Design of IGP Link Weight Changes for Estimation of Traffic Matrices

Antonio Nucci[1], Rene Cruz[2], Nina Taft[1], Christophe Diot[3]
[1] – Sprint Advanced Technology Laboratories, Burlingame, CA, USA
[2] – University of California San Diego, San Diego, CA, USA
[3] – Intel Research Cambridge, Cambridge, UK

***Abstract*—We consider the traffic matrix estimation problem in IP backbone networks, whose goal is to accurately estimate the volume of traffic traveling between network endpoints. Previous approaches to this problem involve measuring the volume of traffic on each link in the network during a time interval where the routing configuration is fixed, and exploit a statistical model of the traffic in order to obtain an estimate of the traffic matrix. These previous approaches are prone to large estimation errors because the link measurements from a fixed routing scenario constitute a data set that is simply too limited to provide enough data to enable estimation procedures that yield very small errors. In this paper we propose the idea of collecting link measurements under *multiple routing scenarios* so that the traffic matrix can be determined very accurately. We present an algorithm for determining a sequence of routing configurations, each of which is specified by a set of link weights. We incorporate carrier requirements into our algorithm so that our proposed routing configurations are operationally viable. We present the results of applying our algorithm to some representative IP backbone topologies and discuss the performance trade-offs that arise.**

Keywords: Network Measurements, Traffic Engineering, Combinatorics.

I. INTRODUCTION

The problem of estimating Internet traffic matrices for backbone networks has recently become a subject of much interest. The end goal is to determine the traffic demand that flows between two end nodes in a network. Obtaining accurate traffic matrices is very important for network operators that conduct traffic engineering tasks. Carriers have recently demonstrated active interest in obtaining traffic matrices in order to improve their ability to do proper capacity planning and reliability analysis. Current direct measurement approaches, such as Cisco's Netflow, are insufficient in that they are not deployed everywhere and can be disadvantageous to turn on as some versions can consume excessive amounts of CPU load. Moreover the process of collecting and processing the data from all network links is still cumbersome and involves too much overhead to be practical.

Thus research efforts in the area of traffic matrix estimation have focused on modeling combined with statistical inference techniques. The inputs to an estimation procedure are measurements on link load levels coming from SNMP (link count vector), and information on the routes used to carry traffic between end nodes. Such nodes can be routers, Points-of-Presence (PoP), links or even prefixes. A pair of such end nodes is commonly termed an origin-destination (OD) pair. Most recent research [1], [2], [3], [4], [5], [6] has focused on either PoP-to-PoP traffic matrices, or router-to-router matrices when the number of routers is limited.

Prior work in [1], [3], [4], [5], [6] consider the traffic matrix estimation problem using statistic inference techniques. All of these prior methods are prone to moderately large errors. Errors in the range of 10-25% are typical and some OD pairs can have errors above 100%. The reason is because all the inference techniques operate on a system of linear equations that is severely under-constrained. It is intuitive that one cannot reduce estimation errors to arbitrarily small values under such very limited information. A fundamental problem of traffic matrix estimation is to find ways to obtain new information that reduces the "underconstrainedness" of the problem.

The routing configuration in a network is determined by an IGP routing protocol. Most of today's large IP networks use a link-state protocol such as ISIS or OSPF for intra-domain routing. Every link in the network is assigned a weight, and the cost of a path is measured as the sum of the weights of all links along the path. Traffic is routed between any two nodes along the minimum cost path which is computed using Dijkstra's shortest path algorithm (SPA). As a consequence, each link in the network is shared by multiple OD pairs and the traffic observed on each link is the aggregation of traffic carried by each of the OD pairs flowing on it.

The relationship between the traffic matrix, the routing and the link counts can be described by a system of linear equations $Y = AX$, where Y is the vector of link counts, X is the traffic matrix organized as a vector, and A denotes a routing matrix, in which element a_{ij} is equal to 1 if OD pair j traverses link i or zero otherwise. In today's networks, Y and A are readily obtained; Y comes from SNMP data and A can be computed from the link weights and the topology. The elements of the routing matrix can take on fractional values if traffic splitting is supported. The problem at hand is thus to estimate X. (We define our notation more thoroughly later on.) The underconstrainedness of the problem is manifested by the routing matrix A having less than full rank.

In current TM estimation methods, SNMP data is gathered under a given fixed routing scenario. Specifically, the link

This work was done when Nina Taft was at Sprint ATL; she is currently affiliated with Intel Research Berkeley, CA.

weights are fixed and the routes are determined as above, leading to an under-constrained problem. In contrast, in this paper we address the underconstrainedness of the problem by effectively increasing the rank of the routing matrix. We achieve this by reconfiguration of the link weights.

If we were to change a link weight, some of the OD pairs on that link could be moved onto other paths. Analysis of the SNMP link measurement data under these new routing conditions effectively decreases the ambiguity inherent in the link measurements. We use the term snapshot to denote different images of the routing; each snapshot has its own set of IGP link weights. Consider two different snapshots k_1 and k_2. Each snapshot has its own associated routing matrix $A(k_1)$ and $A(k_2)$. Collecting measurements under two snapshots implies that we now have two link counts for each link, one for each routing scenario. The rank of the two matrices, considered together, may be larger than the rank of the original routing matrix. This would happen if some of the new link counts (for links carrying a different combination of OD pairs than in the original routing scenario) generated equations that were linearly independent of those in the original system. Our idea is to do this repeatedly until we have enough independent equations so as to have a full rank routing matrix.

Such an approach raises a number of questions. How many routing changes must be carried out to achieve full rank? Which weights should be changed and by how much? Can we find a minimal set of proposed weight changes? We will address all of these issues in this paper. We develop a heuristic algorithm for determining which weights to change and by how much they should be changed. Another key issue with such an approach is whether or not carriers are willing to do such a thing in order to obtain very accurate estimates of traffic matrices. Our discussions with operations personnel indicate that carriers are willing, but only under certain conditions. These conditions, or operational constraints, are incorporated directly into our algorithm. By studying two differently sized Tier-1 backbones, we show that with an algorithm such as ours we can get close to full rank with a reasonable number of snapshots, but not obtain it completely without difficulty with respect to carrier requirements. As a consequence, we believe that the most promising approach for traffic matrix estimation will be a two-step hybrid method in which our algorithm is run first and followed by any of the previous inference methods. Only the "safe" snapshots, in terms of carrier requirements, obtained by our algorithm would be carried out. With our safe snapshots we increased the rank from 25% to 90% in the main network we studied. This kind of improvement in the rank should lead to a dramatic reduction in estimation error rates.

The rest of the paper is organized as follows. In Section II we describe the formal problem statement of synthesizing a set of snapshots designed to yield information about the traffic matrix. In Section III we illustrate the essence of our idea with an example and we discuss some practical considerations. In Section IV we present our heuristic while in Section V we show some results for two real Tier-1 backbones operating in USA. Section VI concludes the paper and discusses possible directions for future research.

II. PROBLEM STATEMENT

Let a network be represented by an undirected graph $G(\mathcal{N}, \mathcal{L})$ where $\mathcal{N} = \{1, ..., N\}$ corresponds to the set of nodes and $\mathcal{L} = \{l_1, ..., l_L\}$ corresponds to the set of links connecting the nodes. A propagation delay d_h and an integer IGP weight w_h are associated with each link $l_h \in \mathcal{L}$. Let $\mathcal{W} = \{w_1, ...w_L\}$ be the initial set of IGP weights given with the network studied. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_P\}$ denote the set of OD pairs flowing across the network. Each OD pair $p \in \mathcal{P}$ is routed along the minimum cost path between the origin and destination nodes with an end-to-end delay d_p equal to the sum of the propagation delays of each link along the path. If there are multiple equal length minimum cost paths between the origin and destination nodes, we assume that traffic is split evenly across all of the equal length paths. The nominal traffic matrix is represented as a column vector $x = (x_1, x_2, \ldots, x_P)^T$, where x_p represents the volume of traffic generated for OD pair p during a measurement interval. A set of IGP link weights is called a *snapshot* in this paper.

We will consider the problem of synthesizing a set of snapshots designed to yield information about the nominal traffic matrix x. For a given snapshot, traffic for each OD pair is routed along the shortest length path from the origin node to the destination node. The routing induced by a snapshot k is described by a *routing matrix* $A(k)$, which is an $L \times P$ matrix. Specifically, the element of $A(k)$ in the l^{th} row and p^{th} column is defined as the fraction of traffic for OD pair p that is routed on link l under snapshot k. For example, if all of the traffic for OD pair p is routed along link l, then $A(k)_{lp} = 1$. If there are two shortest paths between the origin and destination node of p, and link l belongs to exactly one of these paths, then $A(k)_{lp} = 1/2$.

For snapshot k, let $Y(k)$ be the $L \times 1$ column vector whose l^{th} component is the total volume of traffic which is measured on link l. We then have the relation

$$Y(k) = A(k)x \qquad (1)$$

In an operational network there is a nominal set of link weights, which we refer to as snapshot zero. The problem considered in this paper is to synthesize a set of K snapshots 1,2, K, such that the corresponding link measurement vectors $Y(0)$, $Y(1), \ldots, Y(K)$ completely determine the traffic matrix vector x. Snapshot 0 is assumed to be given, and corresponds to the link weights in the nominal operational network.

We stack the link measurement vectors $Y(k)$ into an aggregate link measurement column vector Y, as well as stack the routing matrices $A(k)$ into an **aggregate routing matrix** A, i.e. $A = [A^T(0), A^T(1), \ldots, A^T(K)]^T$.

With this notation, we have $Y = Ax$. If A has full rank P, then the traffic matrix x can be fully determined by measurement of Y. The main problem considered in this paper is to find a set of acceptable snapshots so that the resulting aggregate routing matrix has full rank P.

To be deemed acceptable, each snapshot must conform to carrier requirements about the state of the network. For example, under the current knowledge about the traffic matrix, the resulting average load on each link needs to be below a prescribed limit. In addition, a snapshot must not lead to paths with total propagation delays that are excessive. In particular, we require that the average end-to-end propagation delay for each shortest path is below a prescribed limit, where the average is performed over all OD pairs. We discuss carrier requirements at greater length in the next section; this is very important for such a method to become practical.

For simplicity, in this paper we will assume that the traffic generated by every OD pair is the same for every snapshot. In general, since the snapshots are realized during different intervals of time, the amount of traffic generated by each OD pair will depend on the snapshot. Since the traffic matrix itself can fluctuate throughout a day, such fluctuations need to be explicitly captured by the model used inside the traffic matrix estimation procedure. Such traffic fluctuations accruing across different measurement intervals are discussed in [7], and are ignored in this paper.

A method based on the idea of using multiple snapshots could be implemented in one of two ways. If the link weight changes were carried out during the same one hour slot each day, then one could fairly safely assume the traffic matrix is stationary, and a simple model for the OD elements (such as a Gaussian noise process) would suffice for the estimation step. However if one carried out the changes throughout the day, then the traffic matrix could be cyclostationary (if it has strong diurnal trends) or even nonstationary. In these cases, more complex models for the OD pairs are needed. In [7] we present a model for OD pair traffic fluctuations that incorporates both diurnal trends and noise elements. The algorithm presented in this paper can be used in conjunction with such a model. We point out that our algorithm here can be used either alone for the case of stationary traffic, or as a preprocessing step to a variety of inference techniques, as long as they incorporate long-term fluctuations in the traffic matrix model and estimation procedure.

III. Our Approach

Before tackling the problem of designing an algorithm to identify the desired set of snapshots, we first motivate our approach of changing routes to learn more about the traffic matrix, through an example that illustrates the essence of our idea.

A. Example

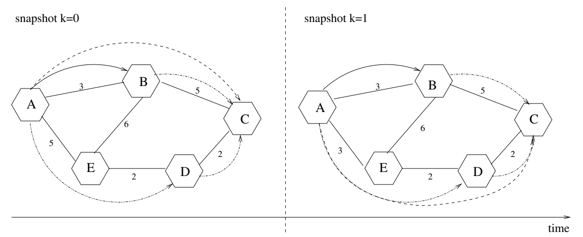

Fig. 1. Essence of new idea. Example

Consider the network shown in Fig. 1 composed of five nodes interconnected by six unidirectional links. Each link has an associated weight and the traffic from each OD pair is routed along the shortest cost path. For simplicity, we consider only five OD pairs (indicated by arrows). On the left of Fig. 1 we represent the network in its normal state, when neither failure events are observed nor link weight changes are implemented (snapshot $k = 0$). Snapshot 0 would generate the following linear system of equations.

$$\begin{bmatrix} Y_{AB} \\ Y_{AE} \\ Y_{BC} \\ Y_{ED} \\ Y_{DC} \\ \ldots \end{bmatrix} = \begin{bmatrix} 1\ 1\ 0\ 0\ 0\ \ldots \\ 0\ 0\ 0\ 1\ 0\ \ldots \\ 0\ 1\ 1\ 0\ 0\ \ldots \\ 0\ 0\ 0\ 1\ 0\ \ldots \\ 0\ 0\ 0\ 0\ 1\ \ldots \\ \ldots \end{bmatrix} \begin{bmatrix} X_{AB} \\ X_{AC} \\ X_{BC} \\ X_{AD} \\ X_{DC} \\ \ldots \end{bmatrix}$$

We only write down the portion of this system related to the links and OD pairs considered in this example. The rank of routing matrix $A(0)$ is four. Two of the five OD pairs ((A, D) and D, C) can be estimated exactly because in this simple example they do not share their links with other OD pairs. On the right of Fig. 1 we show the effect of decreasing the weight of link $l = (A, E)$ from 5 to 3 (snapshot $k = 1$). This perturbation in the weights causes the re-routing of the OD pair (A, C) through the new path $\{(A, E), (E, D), (D, C)\}$. This snapshot generates a new system of linear equations, i.e. a new routing matrix $A(1)$, that can be appended to the previous set. One line of the new routing matrix would look like $Y_{AB} = [1\ \ 0\ \ 0\ \ 0\ \ 0\ \ \ldots]\ \ [X_{AB}\ \ \ldots]^T$. We can see that adding this to the original system of equations adds a new linearly independent equation into the system. As a consequence, the new system $[A(0)^T\ \ A(1)^T]^T$ is full rank and all five OD pairs can be exactly estimated.

B. Practical Considerations

Changing the link weights in a live operational network means that traffic will get shifted around. Such traffic shifts could cause some links to become highly loaded - a condition that carriers try to avoid in order to prevent congestion. Such link weight changes could cause some PoP pairs to traverse long paths, thereby increasing the delay between those PoP pairs. Carriers sign Service Level Agreements (SLA) with their customers and the terms of these agreements must not be violated. SLAs typically specify a guarantee on average end-to-end delay across all OD pairs in the carrier's backbone.

We thus provide the following two user-selectable inputs to our algorithm: (1) average end-to-end delay, called the *delay-limit* and (2) maximum link load, denoted the *load-limit*. The idea is that our algorithm will reject any weight change that would cause either of these two thresholds to be exceeded. Our algorithm can run in three different modes: using only the delay-limit, using only the load-limit, and using both inputs.

Supporting the load-limit feature can be tricky because in order for our algorithm to evaluate the resulting link loads, due to a given weight change, we need to know the traffic matrix itself. But the traffic matrix is the very thing we are trying to estimate. However, we point out that once carriers start using traffic matrix estimation methods regularly they will always have an old one available. It is also possible to obtain a TM from less accurate methods [1], [3], [4]. It is thus not unreasonable to assume that a carrier would have some form of TM at hand, albeit an erroneous one. Such a TM would probably be sufficiently reasonable to use in the process of obtaining very accurate TMs. If such a TM were not available, ISPs could use our method using the delay-limit input alone.

Carrier's have methods for selecting the set of IGP weights under which they like to operate [10]. In IP backbones today, such weights are not changed very often - perhaps once a week, or even once a month. Weights are often changed either after a failure event or after the addition or removal of large customers. Because operators prefer not to move the network away from its usual good operating point, they would clearly like to limit the number of such weight change events. One of the goals of our algorithm is thus to find the minimal set of weight change events needed to obtain full rank on the routing matrix.

We use the term *snapshot* to refer to a weight change event. A weight change event could involve changing a single weight or multiple weights simultaneously. An operator is more likely to prefer changing one or two link weights at the same time rather than changing a large numbers of links simultaneously. There are two main reasons for this. First, operators are typically not comfortable moving the network *too far* from its good operating point. Second, each ISIS weight change requires the modification of two router interfaces at the same time. Changing many router interfaces across multiple router interfaces can become burdensome because today link weights are changed manually. Our heuristic thus prefers to identify as many snapshots as possible in the minimal set that only involve changing the weight on a single link for a given snapshot. We also consider changes involving two or three links simultaneously, but only after seeking all useful single link weight changes. We limit the number of simultaneous link weight changes to three to capture this constraint. This is just a rule of thumb to select a very small number. The carrier can decide later exactly what their limit is; this choice would not impact the approach and ideas behind our heuristic algorithm.

IV. Heuristic Algorithm

Our objective is to identify a small set of snapshots which are acceptable in the sense that the resulting traffic performance stays within the bounds of our *delay-limit* and *load-limit*. To get an appreciation for the magnitude of the complexity of problem at hand, we now consider the cardinality of the space of possible solutions. Let L denote the number of links in a network. Most carriers limit the range of possible weight settings to be within a predefined minimum, W_{min} and a predefined maximum W_{max}. Weights can only take on integer values and thus the number of possible weight settings for a single link is $W = W_{max} - W_{min}$. Note that a single snapshot corresponds to a set of L link weights, and thus the number of possible snapshots we could potentially examine is thus $2^{(W^L)}$, which is an enormous number.

Our heuristic algorithm works roughly as follows. First we identify a set of snapshots which involve weight changes on a *single* link, relative to the original set of link weights used in the operational network. We consider only single weight change snapshots which result in a change in the routing for at least one OD pair. Such snapshots may or may not meet operational needs, so we prune down the set of snapshots to only those which do. The set of snapshots we seek may be a subset of all these candidate snapshots. Since the number of possible subsets is very large, we then form an ordering of these snapshots according to a heuristic. Using this ordering, we then iteratively build the aggregate routing matrix, adding snapshots in the given order as long as it results in an increase in the rank of the aggregate routing matrix, and stopping as soon as the aggregate routing matrix has full rank. If we pass through all candidate (single weight change) snapshots without achieve full rank, we then consider a set of snapshots that are slightly more difficult to implement, which involve changing link weights for a pair or triplet of links. If we are still not able to achieve full rank by considering these additional snapshots, we relax our operational constraints on link load and end-to-end propagation delay in order to enable more snapshots to be considered.

More specifically, our heuristic algorithm is organized into six steps, where the first four steps consider only single weight change snapshots. The first step identifies single weight snapshots which result in prescribed movements of routes for OD pairs onto specific links. The second step prunes the snapshots found in step one to only those which satisfy operational constraints on link loading and end-to-end propagation delay. In step three, we order the remaining snapshots found after step two. In step four, we sequentially build up a set of snapshots by adding one snapshot at a time, where a snapshot is included only if it increases the rank of the aggregate routing matrix. In step five, if necessary, we consider additional snapshots, first those involving weight changes for pairs of links, and then triplets of links. Finally, in step six, if necessary, we repeat the process but relax the operational constraints to widen the space of snapshots considered. Carriers can always choose not to execute steps five and six. However we chose to include them here in order to understand what it takes to obtain a full rank routing matrix and to understand the trade-offs involved.

Step 1: Pruning by Route Mapping

In this step we identify a set of snapshots, where each snapshot corresponds to an IGP weight change on a single link. Let $\Delta'_h = \{\delta w_h(p_1), ..., \delta w_h(p_P)\}$ be the set of potential IGP weight perturbations to apply to a given link l_h. Each perturbation $w_h(p_k)$ provides a routing change for OD pair p_k. Recall that P is the number of OD pairs in the network. For each link $l_h \in \mathcal{L}$, we determine the set Δ'_h as follows.

First, for an OD pair p, let $\theta_0(p)$ be the path length of the minimum cost path from the origin node of p to the destination node of p under the initial set of link weights $\mathcal{W}$. Let $\theta_h(p)$ be the length of the shortest path which is constrained to use link l_h. More precisely, let the origin and destination nodes of p be s and t, respectively, and let the initial and terminal nodes of link l_h be x and y. With this notation, $\theta_h(p) = d_{sx}+w_h+d_{yt}$, where d_{sx} is defined as the length of the shortest path from node s to node x and d_{yt} is the length of the shortest path from node y to node t. Note by definition of $\theta_0(p)$, we have $\theta_0(p) - \theta_h(p) \leq 0$ for any link l_h. Furthermore, if $\theta_0(p) - \theta_h(p) < 0$ and we change the link weight for link l_h from w_h to $w_h + \theta_0(p) - \theta_h(p)$, then under the new set of link weights, the shortest path for OD pair p will pass through link l_h. Note also that there may be more than one shortest path for the OD pair p, and in this case all of the traffic for OD pair p is split evenly across all such paths. If link l_h belongs to exactly one of the shortest length paths for OD pair p, then note that $\theta_0(p) - \theta_h(p) = 0$ and that changing the weight of link l_h from w_h to $w_h - 1$ will result in *all* of the traffic for OD pair p to be routed over link l_h. Motivated by these observations, for each l_h and p, we define $\delta w_h(p) = \theta_0(p) - \theta_h(p)$ if $\theta_0(p) - \theta_h(p) < 0$, $\delta w_h(p) = -1$ if $\theta_0(p) - \theta_h(p) = 0$ and link l_h lies along exactly one shortest path for OD pair p, and $\delta w_h(p) = 0$ otherwise. Recalling that any link weight change must result in a new weight that lies in the range $[W_{min}, W_{max}]$, we prune the set of link weight perturbations for link l_h to the set Δ_h, where $\Delta_h = \{\delta w_h(p) : \delta w_h(p) < 0$ and $w_h + \delta w_h(p) \in [W_{min}, W_{max}], \forall p \in \mathcal{P}\}$.

After considering all possible links l_h, we obtain a set of possible snapshots $\Delta = \cup_{h=1}^{L} \Delta_h$, each of which corresponds to a single weight change and results in traffic from an OD pair being moved on at least one link. In the numerical examples we have investigated so far, we have found that $|\Delta| << WL$, so there is a significant savings over considering all possible single weight changes.

As an aside, note that there are multiple ways to generate the set Δ. One possible way is to apply any possible weight on any link. In that case, for any link $l_h \in \mathcal{L}$ we have $W = W_{max} - W_{min}$ possible solutions. Then, $|\Delta| = \sum_{h=1}^{L} |\Delta_h| = WL$. In our approach we attempt to force every OD pair on each possible link. Then, $|\Delta| = \sum_{h=1}^{L} \Delta_h = PL/2$, since the routing used by OD pair $s \rightarrow d$ is the same as the route used by $d \rightarrow s$. As we can see, the first approach generates a number of solutions that is proportional to the cardinality of the IGP weights allowed, while the second generates a number of solutions that is proportional to the cardinality of the OD pairs. We decided to use the second approach for two main reasons. First of all, if we are interested to disaggregate a specific subset of OD pairs, the second method generates far fewer candidate solutions. Secondly, it is anticipated that carriers will eventually increase the number of bits used to encode a link weight from 6 to 8. That means the cardinality of W will increase from 64 up to 256. In this scenario, the difference in the number of candidate solutions generated by the two approaches is even greater, and thus the second approach leads to a lower complexity of the algorithm.

Step 2: Pruning by Performance Constraints.

In this step we do an evaluation of the set of candidate perturbations to see if they meet the *delay-limit* and *load-limit* constraints. We examine all the entries in the set Δ and build the set of candidate snapshots as a set of tuples $S = \{(l_h, w_{hi})\}$ where each tuple represents a deviation from the original weights set by setting the weight of the link l_h to the value w_{hi}, defined as $w_{hi} = w_h + \Delta w_h(p_i)$. For each tuple, we compute the associated routing matrix A_{hi}. For each candidate snapshot, we compute the new average end-to-end delay for all OD pairs. If this violates the *delay-limit* we set aside the snapshot. If it meets the delay constraint and we have an old TM available X_{old}, then we route this TM on the new topology and compute the resulting link loads (this is done simply by computing the matrix product $A_{hi}X_{old}$). If any of the link loads violate the *load-limit*, then this snapshot is also set aside. We record snapshots that are set aside, as well as their routing matrix, in a temporary storage location so that it may be re-used in step six if necessary. For the moment these set aside snapshots are essentially discarded.

We point out that it is possible for different weight perturbations to have the same net impact on the routing. Therefore, for those snapshots that are within both the delay and load limits, we only include them only if the corresponding routing matrix A_{hi} is not identical to that of a routing matrix for another included snapshot.

Step 3: Ordering the candidate snapshots.

Now that we have a set of candidate snapshots, we need to determine which of those snapshots to include to generate our aggregate routing matrix. Conceptually, we could consider all possible subsets of the candidate snapshots, and compute the rank of the associated aggregate routing matrix for each subset of snapshots. Since the number of subsets is very large, this is computationally prohibitive. Instead, in this step we construct an ordering of the candidate snapshots according to a heuristic. The ordering will then be used in Step 4 to build up the set of included snapshots in our aggregate routing matrix.

In order to produce an ordering on the candidate snapshots that remain up to this point, we define a ranking function for each snapshot. The ordering among the snapshots will then be according to the ranking function, with the highest ranking snapshots at the top of the order.

We now motivate and define the ranking function for each snapshot. Our objective is to identify the snapshots which appear the most promising, i.e. the snapshots which result in a large increase in the rank of the aggregate routing matrix. Suppose that a given perturbation moves exactly one OD pair off of some link and onto link l_h. In this case this perturbation determines the traffic volume on that OD pair exactly; it's value is the difference between the link count on l_h before and after the change. Since a single weight change can move multiple OD pairs, it is possible to learn more than one OD pair exactly in one weight change event. Once an OD pair is known exactly, we call it a well-known OD pairs. Our goal here is to construct ranking function which reflects the impact on the number of newly well-known OD pairs, as well as a general measure of the amount of "information" gained in implementing the snapshot.

We introduce some new notation used to define our ranking function. Let Γ_{hi} be the set of OD pairs whose routing is affected by changing the link weight on link l_h to the new value $w_{hi} = w_h + \Delta w_h(p_i)$. Let $\beta_{hi}(p)$ be a binary variable that is equal to 1 if the OD pair p uses the link l_h and 0 otherwise, with respect to the new set of link weights after changing the weight of link l_h to w_{hi} as described above.

Suppose first we want to evaluate our ability to estimate the traffic for OD pair p_t. For a link l_h which belongs to a new link on a path for OD pair p_t after the link weight change, consider the sum $\sum_{p \in \Gamma_{hi}/p_t} \beta_{hi}(p)$. This sum counts all the OD pairs moved onto link l_h other than p_t at the same time that p_t is moved. If this sum is zero, then only p_t has been moved on link l_h. Let $R_{hi}(p_t)$ be the set of new links along the path used by

OD pair p_t after applying the IGP weight w_{hi}, i.e. the set of links that are contained in a shortest path for p_t after changing the link weight w_h to w_{hi} but are not included in any shortest path for p_t for the original set of weights $\mathcal{W}$. We define the *ambiguity* of p_t after the weight change (l_h, w_{hi}) to be

$$m_{hi}(p_t) = min_{l_h \in R_{hi}} \sum_{p \in \Gamma_{hi}/p_t} \beta_{hi}(p) \qquad \forall w_{hi} : (l_h, w_{hi}) \in S, \tag{2}$$

which takes the minimum of these per-link sums across all the new links in the new path of OD pair p_t. Again, if $m_{hi}(p_t) = 0$ then p_t becomes a well known OD pair. This can be viewed as a measure of disaggregation. For example, a value of $m_{hi}(p_t) = 2$ means 3 OD pairs (including p_t) have been disaggregated from their original bundling and moved to link l_h.

The quantity $m_{hi}(p_t)$ is defined for a single OD pair p_t. We now define the following metric defined over all OD pairs affected by changing link l_h to the value w_{hi}.

$$M_{hi} = \sum_{p_t \in \Gamma_{hi}} (1/(Bm_{hi}(p_t) + 1)) \qquad \forall w_{hi} : (l_h, w_{hi}) \in S, \tag{3}$$

where B is a large parameter. Note that if $m_{hi}(p_t) = 0$ for all the p_t affected by the weight change event (l_h, w_{hi}), then M_{hi} is equal to the number of new well-known OD pairs. Note that for all p_t which are not well known as a result of the weight change on link l_h we have $m_{hi}(p_t) \geq 1$ and there are at most P such OD pairs. Thus if B is large enough so that $P/(B + 1) < 1$, then a single newly well known OD pair is weighted more in the metric than all the terms which measure how much information is gained for OD pairs which do not become well known.

Let's consider an example to understand how this metric behaves. Suppose we have two IGP weight changes to compare, w_1 and w_2. The first change w_1 moves only one OD pair (p_1) but in such a way that it can be known exactly, i.e., $m_{w_1}(p_1) = 0$. In this case $M_{w_1} = 1$. The second change moves three OD pairs p_1, p_2 and p_3, but none of them can be completely isolated from the others. If we suppose that each link is shared by two OD pairs that were rerouted, then $m_{w_2}(p_1) = m_{w_2}(p_2) = m_{w_2}(p_3) = 1$. Without the B factor, w_2 would produce a value of $M_{w_2} = 1.5$. Because this is bigger than $M_{w_1} = 1$, it would make snapshot w_2 more attractive than w_1. However, we want the reverse to happen because we consider it better to know something exactly than simply to get new link counts without any complete disaggregation. By using a large B factor, the proposed snapshot w_1 will achieve a higher priority than w_2 since $M_{w2} \ll M_{w1}$.

The snapshots are then ranked according to the corresponding value of M_{hi} for each snapshot, where the snapshot with the largest value of M_{hi} is at the top of the list.

Step 4: Evaluation of candidate single link snapshots.

We now evaluate the candidate snapshots in the order defined in Step 3. This is done by determining whether or not each snapshot increases the rank of the routing matrix obtained so far. Let A_0 denote the initial routing matrix based on the original given set of IGP weights. Let A denote the improved aggregate routing matrix after useful snapshots have been appended. When we are done A will be the final aggregate routing matrix. We evaluate each entry $(l_h, w_{hi}) \in S$ sequentially by appending its associated routing matrix, (A_{hi}) to A and computing the rank of the combined routing matrix. The snapshots are considered in the order found in step 3, with the highest ranking snapshots tried first. Only if the rank of A increases is A_{hi} kept in A; otherwise the snapshot is discarded. The algorithm stops when the rank of the A matrix is equal to P. If all the entries of S (i.e., the candidate snapshots) are analyzed and the rank of A is less than P, then we proceed to step 5.

Step 5: Multiple IGP weight changes. For large networks performing only single weight changes might not be enough to guarantee that we can obtain a full rank routing matrix A. For example, this is the case for the Sprint backbone. In this step we consider changing either two or three link weights simultaneously, i.e., within the same snapshot. We limit the maximum number of simultaneous changes because we believe that carriers would not be comfortable changing more than three link weights at the same time.

More specifically, snapshots corresponding to weight changes on pairs of adjacent links are considered. A pair of links is defined to be adjacent if they are incident to a common node. For each pair of adjacent links, we consider only two weight changes, i.e. changing the weights of both links to W_{min} and changing the weights of both links to W_{max}. For each such snapshot, we check to see if the the delay-limit and load-limit constraints are satisfied. If so, we try augmenting the corresponding routing matrix to the aggregate routing matrix to see if it increases its rank. If so, we add the snapshot to the included list and update the aggregate routing matrix accordingly. If the aggregate routing matrix is still not of full rank, we consider another snapshot generated by another pair of link weight changes. The process continues until the aggregate routing matrix has full rank or we exhaust all snapshots corresponding to such pairs of link weight changes.

If we exhaust all snapshots corresponding to a pair of link weight changes as described above before the rank of the aggregate routing matrix reaches P, then we try snapshots corresponding to triplets of links having their weights changed. We consider only triplets of links that form "triangles", i.e. triplets of links of the form $\{(a, b), (b, c), (c, a)\}$. For each triplet of links we only consider two possible weight changes, i.e. changing the weights of all three links to W_{min} and changing the weights of all three links to W_{max}. We use "triangles" because this approach will move a batch of traffic either away from (when link weights are increased) a localized region of the network, or onto that region (when link weights are decreased). For each such snapshot, we check to see if the the delay-limit and load-limit constraints are satisfied. If so, we try augmenting the corresponding routing matrix to the aggregate routing matrix to see if it increases its rank. If so, we add the snapshot to the included list and update the aggregate routing matrix accordingly. If the aggregate routing matrix is still not of full rank, we consider another snapshot generated by a triplet of link weight changes. The process continues until the aggregate routing matrix has full rank or we exhaust all snapshots corresponding to such triplets of link weight changes.

Step 6: Relaxing Performance Constraints. If we reach this point and we still do not have an aggregate routing matrix of full rank, then we relax the delay and load constraints. We remove the delay constraint altogether since it is only an average constraint, and we shift the load constraint up to 90%. Clearly carriers cannot go above this or congestion would occur. The set of candidate perturbations we consider at this point are those that were set aside earlier. Recall that this set includes perturbations that passed the pruning of step 1, but failed the pruning of step 2. With this set, the algorithm runs again from step 2 (with the modified performance constraints) through step 4.

V. Results

In this section we assess the performance of our heuristic algorithm by running it on two large Tier-1 backbones. We see which snapshots are chosen and examine their properties. We illustrate the trade-offs between the number of snapshots and the delay and load constraints. The benefit of doing multiple link changes simultaneously will be discussed. We also show the impact of such an activity on the delay and load of the network and discuss the ability of our technique to meet carrier demands. Before providing these results, we explain the topologies and traffic matrices used in our evaluation scenarios.

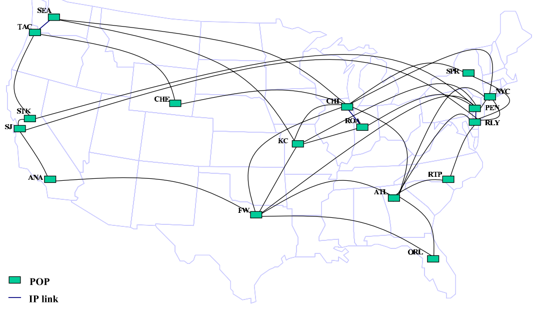

Fig. 2. Sprint North America IP Backbone

Most of the results refer to the Sprint backbone as shown in Fig. 2. We also considered a second backbone (a topology provided by the Mapnet Tool [8]) with a larger number of nodes. Both the topologies represent a simplification of the real backbones operating in the USA. The Sprint topology consists of $N = 17$ Point-of-Presence (PoPs), and $L = 34$ inter-PoP bidirectional links. The number of OD pairs is $P = N(N-1) = 272$. The second topology consists of $N = 25$ PoPs and $L = 41$ inter-PoP bidirectional links, with $P = N(N-1) = 600$ OD pairs.

A 3-tuple is associated with each link: the original *ISIS weight*, the *propagation delay* and the *capacity*. In the actual Sprint network, two neighboring PoPs can be interconnected with multiple links between them. The topology studied, shown in Fig. 2, is obtained by aggregating all the links between neighboring PoPs into one big pipe. Most of these parallel links between neighboring PoPs have the same ISIS weight assignment. The *new big link* is characterized by the same ISIS weight, by a propagation delay equal to the maximum propagation delay among them, and by a capacity equal to the sum of the available capacity of each single link. We do not have the same set of data for the second backbone. To figure out how many OC48 links are present between each pair of neighboring PoPs, we used the information provided by the Mapnet Tool [8]. We calculate the propagation delay of each link by mapping each link along the shortest physical path. Using link speed information and the physical distance we are able to estimate its propagation delay. Later on we explain how we generated an "original" set of ISIS weights for the second topology, i.e., the ones selected for a good daily operating condition.

As mentioned before, we cannot evaluate the impact on the load without a traffic matrix itself. For the Sprint backbone, we used a traffic matrix estimated using the existing choice-model method from [3]. We estimated the traffic matrix for October 10th, 2002 during a *busy time*, from 12pm to 1pm, by using the SNMP data from that time. As discussed previously, we view this as an old traffic matrix with imperfections. Even if we had a real matrix that was old it may contain imperfections as it could be out of date. We realize that imperfections in this TM will propagate to the aggregate load we compute and this hinders our ability to evaluate the impact of our proposed weight changes on the network performance. We remind the reader that carriers can choose to run the algorithm with the delay-limit input only and avoid the load evaluation. However we believe that carriers would prefer to do some load evaluation rather than none. Our goal is not to generate exact numbers for load levels, but rather to understand generally whether such constraints are satisfiable, and the trade-offs that load constraints impose.

Since we do not have any data available for the second backbone, we generate a random traffic matrix according to a Poisson distribution with rate 40 Mbps. We have built a tool called METL (IGP MEtric assignment TooL) that finds an optimal set of ISIS/OSPF weights for a given topology and traffic matrix. This tool is based on an algorithm we developed in [10] and finds a set of weights that minimizes the maximum link utilization. We use these weights for our second topology since we assume that any carrier has their own algorithm for finding optimal weight assignments.

In our evaluation of average delays, we assume that propagation delays are the only delays present in the backbone. It has been shown in [9] that in the Internet core, there is no congestion and queuing delays are negligible. For each routing scenario we compute the end-to-end delays for each OD pair, and then average over all the OD pairs to obtain an average network-wide delay. We use this delay metric because this is the definition of the delay component in today's actual SLAs.

A. Basic Properties of Selected Snapsnots

In this first section where we examine the basic properties and trade-offs, we use only the Sprint backbone since our information on this network (topology, link weights and old TM) is more accurate. We set the delay-limit to be 60ms. We consider three different load-limit thresholds, namely 60%, 70% and 80%. In Figure 3 we illustrate how the rank increases as we add more snapshots. First we see that with no additional snapshots (the zero case on the x-axis) we have only 25% of the full rank. Recall that this represents the conditions under which previous traffic matrix estimation methods operate - indeed a severely underconstrained system. With the extra information provided by additional well chosen snapshots, we see that the rank rises steadily. Because our snapshots are carefully selected, each one should increase the rank by at least one. Since these curves have no flat sections, we confirm that our algorithm is working properly in that snapshots are doing what they are suppose to do. If the load is permitted to reach 80%, then full rank can be obtained with 27 snapshots. With a *load-limit* of 70%, 33 snapshots are needed. If the load constraint is tightened to 60% then we need 47 snapshots. This illustrates our first trade-off. The more constrained the load-limit is, the more snapshots are needed to reach full rank. This makes intuitive sense. The looser the constraint (i.e., the higher the load-limit), the more flexibility we have to move OD pairs around.

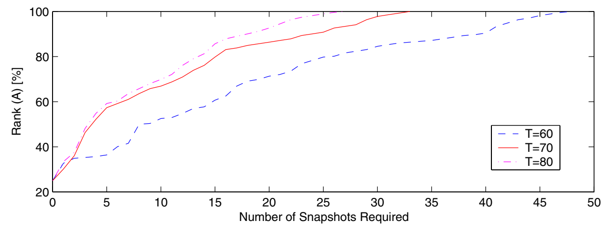

Fig. 3. Tradeoff between the number of snapshots needed and the load constraint.

Recall that our algorithm allows the delay and load constraints to be violated towards the end if we cannot reach full rank under these constraints. In all three cases considered here (for the three values of T) we were unable to reach full rank, and thus allowed violations. Table I gives the number of violations for each of the three load-limit thresholds considered. Here we see our second trade-off: the tighter the load-constraint the more violations that will occur.

load-limit	60%	70%	80%
Num snapsnots	47	33	27
Num violations	24	4	3

TABLE I
NUMBER OF VIOLATIONS FOR DIFFERENT LOAD-LIMIT THRESHOLDS

The two trade-offs together indicate that the lower the load-limit (i.e., the tighter the constraint) the more difficult it is to obtain a full rank matrix. In comparing the 70% load-limit and 80% load-limit cases, we see that the 70% limit is worse than the 80% case in that it requires 6 more snapshots and has 1 more violation. However limiting the load to 70%, rather than 80%, is much more appealing to carriers. In the rest of this section we use the 70% load limit as we believe this is a reasonable trade-off between the load and the number of snapshots/violations. Clearly the 60% load-limit is not attractive as it has such a large number of violations and requires almost 1/3 more snapshots than the 70% case.

B. *The Order of Selected Snapshots*

In the previous subsection we saw that 33 snapshots were needed (with a delay-limit of 60ms, and a load-limit of 70%). Recall that our algorithm first selects single-link weight changes. Only after single link changes have been analyzed, double and triple link changes are considered. Thus there is an implied order of the snapshots on the x-axis in Figure 3, namely that the single-link changes are done first, then the double link changes and finally the triple link changes. It is intuitive that if we carried out the multiple link changes before the single ones, we might need fewer overall snapshots. This is because the double and triple weight changes are more dramatic in that they move more traffic around. Hence we are likely to obtain more linearly independent equations into our system than we do from the single weight changes, and similarly we force more OD pairs to become “well-known” using a multi-link weight change rather than a single link weight change.

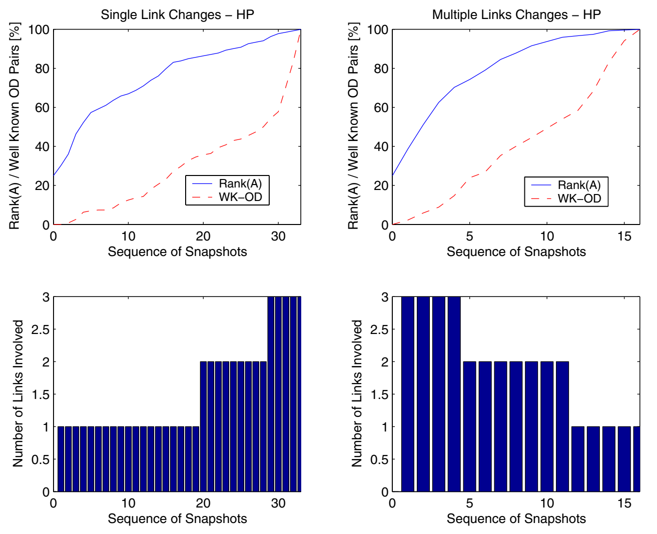

Fig. 4. Rank(A) and Well-Known OD Pair versus Number of ISIS weight changes.

Recall the metric we used for ordering snapshots in Step 3 of our algorithm. This ordering metric is applied only to single snapshots and thus should give us the best ordering for those snapshots. However it does not quantify the prefered ordering of the snapshots when the double and triple link changes are included. To quantify this intuition we now look at two orderings, the original one and a second one in which the original order is exactly reversed. In other words, in the second ordering, triple link changes are done first, then double link changes and finally single link changes. Results for these two orderings are given in Figure 4. In the top-left figure, we plot the achieved rank (as a percentage of the total rank) as a function of the number of snapshots when they are added in the original ordering. The rank is given by the solid line. Note that as we increase the rank, we will also increase the number of OD pairs that are known exactly. The number of OD pairs, that are known exactly once a given number of snapshots have been added, is indicated in this figure by the dashed line. We see that a good deal is learned with the last few snapshots, indicating the utility of the triple-link changes. In the bottom left of this figure, we see that the first 19 snapshots are single-link changes, the next 9 are double link changes and the last 5 are triple link changes.

In the top-right of this figure we plot the rank and number of well-known OD pairs for the reverse order. We find that only 16 snapshots are needed to achieve full rank. Indeed the order has a large impact. We now need only 4 triple-link changes, 7 double-link changes and 5 single-link changes. Note that the number of triple-link changes (and double-link) is not the same in both cases. When we reversed the order we found that we could eliminate one of the triple-link changes. This behavior is technically sound under our approach. Recall that snapshots are evaluated in a given order and retained if they increase the rank. In the original order each snapshot increased the rank, but since we do not consider all possible permutations of the triple-link changes, we cannot know at the time we evaluate one snapshot whether another snapshot coming up soon in the order would have provided the same information. When we evaluated the triple-link changes in reverse order, one of the triple-link changes was eliminated in Step 4 of our heuristic because it would have added redundant information. The same happened for the double-link changes.

C. *Impact on Delay Performance*

We now examine the impact of all these link weight changes on the delays experienced between PoP pairs. In order to examine the impact of each snapshot on each OD pair, we use the scatterplot in Figure 5. Each dot indicates that the end-to-end delay for that OD pair under the given snapshot was under 80ms. A star indicates that the delay was between 80 - 120 ms, and a circle indicates that the delay was between 120 - 160 ms. This scatter plot indicates that 6 snapshots caused some OD pairs to have delays larger than 120ms. However, among those 6 snapshots, 3 of them affected two OD pairs or less. In fact we see that there is only one *bad* snapshot in that it affected a large number of OD pairs; namely that is snapshot #1. Most OD pairs that may reach larger delays experience that delay only in one snapshot. There are only 3 OD pairs whose delay will rise above 120ms in multiple snapshots. Note that these values are end-to-end delays. However carrier's SLA's today are given in terms of average end-to-end delay. We had used a 60 ms target for our *delay-limit* in our algorithm. In Figure 6 we see that indeed the average delay, or SLA, value is achieved for each snapshot. We thus conclude that in the scenario studied here, the delay constraints imposed by carriers are achievable.

D. *Impact on Load Performance*

We now illustrate the impact of our snapshots on link load levels. We use another scatter plot, in Figure 7, to show the utilization level of the links under each snapshot. In this plot, a dot indicates that the load level was under 60%; a star indicates that the load was between 60-70%, and a circle indicates that the load was between 70-90%. We observe that under the initial set of link weights (i.e., snapshot 0), there were two links loaded

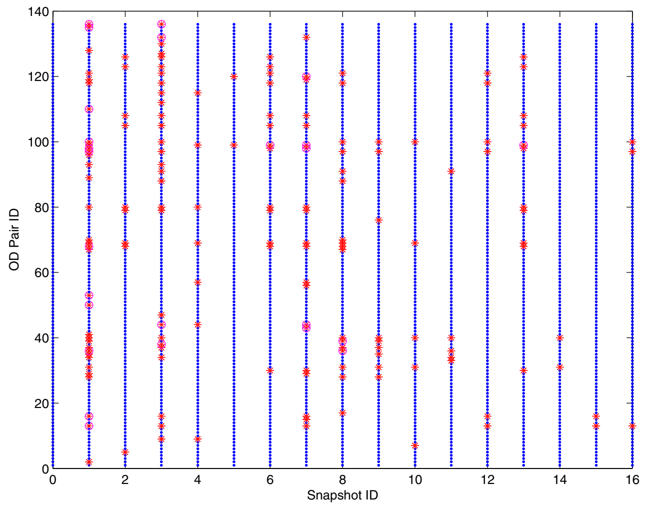

Fig. 5. Delay Scatter Plot for Sprint Backbone.

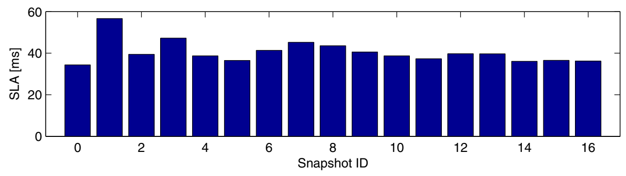

Fig. 6. Average Delay SLA for Selected Snapshots.

between 60-70%. We see that these same links stay in this load range during many snapshots, but never pass into the above 70% load range. This indicates that our heuristic manages to avoid further loading links that are already heavily loaded. We see in this scatter plot, that there are essentially 3 bad snapshots because all of the circles (over 70% load) are contained in those 3 snapshots. In each of those snapshots, there are exactly 4 links and these turn out to be two bidirectional links in all cases. Hence there are two links affected (although in both directions) poorly by these so-called *bad* snapshots. All other snapshots meet the 70% *load-limit* threshold we imposed. Note that these bad snapshots are among the triple-link change snapshots. This is not surprising as multiple-link weight changes are likely to shift more load around.

From these observations we conclude that it could be possible to follow our approach and meet the delay constraints imposed by carriers, but that meeting the load constraints can be a challenge if the end goal is full rank. A carrier could decide to execute all the snapshots except the bad ones. This would significantly increase the rank of the original linear system, but not achieve full rank. This could then be followed by one of the previously existing inference techniques such as either of [3], [2], [4], [5]. Any of these techniques would perform much better because they would have a system that is much much less "underconstrainted" than previously. Moreover a large number of the OD pairs would be exactly known and thus the number of variables to estimate would be substantially less. We checked what happens if all the snapshots except the three bad ones were used, and found that the rank would be at roughly

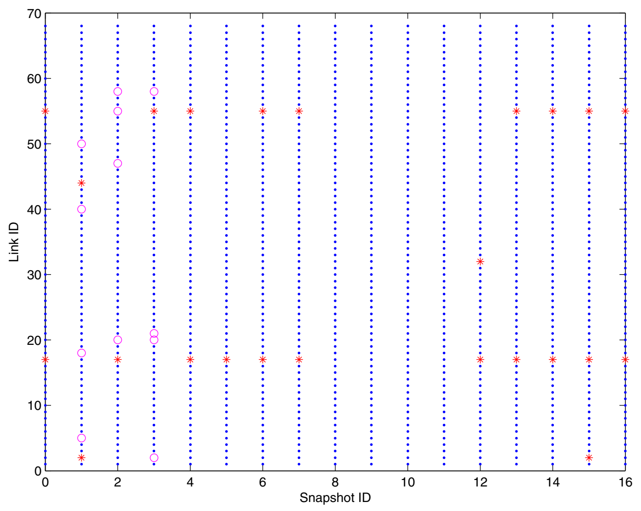

Fig. 7. Link Utilization Scatter Plot for Sprint Backbone.

90% and about 55% of the OD pairs would already be known exactly. Recall that our original system started with a rank of 25%. We thus believe that the best solution for traffic matrix estimation is likely to be a hybrid solution, involving an algorithm such as ours (to increase the rank) as a first step, and an inference technique for estimation as a second step.

E. Comparing Multi-Link Weight Change Heuristics

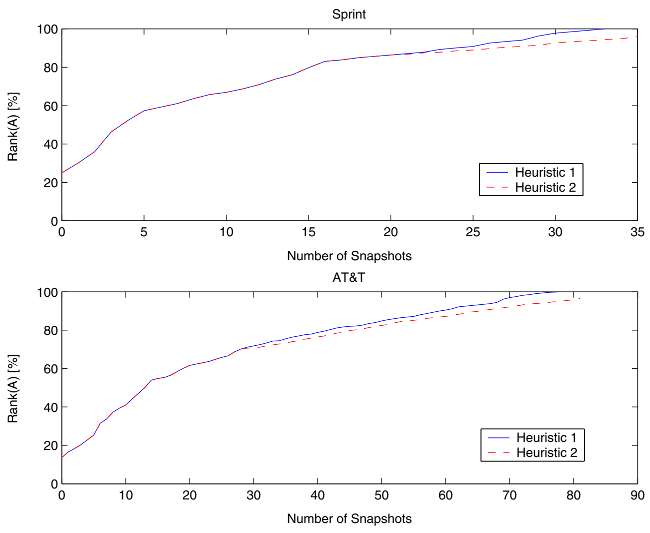

Fig. 8. Impact of topology characteristics and *multi-link* heuristic selection on the number of snapshots required.

In this section we compare two different heuristics for the selection of weight changes involving multiple links simultaneously. As described in Section IV, our heuristic (called *Heuristic 1*) considers only changing the weights on pairs and triplets of links that are consecutive, i.e,, links that have a node in common. To these links, either they are all assigned the minimum allowable weight or the maximum allowable weight. This will either force a good deal of traffic to be moved onto a specific region of the network, or force a lot of traffic away from the region. This should cause sufficient traffic shifting so as to create meaningful new inputs. For the sake of comparison we also consider another heuristic. In *Heuristic 2* pairs and triplets of links are selected randomly among all network links. One weight is then selected randomly within the range $[W_{min}, W_{max}]$ and assigned to either both (or all three) links.

The comparison between the two heuristics is given in Fig 8 where the rank of the A routing matrix is plotted as a function of the the number of snapshots selected. Again we use an average delay bound of 60ms and a maximum link utilization of 70% for testing these two heuristics. The results for the Sprint backbone are given in the top graph while those for the second backbone are in the bottom graph.

In both topologies, Heuristic 1 achieves full rank before Heuristic 2 does. In fact, Heuristic 2 does not achieve full rank. We let heuristic 2 run long enough so as to search through more than twice the search space (in terms of the number of snapshots evaluated). After this, Heuristic 2 was not able to find a set of snapshots that would achieve full rank.

The two curves, for a given network, behave the same for a while (e.g., until snapshot #22 in Sprint network) because until this split we are considering single-link changes. Clearly these curves should overlap because our two heuristics only differ with respect to double and triple link weight changes.

Another interesting observation to make here is that the Sprint network requires roughly half of the snapshots needed by larger topology. We have not quantified the reduction on the number of snapshots when triple and double link weight changes are carried out first. Nevertheless, the number of snapshots needed to bring a network's routing matrix to full rank, is likely to be due to the characteristics of the two topologies, such as the number of nodes and links. The number of snapshots may be proportional to the number of OD pairs to estimate.

We include the example of this larger network primarily for the purposes of illustrating that our algorithm (since it is a heuristic) can and does achieve a full rank routing matrix even in this larger topology. Some simpler heuristics we originally tried could get close but not actually achieve full rank as they would effectively get stuck. Although the number of needed snapshots may become too large to be practical in very large topologies, our algorithm can still be helpful because it can be used inside a hybrid method (as mentioned earlier). Alternatively, if a carrier knew that some OD flows are more important to estimate accurately then others (e.g., large flows as opposed to small ones), then we only need to increase the rank enough so that the desired OD pairs become well known. This would probably need far fewer snapsnots than a full rank target as it seems to be that getting the rank from 90% up to 100% is the difficult part requiring many snapshots. We leave the exploration of these ideas for future work.

VI. CONCLUSIONS

In this paper, we have presented a methodology to construct a collection of "snapshots" for an IP network, where each snapshot is described by a set of link weights that implicitly determines a routing configuration using an IGP routing protocol.

Our method attempts to minimize the number of snapshots, while inducing routing configurations which allow the traffic matrix to be accurately inferred from the measurements of traffic volume on each link. In addition, our snapshots are chosen so that resulting link loads are not excessive and propagation delays are acceptable.

For a carrier backbone such as Sprint's, our method produced under 20 snapshots that meet the link load and propagation delay constraints and result in an aggregate routing matrix of full rank. This number of snapshots seems within the limits of practical realizability. In general, the number of snapshots required is dependent on the network size and topology.

We note that the goal here of obtaining an aggregate routing matrix of full rank may not be necessary. We illustrated that with an algorithm such as ours we can get close to full rank, but not obtain it completely without difficulty with respect to carrier requirements. We believe that the most promising approach for traffic matrix estimation will be a hybrid method. The first step in such an approach would be to use an algorithm like ours to significantly increase the rank of the routing matrix, thereby reducing the underconstrainedness of the problem and identifying many OD pairs exactly. In a second step to estimate the remaining OD pairs, any of the previous inference methods could be applied together with the aggregate routing matrix of high rank. We believe this approach is very promising because it substantially reduces the number of variables (traffic matrix elements) to estimate and greatly reduces the search space of the inference problem.

A direction for future work is to consider more dynamic approaches for synthesis of snapshots. In particular, after link measurements for each snapshot are recorded, the uncertainty in the traffic matrix can be reduced. An updated intermediate estimate of the traffic matrix can be used to predict link loads after future potential link weight changes with more confidence. We remark that in general, if the overall methodology is applied, the inherent risk of overloading links is reduced over time, due to the increased knowledge gained about traffic patterns.

REFERENCES

[1] Y. Zhang, M. Roughan, N. Duffield and A. Greenberg, "Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads", *Proceedings of ACM Sigmetrics* June 2003.

[2] Y. Zhang, M. Roughan, C. Lund and D. Donoho, "An Information Theoretic Approach to Traffic Matrix Estimation", *Proceedings of ACM Sigcomm* August 2003.

[3] A.Medina, N.Taft, K.Salamatian, S.Bhattacharyya and C.Diot, "Traffic Matrix Estimation: Existing Techniques Compared and New Directions", *ACM Sigcomm*. Pittsburgh, PA, August 2002.

[4] J.Cao, D.Davis, S.Vander Weil, and B.Yu, "Time-Varying Network Tomography", *Journal of the the American Statistical Association*, 95(452), 2000.

[5] Y.Vardi, "Estimating Source-Destination Traffic Intensities from Link Data", *Journal of the the American Statistical Association*, 91(433), March 1996.

[6] C.Tebaldi and M.West, "Bayesian Inference of Network Traffic Using Link Count Data", *Journal of the the American Statistical Association*, 93(442), June 1998.

[7] A. Nucci, R. Cruz, N. Taft and C. Diot, "Traffic Matrix Estimation in a Dynamic Enviroment", *Sprint ATL Research Report RR03-ATL-070102*, Sprint ATL, July 2003.

[8] `http://www.caida.org/tools/visualization/ mapnet/Backbones/`.

[9] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran and C. Diot, "Measurement and Analysis of Single-Hop Delay on an IP Backbone Network." *IEEE Journal on Selected Areas in Communications. Special Issue on Internet and WWW Measurement, Mapping, and Modeling*. 3rd quarter, 2003.

[10] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft and C. Diot, "IGP Link Weight Assignment for Transient Link Failures", *18th International Teletraffic Congress*. Berlin, Germany. August 2003.