network

Sprint is a Tier-1 provider whose European backbone IP network comprises 13 Points of Presence (PoPs), one for each major European city. Typically the number of routers in each PoP ranges from 5 to 10. The routers are organized in a hierarchy as depicted in Fig. 1. Customers connect to the network by being directly attached to gateway (gw) routers. Backbone (bb) routers aggregate the traffic of multiple gateway routers and forward it to the core of the network. The backbone routers are used for connecting peers to the backbone and also to inter-connect the PoPs.

In order to obtain a traffic matrix by direct measurement, we need to examine all the incoming packets to the backbone. We therefore enabled Netflow on all incoming peering links and all the links going from gateway routers to backbone routers. This latter set of links captures nearly all customer traffic. It only misses traffic that enters and leaves the network at the same gateway router[1].

---

[1]This implies that the only elements that may be impacted from this configuration choice are those that feature the
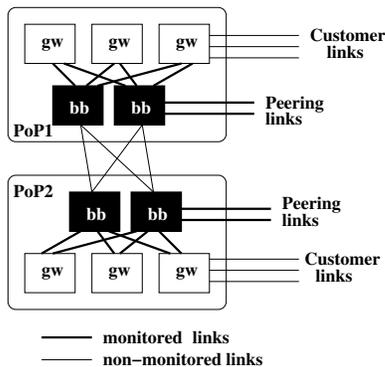
Figure 1: Setup for Netflow data collection.

We used Netflow v8 that is usually referred to as "sampled aggregated Netflow" [2]. Each record for Netflow v8 is 40 bytes long, and features a flow identifier, its source and destination network prefix, its associated load in bytes and packets, as well as its starting and ending time. Rather than examine every packet, Netflow v8 employs periodic sampling in which one sample is collected every 250th packet. Traffic statistics are not computed over each packet but based on this subset of the packets.

Each router ships the collected Netflow statistics to a configurable node (the collection station) every 5 minutes. We instrumented our European backbone with a single collection station for all 27 routers in this backbone. We used 5 minutes as our reporting interval since it coincides with the default SNMP reporting time interval. The collection station stores all flow statistics from *all* backbone routers in the IP network. We collected 3 weeks of data during the summer of 2003.

2.2 Traffic Matrix Granularities

The elements of a traffic matrix are origin-destination (OD) flows where the definition of the origin and destination object (i.e., node) can be selected as per the needs of the application using the traffic matrix. The granularity of a traffic matrix is determined by the choice of definition for the source or destination "object". The most typical objects are links, nodes and PoPs. In this work we consider traffic matrices at the granularity of "link to link", "router to router", and "PoP to PoP". For a taxonomy of IP traffic matrices and their potential applications in traffic engineering and network planning please refer to [7].

In addition to selecting the flow granularity of a traffic matrix, a network operator also needs to specify its time granularity. As mentioned above, the Netflow statistics are collected every five minutes. When we build traffic matrices from our data we average over one hour time intervals. We thus have one traffic matrix for each hour for a three week period. When we estimate traffic matrices using our proposed scheme, we generate estimates for one hour averages. We choose to focus on one hour since most traffic engineering applications are targeted toward longer time scales. One hour is actually small for such applications but we believe this is a useful time scale because it allows us to observe diurnal patterns [9] and busy periods that may last for a same source and destination node.

small number of hours. Notice that according to the above problem definition our scheme is designed not to address problem areas such as anomaly detection, that may require measurements at smaller time scales.

To the best of our knowledge, this is the first work that analyzes an IP traffic matrix computed from flow statistics collected across the *entire edge* of the network for a *multi-week* period of time. It is also the first study that examines the performance of a TM estimation scheme across three levels of granularity in a single work.

3. STATE OF THE ART: TODAY AND TOMORROW

There are basically three steps to obtain a traffic matrix from measurements. The first is to gather information about the traffic *source* by collecting measurements using Netflow, or a similar monitor. Packets are observed and statistics are stored at the granularity of flows. The second step is to identify the *destination* for each flow. The third step is to assemble all the information at the right granularity level (link, router or PoP) in a way that is consistent with the network topology.

In this section we provide a generic algorithm for doing steps 2 and 3. The Netflow data, gathered for step 1, serves as input to this algorithm. We describe the state of the art today for implementing such an algorithm. This is based on Netflow v8 and essentially requires a (semi-)centralized approach[2]. Cisco's most recent release of Netflow, v9, makes initial steps toward enabling the TM to be computed using a distributed approach. However we will see that this version does not go far enough to enable a truly distributed approach. We quantify the storage and communication overheads for both centralized and distributed approaches.

3.1 Identifying the egress node

A traffic matrix is typically computed for a single domain or Autonomous System (AS). As described above, Netflow statistics collected at each router are computed at the granularity level of source and destination network prefixes. These source and destination prefixes will often reside outside the AS whose traffic matrix is computed. Thus the source and destination of each packet need to be mapped onto the entry node and exit node within the given AS. Identifying the entry node is simple, as it is defined as the link or node where a packet enters a given domain (i.e., the place where Netflow sees the packet). The exit point for a particular source/destination network prefix flow will depend both on its entry point as well as its actual destination. To identify this exit point one needs to obtain a view of the forwarding table of the router that recorded the flow. Consolidation of intra- and inter-domain routing (from the vantage point of the router), as well as topological information can resolve each prefix flow into its egress node inside the network.

The task at hand is to accurately map the destination network prefix in each flow record to (i) a backbone-gateway egress link, (ii) an egress backbone router, and (iii) its egress PoP. To do this we need the BGP routing table from each

---

[2]Throughout the remainder of the paper we will use the term *centralized* to describe fully centralized or semi-centralized approaches. The latter would employ multiple collection stations, say one in each PoP, where a subset of the network routers will report their flow statistics.

router (typically we collect those from a route-reflector inside a PoP), the ISIS/OSPF link weights, and the router-level topology [5].

If a network prefix is advertized through BGP, then the BGP table specifies the last router within the AS that needs to forward traffic for this destination prefix, usually referred to as "BGP next hop". However, often the BGP next hop corresponds to the IP address of the first router outside the AS. In this case, configuration files can be used to map this address to a router interface within our network.

This identified router interface will very likely correspond to the interface of a gateway router. At this point, we can identify the egress PoP but we need additional information to be able to map this router interface (for the given destination prefix) to a particular egress backbone router, and backbone-gateway link. Here we use the router topology, along with the associated link weights, to compute the shortest paths across the network. Using these paths we can find the egress backbone router(s) and backbone-gateway link(s) that will forward traffic toward the previously identified gateway router. The number of such routers and links may be more than one, if the PoP topology is densely meshed and equal-cost multipath is enabled by the provider. In that case, we apportion the total flow equally toward each one of the routers or links selected using the process mentioned above.

3.2 Computing the traffic flow

A generic algorithm for the computation of the traffic matrix of an IP network can be summarized as in Fig. 2. At the heart of this algorithm is a routine called $find\_egress\_node(f)$ that returns the egress node at the desired level of granularity (link, node or PoP) according to the method described above. There are four nested loops in this algorithm, one for each time interval n, one for each router r, one for each link l and one for each flow f. The $find\_egress\_node(f)$ routine operates at the level of a flow because that is the form of the Netflow input. After the egress node is identified, the flows are aggregated so that the algorithm yields a traffic matrix at each of the granularity levels. In this pseudocode, $L(r)$ denotes the number of links at router r, and $F(l)$ denotes the number of flows on link l.

We note that a variant of this algorithm was originally proposed in [5]. We include this here not as a contribution, but merely in order to facilitate the ensuing discussion. This algorithm statement makes it easy to see how the overheads are computed, to identify all the additional routing/configuration data needed, and to clarify what the change from a centralized to a distributed approach implies.

A Centralized Approach. Because Netflow today does not implement a procedure such as $find\_egress\_node(f)$, all of the flow data needs to be shipped by each router to a specific collection station that can carry out the above algorithm. Thus today's state of the art essentially mandates a centralized solution. The collection station needs to have explicit information about each PoP's BGP routing table, and the ISIS weights in effect at each time interval n. In addition, it needs to have an accurate view of the network topology, in terms of the configuration of each router inside the network. For our implementation of a centralized solution, we downloaded router configuration files once a week, and BGP routing tables once a day from each PoP inside the network. Due to the fact that router configuration files do

Algorithm : COMPUTETM($data, T, R, L, F$)

for $n \leftarrow 1$ **to** T
  $ISIS = isis(n); \% the\ same\ topology\ network - wide$
  $configuration = \cup_{r=1}^{R} configuration\ file(r, n);$
for $r \leftarrow 1$ **to** R
   $routingtable = BGProutingtable(RR(r), n);$
   $\% BGP\ routing\ table\ of\ the\ route\ reflector$
   $in\ r's\ PoP.$
   for $l \leftarrow 1$ **to** $L(r)$
     for $f \leftarrow 1$ **to** $F(l)$
       $EN(f) = find\_egress\_node(f, routingtable,$
       $configuration, ISIS);$
       $TM(l, EN(f)) = TM(l, EN(f)) + data(f, t);$
return (TM)

Figure 2: Pseudocode for the computation of the traffic matrix.

not change on a daily basis, and that routing table changes occurring during a single day rarely affect large amounts of traffic [10], we found these choices reasonable. Certainly there are inaccuracies incurred by not having perfectly up to date routing information. However obtaining more frequent updates of this information greatly increases the communication overhead. (The only way to completely avoid these inaccuracies is to use a distributed approach as described below.)

Toward Distributed Approaches. A process similar to $find\_egress\_node(f)$ is already performed by the router itself before switching the packet to its destination. Therefore a truly distributed approach would be one in which each router saved the information on the egress point of each network prefix while performing the lookup in its routing tables. Since one router constitutes one source that sends potentially to all other routers in the network, saving traffic statistics on the amount of traffic destined to each egress point is equivalent to the router computing one row of the traffic matrix. With this approach the only data that needs to be shipped to a collection station is the TM row itself.

In order to do this, the router would need to change the flow record to include fields such as egress link, egress router, and/or egress PoP. If flow records were kept at the level of links or routers rather than prefixes, this would dramatically reduce the on-router storage. The communications overhead is also greatly scaled back since sending one row, of even a link-to-link traffic matrix, is far smaller than shipping individual prefix-level flow records. Furthermore, the computational overhead at the collection station has now been reduced to simply that of assembling the rows without any egress node identification activity.

Recent advances in the area of flow monitoring have led to new definitions for flow records that do incorporate explicit routing information defining flows such that the destination field captures the BGP next hop address. Such a change can be found in Netflow v9 [3] which thus constitutes a significant movement toward the distributed solution described above. This improvement is not yet sufficient though for the following reason. When a particular 5-tuple flow is mapped onto a "BGP-next-hop" flow, there is always the risk that the destination network prefix may not be advertized through BGP. For ease of implementation Netflow v9 addressed this issue by using "0.0.0.0" as the BGP next hop.

Such a design choice implies that all the traffic that may be going to internal customers is missed, if these customers are not advertized through BGP. For ISPs with a large number of customers, this may translate to many elements of the traffic matrix being altogether missed or inaccurately estimated, when a particular "unknown" flow would actually map to an existing flow in the cache.

The feasibility of direct measurement approaches is dependent upon the ability to implement a routine such as *find_egress_node(f)* at a router. We distinguish two factors regarding the implementation of this routine. First, we point out that the information needed is already available in today's routers in the Routing Information Base (RIB). The RIB contains (i) the mapping between each destination prefix and its BGP next hop (as dictated through BGP), (ii) the mapping between the BGP next hop and its egress node (as identified through the intra-domain protocol in use), and (iii) the mapping between the egress node and the appropriate local interface that should be used to reach that node. The second factor regarding the implementation is the need to gain efficient access to this information. This could require changes in the software architecture and is the main challenge to implementation. Our goal in this paper is not to spell out a complete implementation solution, but rather to identify what is needed to achieve a measurement based approach to populate traffic matrices and avoid inference-based techniques.

In summary, we thus make two recommendations to developers of Netflow type systems on routers. First, a mechanism to resolve destination prefixes to their egress link or router (such as the *find_egress_node* scheme) is needed on the router itself. Second, the flow record definitions need to be adjusted to include fields such as egress link or router. Because router manufacturers have already accommodated new flow definitions (e.g., the BGP next hop described above), this indicates that they are heading in the right direction, and we are hopeful that the proposed recommendations will eventually be undertaken. The position paper [12] advocates a similar approach. Their suggestion of using class-based counters is similar to ours when a class is defined to be either a link, a router or a PoP. For the remainder of the paper we will assume these steps will be taken in the future state of the art, and hence our discussion of overheads for distributed solutions is in terms of this vision.

3.3 Storage, Communications and Computational Overheads

We now describe the overheads involved in computing a traffic matrix at each of the granularity levels. The storage overhead per router refers to the amount of flow statistics stored at the router. These elements are updated and shipped to the collection station on an ongoing basis. The communications overhead refers to the total amount of information that needs to be transmitted through the network domain toward the collection station. This includes the inputs from all domain routers. The computational overhead we show is that of the activity at the collection station.

First we consider the centralized approach that is available today. Because the basic granularity of the collected and processed flow statistics is that of a network prefix, we have labelled the centralized solution as prefix-to-prefix in the first row of Table 1. Table 1 lists the associated overheads for the centralized solution for *one* time interval, that is the derivation of *one* traffic matrix, regardless of its actual granularity. Let L denote the total number of links, R the number of routers, and P the number of PoPs in the network. The average number of links per router is thus L/R. Based on the actual configuration parameters of the European network under study, and the current state of the art, the collection station had to perform 5.5 million egress node identifications (last column). This required 13 BGP tables, since each PoP has a different vantage point, and 1 ISIS routing table residing at the collection station.

The next three rows of this table correspond to the overheads for a distributed solution for cases where the end goal is a TM at a specific granularity level. The notation 'p2p' refers to a PoP-to-PoP matrix, 'r2r' indicates the router-to-router, and similarly for 'l2l'. With fully distributed solutions, the collection station merely needs to assemble the rows it receives as input and build the matrix. Letting F denote the total number of flows, then clearly in any typical network, we will have $P < R < L << F$. As an example, in a typical large Tier-1 backbone for a geographic region the size of the USA, P is on the order of tens, R is on the order of hundreds, L is on the order of a few thousands, and F is on the order of millions or even billions.

The advantages of moving from a centralized approach to a distributed one are clear: (1) the router storage overhead is reduced from $O(F)$ to $O(R)$ or $O(L)$ where R and L can be many orders of magnitude smaller than F; and (2) the communications overhead is reduced by two or three orders of magnitude (depending upon the target TM). In addition, recall that a problem with the centralized approach is that the routing table information at the collection station will regularly become out of date. Distributed solutions do not suffer from this problem since the information on the routers is essentially always up to date (as fast as the protocol can perform updates).

The amount of storage and communications overhead partly depends upon what the ISP wants to collect. For example, if an ISP is sure it only ever wants to collect the r2r matrix, then each router should compute one row of this matrix and the communications overhead is limited to R^2. If however a carrier prefers to leave open the flexibility of looking at a TM at any of the granularity levels, then they should seek the l2l matrix, which can be aggregated into a r2r and p2p by checking which link belongs to which router, which router belongs to which PoP, and so on. In this case the communications overhead is L^2.

There is one other very critical piece to the overhead issue, and that is the frequency with which one wants to collect a traffic matrix. Suppose carriers decide they want the traffic matrix to be updated K times within each day. Then all of the numbers in Table 1 would be multiplied by K for each day. In the case we focus on, these overheads would be incurred once every hour.

Clearly one needed step for direct measurement of a traffic matrix to become practical is for flow monitoring at routers to increase their functionality so as to compute rows of traffic matrices, thereby enabling a distributed approach. However we believe that this is still not enough. A communications overhead of L^2 or R^2 records incurred every hour may still be regarded as excessive. In the next section, we show that there indeed exist more *efficient* ways to obtain accurate estimates of TMs on an hourly basis.

Scheme	Storage	Communications	Computational
approach	(router)	(network)	(collection station)
SrcPrefix2DstPrefix	$L/R \cdot F(l)$	$L \cdot F(l)$	5,5M lookups, aggregation to the required granularity
centralized	$3 \cdot 67,000$	$81 \cdot 67,000 = 5,5M$	(additional storage: 13 BGP routing tables, 1 ISIS routing table, topology)
l2l row	$L/R \cdot L$	L^2	Matrix Assembly
distributed	$(3 \cdot 81 = 243)$	$(81 \cdot 81 = 6561)$	
r2r row	R	R^2	Matrix Assembly
distributed	(27)	$(27 \cdot 27 = 729)$	
p2p row	P	$R \cdot P$	Matrix Assembly
distributed	(13)	$(27 \cdot 13 = 351)$	

Table 1: Overheads (in number of records) for computing the traffic matrix in *one* time interval.

4. TOWARD A MORE SCALABLE APPROACH

4.1 Initial Observations

In examining our collected traffic matrices, we found that the elements of a traffic matrix typically span a few orders of magnitude. In Fig. 3 we provide the empirical CDF of origin-destination throughputs for the first week in the data, one at each level of granularity. The flow ranges from below 1 Kbps to roughly 100 Mbps at each level of granularity. Flows near 1 Kbps (the leftmost point on the x-axis of the figure) can essentially be viewed as near zero valued elements. In p2p matrices, 15% of the matrix elements are near zero; for r2r matrices the number is roughly 32% and for l2l the percentage is over 60%. This indicates that these matrices are sparse and that the sparsity increases at smaller levels of granularity.
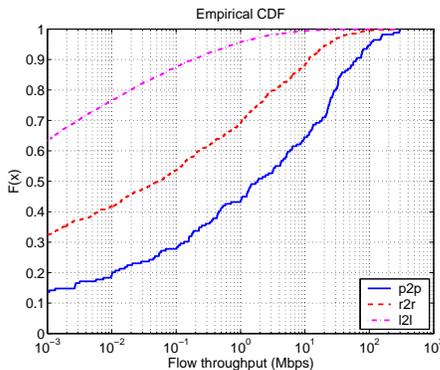

Figure 3: Empirical cumulative density function for p2p, r2r and l2l flows.

In addition, a large fraction of the OD flows experience throughput values below 1 Mbps, which could be considered as *tiny* in a Tier-1 network. There is a problem with these tiny flows due to a limitation stemming from the way our measurements are collected. The systematic sampling as employed by Netflow v8 is likely to provide us with inaccurate flow measurements when flows are tiny in magnitude. Given that we compute flow statistics on packet samples (every 250th packet), it is likely to be the case that tiny flows are undersampled or missed altogether. Since these

tiny flow throughputs are likely to be distorted due to infrequent sampling, there is little confidence in their observed properties.

The impact of sampling on the collected flow measurements is outside the scope of this work (for the issues involved refer to [4]). Nevertheless, in order to avoid possible inaccuracies that come from working with improperly sampled flows, our scheme is tuned to focus on the relevant flows. We define *relevant* flows as those with average weekly throughput greater than 1 Mbps. These relevant flows still include flows of varying sizes that can be considered small, medium or large. Our intention is to estimate *all* flows but with the goal of achieving low errors on the relevant flows. Many have argued that what carriers care about is estimating the larger elements well, because it is those elements that capture the majority of the traffic [5, 13, 11]. We concur with this statement. With this definition of relevant flows, we capture 95% or more of the total traffic load. Table 2 shows the number of OD pairs categorized as "relevant" and the total captured load for each of our three types of traffic matrices.

Granularity	# relevant	captured load
p2p	85	98%
r2r	216	95%
l2l	470	96%

Table 2: Relevant flows for l2l, r2r, and p2p.

4.2 The notion of a node fanout

First we establish our notation. We denote each element in the traffic matrix as $X(i,j,n)$ for the amount of traffic flowing from node i to node j at time interval n. Time is discretized into 1 hour intervals; we let T denote the total number of hours in our measurements and thus $1 \leq n \leq T$. The total number of nodes is given by M so we have $1 \leq i,j \leq M$. Each row of a traffic matrix is a description of how all the traffic from one source node is distributed among all other nodes. Let $f(i,j,n)$ denote the fraction of node i's traffic destined to egress node j at time interval n. This is given by,

$$f(i,j,n) = \frac{X(i,j,n)}{\sum_{k=1}^{M} X(i,k,n)}, \ 1 \leq n \leq T \qquad (1)$$

where $0 \leq f(i,j,n) \leq 1$. We define the vector $\vec{f}(i,n)$ to be the *node fanout* as it captures how node i's traffic is

apportioned across all the egress nodes. The elements of this vector sum to one, i.e., $\sum_{j=1}^{M} f(i,j,n) = 1, \forall i, n$.

Note that the denominator in Eq. 1 corresponds to all the traffic that node i emits at time n. If the node corresponds to a link, then this traffic load is available from SNMP counters. If the node is a router, then this load is computable by summing the SNMP counts from all ingress links to the router (in our case all links from gateway routers and peers that attach to this particular backbone router). Letting $Y_i(n)$ denote this load at time n, we can rewrite equation 1 as

$$X(i,j,n) = f(i,j,n) \cdot Y_i(n) \tag{2}$$

Representing traffic matrix elements as a product of a fanout and SNMP data has been used before [8, 13]. Now, for the first time, we examine temporal properties of these fanouts.

4.3   Temporal Properties of Fanouts

By examining $f(i,j,n)$ for a fixed i and j, and letting n vary, we can see how the fanout from node i to a particular egress point j varies over time. In Fig. 4 we show the temporal behavior of such fanout elements for three example flows, one at each level of granularity. The fanout for the p2p flow remains almost constant for the entire week. The fanouts for the r2r and l2l example flows exhibit diurnal patterns that repeat themselves throughout the week (no significant difference is observed in the weekend). This implies that these fanout elements are very predictable. The constant ones can be predicted by simply measuring in a single time interval. Those with diurnal patterns are predictable once you know the daily cycle.
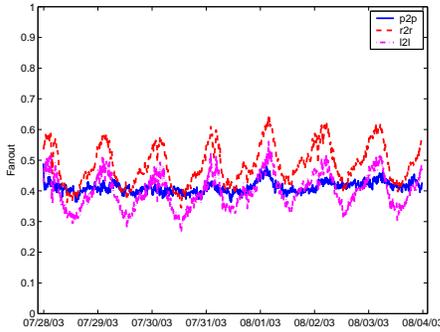

Figure 4: Example fanouts at the three levels of granularity.

We looked at many OD pairs and observed the same behavior. In order to verify the generality of this observation more thoroughly we did the following. First, we computed the Fast Fourier Transform (FFT) for the flow fanouts, during the period of July 28th, 2003 to August 4th, 2003, at all levels of granularity. We found that the fanouts of 98% of the p2p flows, 95% of the r2r flows, and 96% of the l2l flows exhibit a strong periodicity at the cycle of 24 hours. This implies that fanouts are predictable across days, i.e. the fanout for a flow at 1pm on day2 can be predicted based on the same hour on day1. Second, to check the stability of fanouts across days we computed the coefficient of variation for each hour for each fanout (and for each OD flow) across
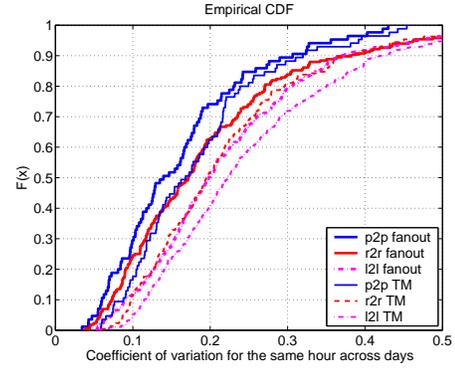

Figure 5: Variability of fanouts vs. traffic matrix.

all three levels of granularity. In other words, we computed the coefficient of variation for each fanout over all days for the 1-2pm slot, and similarly for all other slots. This metric allows us to assess the variability of fanouts (and OD flows) within the same hour across days. We found that typically this behavior is consistent across hours; all hours produce similar results. We thus summarize our results using the average coefficient of variation for each fanout (and for each OD flow) as the average of the 24 measurements we have (one for each hour). Our results are plotted in Fig. 5. We notice that fanouts tend to show very limited variations around their mean value for the same hour across days.

The predictability of fanouts is an interesting finding and should not be confused with the predictability of a traffic matrix. One could postulate that the fanouts are predictable because the TM is. Recall that the TM is the product of the fanout and the SNMP link counts, hence both of these components would need to be stable for the TM to be so as well. In Fig. 5 we have also included results on the coefficient of variation of the traffic of OD flows across days for the same hour (similar to the fanouts described above). Since all three curves lie below the curves for the fanouts, this implies that the traffic matrix is more variable and less predictable than the fanouts. We thus focus on the fanouts.
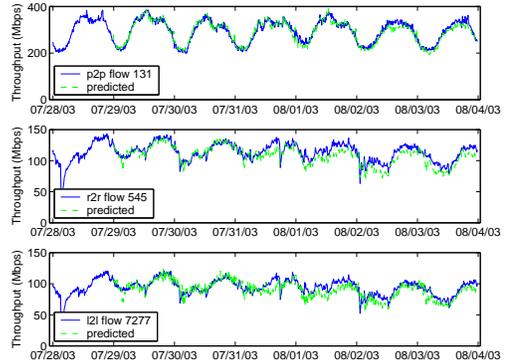

Figure 6: Estimated throughput based on the fanouts measured on July 28th, 2003.

This implies the following approach to TM estimation

could be viable. The idea is to use fanouts from day1 to estimate the behavior of OD flows for the remainder of the week, where the TM is computed from the fanouts and SNMP data according to Eq. 2. That is, we use updated SNMP counts every hour[3] to generate the matrix, but do not use updated fanouts. In Fig. 6, we show the estimates obtained using the fanouts and the actual traffic measured through Netflow for a particular OD flow at each level of granularity. Our results show that the estimated throughput is surprisingly close to what was measured through Netflow. Even small fluctuations observed in the p2p flow during the fourth day of the measurements were captured with high accuracy. The idea of using one day to predict 7 days is merely an example used here to illustrate this idea of using fanout stability for TM prediction. In later sections we will examine the issue of how long a given fanout can be used for prediction before becoming stale.

5. PRACTICAL MEASUREMENT OF AN IP TRAFFIC MATRIX

We now develop a new method for obtaining traffic matrices based on our two key findings so far. First, we assume that flow monitors will adopt our suggestions and routers will have the ability to directly compute TM rows. We can thus focus on distributed solutions. Second, we rely on the observation that fanouts are both quite stable, and also more stable than the traffic matrix itself. Rather than promote full direct measurement of a TM, we promote an approach that relies on partial flow monitoring coupled with the usual SNMP data.

The method contains two basic elements, an estimation part and an update part. For the estimation part, we use 24 hours of Netflow type measurements to compute a baseline for fanouts. Because of the presence of strong diurnal patterns, the baseline for each node contains 24 vectors, one for each hour of the day. The traffic matrix at time n can thus be computed using a small modification to Eq. 2 as follows:

$$X(i, j, n) = f_{bl}(i, j, n') \cdot Y_i(n), \ n' = (n\%24) + 1 \quad (3)$$

where % corresponds to the modulo operator, and $f_{bl}(i, j, n')$ denotes the baseline fanout. Assuming that the hour of a fanout is indexed by $1, \ldots, 24$, then the traffic matrix at time n is computed using the same hour n' from the baseline day.

The idea is to initially measure the fanouts exactly, but then to update them only on an as needed basis. The SNMP data is available every 5 minutes, so when estimates are made they are a combination of recent SNMP data and fanouts that may be many hours or days old. Since each router computes its own fanout, we want each router to be responsible for updating its own fanout, especially given that some routers may need their fanouts updated at different times than others. Our goal is to develop a method that allows this to happen, thus maintaining a purely distributed approach.

Because an IP network is a highly dynamic environment, clearly changes in fanouts are going to occur over time. We thus need a scheme that monitors for change and triggers a recomputation of the baseline when the fanouts have substantially diverted from the previously calculated ones. Re-

---

[3]We simply average the 5 minute counts over 1 hour periods for our purposes.

computing the baseline means here that the flow monitor is enabled for another 24 hours.

5.1 A trigger-based baseline update scheme

We use a three-step procedure to determine when updates are needed for the fanout baseline.

1. Compute the fanout *baseline*

2. Check for diversion from baseline

   (a) Pick 1 hour randomly within the next H hours
   (b) recompute the fanout only for that hour
   (c) measure *diversion* of current hour from baseline for relevant flows

3. **If** diversion $> \delta$, **then** recompute the baseline for all 24 hours; **else** return to Step 2.

The first time we compute the baseline in Step 1 is an initialization step when all ingress links, routers and PoPs compute the total amount of traffic they send to every other egress link, router or PoP inside the network for each 1 hour interval for an entire day. We denote by $\vec{f}_{bl}(i, n)$, $1 \leq n \leq 24$ the fanout vector that each node computes for itself, according to Eq. 1.

There are two important design choices in any change detection scheme: (i) How frequently should we check for changes in the fanouts? (ii) How large should the change in fanout be so as to trigger re-computation of the baseline? The frequency with which we check is controlled by parameter H. Step 2a above means that at a time slot n, a node randomly selects one hour h, where $1 \leq h \leq H$. The node computes its new fanout vector at time $n + h$ (i.e., $\vec{f}(i, n + h)$) *just for that hour*. We refer to hour $n + h$ as the *checking hour*. We compute the difference between the new fanout and the corresponding hour in the baseline $h_{bl} = (n + h)\%24 + 1$. In other words, we compare 3pm on the new day to 3pm in the baseline. We call the measured "change in fanout" either *diversion from baseline* or merely *diversion* for short. This approach ensures that the baseline is checked at least once every H hours.

We measure diversion in terms of relative change as in Eq. 4, so that even flows with small fanouts values (that may nonetheless correspond to high throughput flows) can trigger re-estimation of the baseline.

$$D(i, j) = |\frac{\hat{f}(i, j, n + h) - f_{bl}(i, j, h_{bl})}{f_{bl}(i, j, h_{bl})}|. \quad (4)$$

The condition for triggering a recomputation of the entire baseline (all 24 hours) is for the difference to exceed a threshold, namely if $D(i, j) > \delta$. We only check for fanout diversion among the *relevant* flows (those larger than 1 Mbps). Note that we check for diversion amongst all relevant flows, but we only require *one* OD flow to exceed δ in order to trigger a baseline update. The reasons for focusing on relevant flows are the following. First, the behavior of relevant flows is more interesting from the network operators' point of view because it affects traffic engineering applications. Second, as previously discussed, the irrelevant flows may not be reliable due to the sampling mechanism used. Third, because these traffic matrices are sparse, we reduce the amount of checking needed by limiting our checking to only relevant flows.

Recall that by limiting ourselves to only the relevant flows, we are still checking for fanout diversion among 95-99% of the total traffic load. Moreover, even though we only check for fanout diversion amongst relevant flows, when diversion is detected, our scheme re-estimates the *entire* fanout vector for that node. Therefore, when the baseline is recomputed we have an accurate picture of *all* fanouts sourcing at a particular node.

A subtle point to be taken into account is that given the router is only aware of the TM row during a single randomly selected hour, it is not capable of correctly identifying the *relevant* flows itself. A flow should be observed for longer than one day to reveal its long-term average throughput. Therefore identification of these flows will have to be performed by the collection station. One possible way is for the collection station to identify the relevant flows upon the assembly of the TM and notify each router accordingly.

Throughout the 3 week period of measurements at our disposal the set of relevant flows did not change. Nevertheless, there will be cases when flows change from *irrelevant* to *relevant*, or vice versa. Consequently, in an operational environment this list of relevant flows would need to be re-evaluated every so often. Given what we have seen in our data, we expect that it would probably be sufficient to re-evaluate the relevant flows and ship the list to router nodes once a week. Even if this list is not entirely up to date, triggers generated by other relevant flows on the same node will lead to accurate throughput estimates across *all* flows.

Our scheme has two main objectives: i) to reduce the required number of measurements for the computation of the IP traffic matrix at the three listed granularity levels, and ii) to do so in an accurate fashion. There is a natural tradeoff between our two objectives. If one were to collect measurements more frequently the OD flow estimates would be more accurate. On the other hand, cutting down the number of measurements may lead to higher TM estimation errors. Our scheme allows us to explore this tradeoff using parameters δ and H as knobs that a network operator could tune to achieve a target result. We explore this tradeoff in Section 6.

5.2 Benefits of the proposed scheme

The benefits of the proposed technique lie in its ability to reduce the number of measurements needed to obtain accurate traffic matrices. This has consequences that enable reductions in the associated communications, and storage overhead.

One benefit comes from the fact that we are essentially advocating that NetFlow (or a similar software) can be turned off between the checking hours. These periods of down time will last anywhere between 1 and 23 hours. This reduces the computation, or processing overhead at the router, thus freeing up the router CPU for other activities.

We reduce the communications overhead in two ways. First, instead of shipping R^2 (or L^2) records every hour, we ship them *on average once every few days* (we will see later on that the average number of days between updates is roughly 2 to 5). Second, we never actually ship R^2 records at once. Since routers are likely to trigger recomputations at different times, each router will send its new baseline at different times, thus *spreading the information transfer across time and space*. Similarly, we do not need to collect flow statistics on all the links of a router at the same time, but

whenever a particular link requires re-estimation of its row. This implies that *the statistics collection on the router can be asynchronous*, considerably reducing the peak load on the CPU.

This is quite an appealing feature that results from the use of a distributed approach. All other TM techniques based on inference require that the TM, or any model used within the inference procedure, be updated for all OD flows at once (i.e. a centralized approach). The attractiveness of our scheme is based on the fact that change in OD flow behavior will not be uniform across an entire domain. Some sites may experience a change in popularity, resulting in a shift in the volume of some OD flows but not others. Our scheme enables updates to a baseline model to take place only for those OD flows impacted by any such change in popularity.
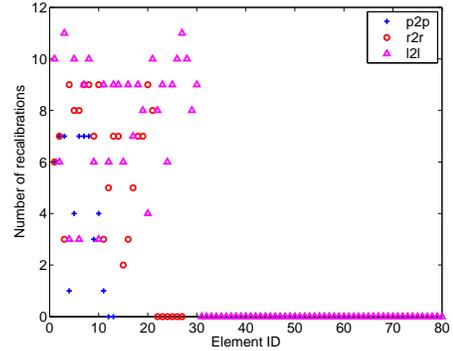

Figure 7: Number of baseline recalibrations per node.

To illustrate this further, we look at the number of times the baseline was recomputed at each link, node or PoP (depending upon the TM granularity), when $\delta = 0.5$ and $H = 24$. In Figure 7 we see that 2 out of 13 PoPs needed no recalibrations at all, 6 out of 27 routers, and 49 out of 80 links needed no recalibration. This demonstrates that running Netflow all the time would clearly be wasteful since it is unnecessary. Moreover, examining our 27 routers, for example, we can see that some updated as few as two times during our three weeks of data, while others updated as much as 10 or 11 times. Clearly different routers have different needs. Thus when we say we update the baseline only as needed, this implies that we determine necessity in both time (per day) and in space (per node).

6. RESULTS

In this section we show the results on our scheme's performance, discuss the calibration of its parameters and present the reduction in overheads that it enables.

6.1 Measurement Load

We quantify reduction in measurement load using the *number of hours between successive baseline estimations* for each node i. We ensemble all the data points from all the nodes, when $\delta = 0.5$ and $H = 24$ (the impact of δ and H is explored later in this section), and present the cumulative density function for the number of days between baseline re-estimation in Fig. 8. Baseline re-estimation appears to be

needed less often with higher levels of flow aggregation. In addition, 50% of the baseline re-estimations will be typically triggered after more than 2-3 days.
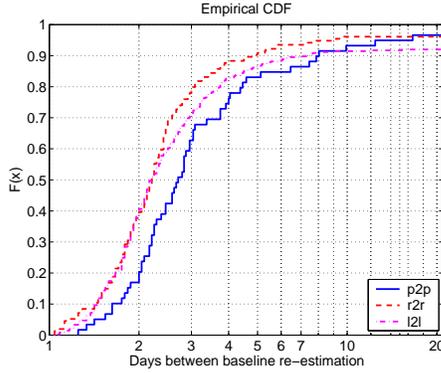

Figure 8: Days between baseline re-estimation for all three levels of flow granularity.

6.2 Error Assessment

A traffic matrix estimate yields one throughput estimate $\hat{X}(i,j,n)$ for each hour n for each OD flow (i,j). The relative error in the estimation of an OD flow is given by:

$$e(i,j,n) = \frac{\hat{X}(i,j,n) - X(i,j,n)}{X(i,j,n)}, \; when \; X(i,j,n) \neq 0 \quad (5)$$

The set $\{e(i,j,n)\}$ yields a large set of errors across both time and space (e.g. flows). For each OD flow (i,j), we have a time series for the estimation errors for that flow. For a fixed value of n, we can observe the errors across all the OD flows. There are different ways to view and summarize these errors.

We begin by looking at the entire set of error measurements at our disposal, across both space and time, simultaneously. The distribution of the set $\{e(i,j,n)\}$ is given in Fig. 9 for each of the three granularity levels. This distribution captures an instantaneous quantity - namely the probability that any flow, at any moment in time, incurs an error of some value. More than 80% of all our estimates (over time and space) yield a relative error that is contained between -25% and 25%. In other words, if one were to randomly select one flow and observe its estimated throughput at a random time interval, then this value would be within 25% of the actual throughput with a probability of 0.8.

When errors in our scheme occur, it will be due to a delayed detection of a change in fanout. If a change in fanout occurs, and the checking hour is a few hours later, then we will not pick up the change for that many hours. This will clearly generate errors during those hours. The instantaneous errors (Fig. 9) are sufficiently well contained, that this delayed change detection typically does not hamper our ability to track and estimate flows well (as in Fig. 6).

Temporal Errors. First we look at the errors in time. For each time interval, we compute the relative L2 norm across flows. In other words, we end up with one error

$$e_t(n) = \frac{\sqrt{\sum_{i,j=1}^{M}[(X(i,j,n) - \hat{X}(i,j,n))^2]}}{\sqrt{\sum_{i,j=1}^{M} X(i,j,n)^2}}$$
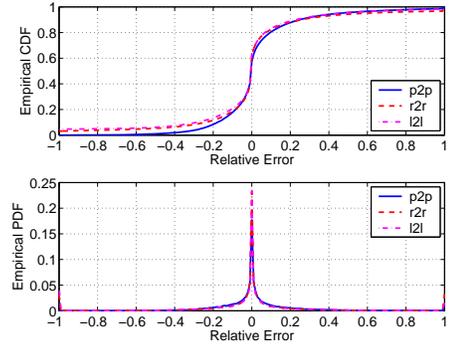

Figure 9: Relative error distribution for all three granularities.

for each time slot, in which the summation inside the L2 norm is taken over OD pairs (i,j). These results are displayed in Fig. 10. We see that 93% (75%) of the flows had errors less than 18% (20%) for p2p (or r2r), respectively. To check the performance of our scheme at different time scales, we also ran our method using 10 minute time intervals. The error results for this case are also presented in Fig. 10. We see that the results are very similar.
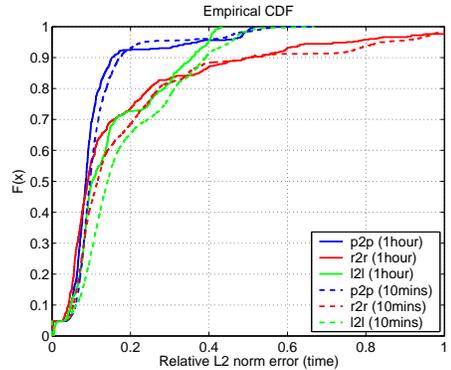

Figure 10: Relative L2 norm error in time.

Spatial Errors. We now look at errors across flows. For each flow we compute the relative L2 norm across time thus deriving a summary error per OD flow. In other words, we get one error metric per flow,

$$e_s(i,j) = \frac{\sqrt{\sum_{n=1}^{T}[(X(i,j,n) - \hat{X}(i,j,n))^2]}}{\sqrt{\sum_{n=1}^{T} X(i,j,n)^2}},$$

in which the summation inside the L2 norm computation is over time slots n. The distribution of these flow errors is given in Fig. 11. We see that these errors appear larger than the temporal ones do. This indicates that there are plenty of flows for whom it is difficult to achieve low errors at all instances of time (for both time scales of 1 hour and 10 minutes). Notice that the summary error $e_s(i,j)$ includes errors incurred for that flow whether it experiences high or smaller throughput (at which time instants this flow would be characterized as small). Our findings so far highlight the utility of examining errors in both time and space.
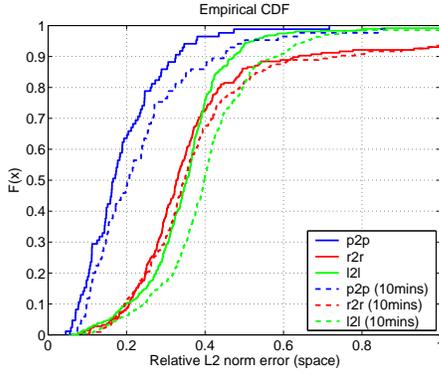
Figure 11: Relative L2 norm error in space.

Small Flows. Most previous studies look at the temporal errors, but eliminate the small OD flows from their evaluation by selecting the flows constituting the top 80% or 90% of the total demand. We visit the issue of small flows more closely here. The relative error metric $\{e(i,j,n)\}$ suffers from the following weakness. If the actual throughput achieved by an OD flow is very low, then the relative error can be rather high. It's not infrequent in our data for some OD flows to experience short periods of time when they may have negligible throughput. If a flow's throughput gets close to zero, then the relative error as defined in Eq. 5 can be huge, certainly much higher than 100%, until the baseline re-estimation is triggered. This leads to outliers in the error set $\{e(i,j,n)\}$.

We illustrate this scenario with a sample flow in Fig. 12. This particular router to router flow experiences two small periods of time when its throughput is very small. The corresponding relative errors measured are on the order of 10^4. We claim that these outliers are not important for two reasons. First, we see in the top portion of Fig. 12, that our estimates can still do a good job at tracking the actual OD flow. Second, flows nearing zero throughput can be ignored by most traffic engineering applications at this time scale. Recall that we target network design and planning applications, and not anomaly detection or billing, when behavior needs to be tracked at smaller time scales, and where TMs would need to be continuously measured (in our scheme this would require $H = 1$).
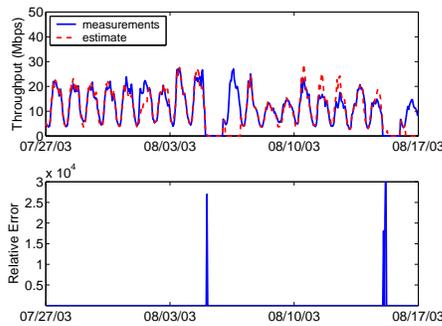

Figure 12: Flow with brief periods of inactivity. Upper: estimated behavior, lower: relative error.

An error computation that selects the flows constituting the top 80% or 90% of the load, *at each time slot,* can be ambiguous since a flow such as the one in Fig. 12 is sometimes included (during periods of regular activity) and sometimes excluded (during periods of low activity). To avoid having to pick thresholds for excluding OD flows (or portions of an OD flow's time series) in error metrics we use a *weighted relative L2 norm* error metric, denoted $e'(i,j)$, where the weights are set to be proportional to the actual OD flow throughput as follows:

$$e'(i,j) = \frac{\sqrt{\sum_{n=1}^{T}[(X(i,j,n) - \hat{X}(i,j,n))^2 w(i,j,n)]}}{\sqrt{\sum_{n=1}^{T} X(i,j,n)^2}},$$

$$w(i,j,n) = \frac{X(i,j,n)}{\sum_{k=1}^{T} X(i,j,k)}$$

This gives us a single error metric for *each* OD flow. Note that the threshold approach applied in previous efforts is in essence a temporal error metric (an $e(n)$ metric). We prefer to deal with the issue of small flows via a spatial error metric (an $e(i,j)$ one) rather than a temporal one for the following reason. The threshold approach does not only eliminate small flows, but also the portions of any flow when it is near zero. Our weighted error metric allows us to include flows throughout their lifetime, but weighed instantaneously according to their volume; the weights in our metric vary over time as does the flow volume. This error definition assigns small weights to those time instances when the flow exhibits little activity, and larger weights to high activity time slots. In other words, instances of flows are assigned a relevance level according to their volume. This is reasonable since carriers design their networks such that they can accommodate the maximum amount of traffic that may be offered. In addition, this definition does not bias our error computation in any favorable way - if a large error is made when a flow is at its peak, then this error will have a heavy weight. The weighted relative L2 norm metric gives us an alternate view of the errors and alleviates the problem of outliers without eliminating them.

Fig. 13 presents the empirical cumulative density function of the weighted relative L2 norm. More than 90% of the flows at all three levels of granularity experience a flow error less than 10%. The fact that this figure is quite different from Fig. 10 highlights the distorting effect that OD flows dropping to near zero levels may have, as well as the effect of the delayed reaction mechanism employed by our scheme when these events occur.

In order to see which flows are lying in the tail of this distribution, we plot the weighted relative L2 norm error versus the flow throughput in Fig. 14. This shows that the smaller flows are the ones that incur the larger errors. Indeed, this figure shows that most large flows are typically estimated with an error below 5%. Other traffic matrix estimation methods [13, 5, 11] have also observed that they estimate the large flows well, and better than the small flows.

6.3 Frequency of change detection

Parameter H in our scheme impacts the frequency of baseline change detection. Each node randomly selects one hour in the future H hours to test its fanouts against its 24 hour baseline. We now look into the impact of parameter H on
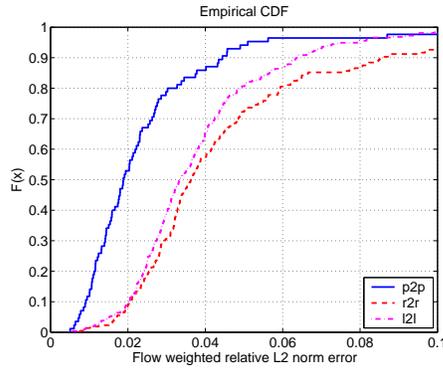
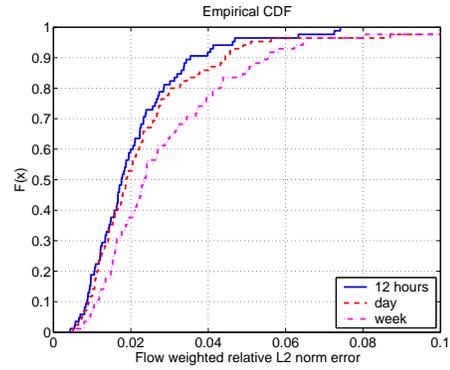Figure 13: CDF of weighted relative L2 norm error.

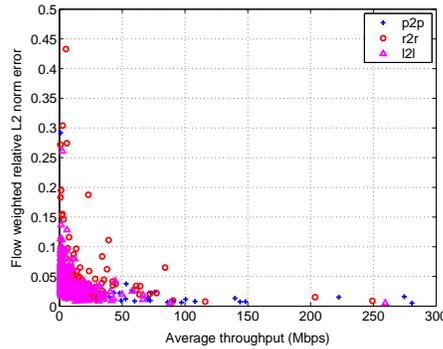Figure 15: Impact of frequency of checking for baseline diversion.

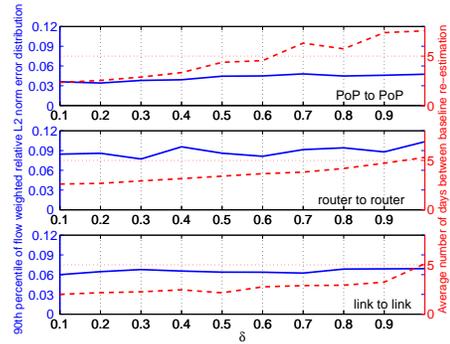Figure 14: Weighted relative L2 norm error vs. flow throughput.

Figure 16: Impact of diversion threshold δ (dashed line for right-hand y-axis).

the performance of the proposed scheme. We set parameter δ equal to 0.5, and apply our scheme while testing for diversions (i) every 12 hours ($H = 12$), (ii) every 24 hours ($H = 24$) and (iii) every week ($H = 168$). Fig. 15 illustrates that testing for fanout discrepancies more frequently than once a day does not lead to significant improvement in terms of flow errors. However, decreasing H to one week does lead to deterioration in OD flow estimates. These observations were found to hold across all three levels of granularity, and across values of δ spanning from 0.1 up to 1. Consequently, we conclude that testing for fanout change once a day is frequent enough for our purposes.

6.4 Tradeoff between baseline re-estimation and accuracy

According to our scheme baseline re-estimation is triggered when the measured fanout diversion exceeds a specific value of δ. If fanouts are updated less often, TM estimates will be less accurate but the savings in terms of measurement load (i.e., running flow monitors) will be larger. On the other hand, when the fanouts are updated more frequently accuracy is likely to be higher. In this section we look into the tradeoff between accuracy and reduction in measurement load for different values of δ. In Fig. 16 we present the 90th percentile of the weighted relative L2 norm error distribution and the average number of days between baseline re-estimation for different values of δ.

We observe the following. (i) Baseline re-estimation needs to be performed more frequently for finer levels of the traffic matrix granularity. (ii) Greater values of δ lead to higher flow errors and greater periods of time when the baseline measurements can be used unaltered. (iii) The proposed scheme leads to errors below 12% across all three levels of granularity and across all values of δ, and incurs baseline re-estimation in the worst case every 3 days, when $\delta = 0.1$. (iv) Our error metric does not appear to be that sensitive to δ. This lack of sensitivity to δ is an attractive feature of our scheme since it implies that careful optimization of this parameter is not needed, but rather that a wide range of values will do well. The average number of days between baseline recalibration appears to be a little more sensitive to δ as we do see a variation between 3 to 7 days. We have selected $\delta = 0.5$ in most of our experiments; it appears quite reasonable with typical errors below 5% and baseline recalibrations occurring once every 3 to 4 days. The final decision as to how to tune such a scheme's parameters will ultimately lie with the network operators as they will choose the accuracy versus overhead tradeoff.

We consider that running a flow monitor for the purposes of TM estimation constitutes measurement overhead for this task. We have been advocating a limited use of such monitors. In order to assess this tradeoff, we present Fig. 17. For the p2p case, each of the 10 points in Fig. 17 corresponds to a different value of $\delta = 0.1, 0.2, \ldots, 1.0$ (similarly for r2r and

l2l). The y-axis states the fraction of absolute relative errors below 25% for the given scenario (value of δ and granularity level). The x-axis is our metric for measurement overhead. To compute this metric we count the number of hours each link ran the flow monitor and sum across the entire network. We then divide by the total number of link-hours in our measurements (e.g. 81×504 hours). Thus a value of 0.25 on this axis means that overall (across time and space), monitors were used during 25% of the 3-week lifetime of our experiment.
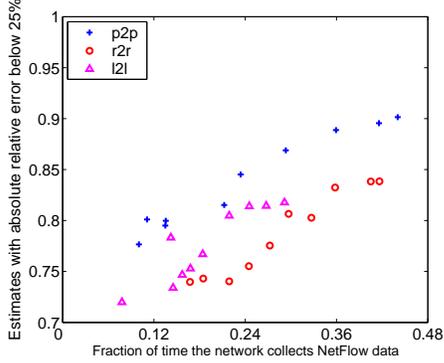

Figure 17: Tradeoff between accuracy and measurement overhead.

The percentage of error improvement achieved using additional measurements appears to be non-linear. For all three levels of granularity there is a plateau after which large increases in measurement overhead are needed for further error reduction. This figure implies that with roughly 45% of the measurements from an always on monitor, one can estimate between 80-90% of the flows quite well. Clearly monitoring at 100% is the only way to achieve zero errors, all the time, for all error metrics. However this level of accuracy is typically not needed for most traffic engineering applications.

7. REDUCTIONS IN OVERHEAD

To illustrate the impact of our scheme on overheads, we now quantify the savings that would result over a period of 3 weeks (the length of our entire dataset). Letting T denote the duration of this 3 week period in one hour increments we have $T = 504$ for our calculations. In Table 3 we summarize both the router storage and communications overheads incurred by the centralized, distributed, and trigger-based solutions. By 'distributed' we refer to a solution in which flow monitors are left on all the time, thus continuously measuring the traffic matrix without using any estimation at all. We use the numbers from our backbone (e.g., $R = 27$, $L = 81$, etc); these are stated at the top of the table.

According to our scheme each router needs to store 24 hours of its fanouts as well as one extra row of fanouts that corresponds to the randomly selected hour. Consequently, at any point in time the router needs to store at most 25 rows of fanouts. In our network this means that for the computation of the p2p traffic matrix each router needs to store $(H+1)*P$ records (or $25*13 = 325$) at any point in time. In Netflow v8 every router stores approximately 67,000 records for each one of its links, with an average of 3 links per router that means 210,000 records per router. (The centralized and distributed router storage in this table is the same as that in Table 1. They are included here for comparative purposes.) Clearly, the storage for the trigger-based scheme is significantly lighter than for today's centralized solutions. The trigger-based scheme does slightly worse than the distributed solution because the distributed solution maintains less by shipping it out every hour to the collection station.

The communications overhead for the centralized and distributed solutions in this table differ from that in Table 1 because we now include the length of time over which we assess these overheads. For example the communications for an r2r matrix is now $T*R^2$ (rather than just R^2). The numbers in the table correspond to the total number of records shipped through the network during this three week period.

To quantify the reduction in communication overhead under our approach, we count how many times baseline re-estimation was triggered for each link or router throughout the 3 week period. The numbers of triggers generated for each TM granularity level are given at the top of the table. The total overhead is the sum of the initial baseline (24 measurements) plus the number of baseline measurements that are triggered for specific links or routers throughout the network. For example, in the l2l case, the initial baseline sent contains $24*L^2$ records. After that we simply communicate the rows of the matrix that change. With 333 triggers (each of which requires another 24 hours of measurement), the remainder of the overhead is $333*24*L$.

In order to compare the three solutions, we look at the percentage reduction from one scheme to the next. We quantify the reduction in overhead as the fraction of the overhead difference between two schemes divided by the original number of records. We find that moving from a centralized to a distributed approach leads to 99.99%, 99.98% and 99.87% reduction for the p2p, r2r and l2l levels respectively. Alternatively, we can say that the distributed scheme's overhead is roughly a factor of a thousand less than the centralized one. Relative to the distributed solution, the trigger-based approach leads to an extra 85.18%, 69.13% and 75.66% reduction for the three levels respectively.

Because the overhead for our scheme involves a constant initial factor plus a term that is a function of time (i.e., the number of triggers), the impact of the initial constant factor eventually disappears after enough time. If D denotes the average number of days between baseline re-estimation, then essentially we will need to collect one day's worth of measurements every D days. The savings here, relative to a full-on distributed scheme, will converge to $\frac{D-1}{D}$. For example, if $D = 4$, then we recompute the baseline once every 4 days, leaving 3 days without any measurements communicated to the collection station. This reduction in measurement overhead was estimated to be 75.78%, 68.85%, and 65.47% for the p2p, r2r and l2l respectively (for the respective values of D derived from our measurements), as compared to the always on measurement method.

8. CONCLUSIONS

In this paper we addressed the question of whether or not Internet traffic matrices can be obtained via direct measurement by flow monitors on routers. We showed that with today's technology, centralized solutions are needed and that these indeed are computationally prohibitive.

We strongly encouraged moving toward a distributed solution and illustrated the reduction in overheads that this

Overhead in # records for the three weeks of measurements		
$T = 504, P = 13, R = 27, L = 81, L(r) = 3, F(l) = 67,000$		
H=24, # triggers (p2p)=57, # triggers (r2r)=148, # triggers (l2l)=333		
Router Storage		
	Centralized	Distributed	Trigger-Based
p2p	$L(r) * F(l) = 210,000$	$P = 13$	$(H + 1) * P = 325$
r2r	same as above	$R = 27$	$(H + 1) * R = 675$
l2l	same as above	$L(r) * L = 243$	$(H + 1) * L(r) * L = 6,075$
Communications Overhead		
	Centralized	Distributed	Trigger-Based
p2p	$T * L * F(l) = 2,735,208,000$	$T * R * P = 176,904$	$24 * R * P + \#triggers * 24 * P = 26,208$
r2r	same as above	$T * R^2 = 367,416$	$24 * R^2 + \#triggers * 24 * R = 113,400$
l2l	same as above	$T * L^2 = 3,306,744$	$24 * L^2 + \#triggers * 24 * L = 804,816$

<p align="center">Table 3: Summarization of overheads for a three week period.</p>

would enable. For example, the reduction in communications overhead can be as large as 99% since we change by orders of magnitude the amount of data shipped through the network. Although recent advances in Netflow illustrate a move toward being able to compute traffic matrices, we explained why these advances are not sufficient. We identified the critical factors for which implementation changes are needed to enable truly distributed solutions to this problem, and presented these as two recommendations to router manufacturers. These recommendations include (i) implementing a function to map destination network prefixes to egress links or routers within the domain *at the router itself*, and (ii) modifying the definition of the flow record in order to include the result of this mapping. We believe that once this function and definition can be implemented, distributed traffic matrix measurement by routers would become a reality.

Nonetheless, we showed that it is possible to go beyond this vision, further reducing the frequency with which measurements need to be taken. This is based on our finding that the node fanouts are remarkably predictable, at three granularity levels. We presented a scheme that exploits this finding to further reduce communications overhead and the frequency of router measurements. We showed that by taking measurements only once every few days, we can obtain a traffic matrix that is accurate on the time scale of hours. We demonstrated that by allowing Netflow to be turned on and off, on an as needed basis, we can reduce communications overhead by 70-85% and can reduce measurement overhead by 65-75% relative to a direct measurement solution that leaves flow monitors on continuously. (This also implies that there is a strong potential for advanced sampling techniques to succeed in the area of TM measurement.) Our approach is capable of achieving such reduction while introducing limited estimation errors. Moreover, unlike other schemes, ours has knobs that can be tuned so as to achieve a specific target error rate. The attractiveness of our scheme is that it is simple, scalable (for traffic matrices at all three granularity levels) and works well, thus rendering our approach very practical. In future work we intend to compare our technique with proposed inference techniques, and quantify the differences against the same dataset using the same error metrics.

9. ACKNOWLEDGEMENTS

10. REFERENCES

[1] J. Cao, D. Davis, S. VanderWeil, and B. Yu. Time-Varying Network Tomography: Router Link Data. *Journal of the the American Statistical Association*, 95(452), 2000.

[2] Cisco. NetFlow Services Solutions Guide, July 2001.

[3] Cisco. Cisco IOS NetFlow Version 9 Flow-Record Format , June 2003.

[4] N. Duffield, C. Lund, and M. Thorup. Properties and Prediction of Flow Statistics from Sampled Packet Streams. In *ACM Sigcomm Internet Measurement Conference*, Marseille, France, Nov. 2002.

[5] A. Feldmann, A. Greenberg, C. Lunc, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Transactions on Networking*, June 2001.

[6] G. Liang and B. Yu. Pseudo Likelihood Estimation in Nework Tomography. In *Proceedings of IEEE Infocom*, San Francisco, CA, Mar. 2003.

[7] A. Medina, C. Fraleigh, N. Taft, S. Bhattacharyya, and C. Diot. A Taxonomy of IP Traffic Matrices. In *SPIE ITCOM: Scalability and Traffic Control in IP Networks II*, Boston, Aug. 2002.

[8] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic Matrix Estimation: Existing Techniques and New Directions. In *ACM SIGCOMM*, Pittsburgh, USA, Aug. 2002.

[9] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot. Long-Term Forecasting of Internet Backbone Traffic: Observations and Initial Models. In *Proceedings of IEEE Infocom*, San Francisco, March 2003.

[10] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP Routing Stability of Popular Destinations. In *IMW*, Marseilles, France, Nov. 2002.

[11] A. Soule, A. Nucci, E. Leonardi, R. Cruz, and N. Taft. How to Identify and Estimate the Largest Traffic Matrix Elements in a Dynamic Environment. In *ACM Sigmetrics*, New York, June 2004.

[12] G. Varghese and C. Estan. The Measurement Manifesto. In *Proc. of the 2nd Workshop on Hot Topics in Networks*, Cambridge, MA, Nov. 2003.

[13] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads. In *ACM Sigmetrics*, San Diego, CA, 2003.

[14] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An Information Theoretic Approach to Traffic Matrix Estimation. In *ACM SIGCOMM*, Karlsruhe, Germany, August 2003.