Computing the Unmeasured: An Algebraic Approach to Internet Mapping

Yuval Shavitt[1]
Bell Labs,
101 Crawfords Corner Rd.,
Holmdel, NJ 07733, USA.
`shavitt@ieee.org`

Xiaodong Sun[2]
Math Dept., Rutgers Univ.,
110 Frelinghuysen Road,
Piscataway, NJ 08854, USA.
`sunxd@math.rutgers.edu`

Avishai Wool   Bülent Yener
Bell Labs,
700 Mountain Ave.,
Murray Hill, NJ 07974, USA.
`yash@acm.org`   `yener@bell-labs.com`

Abstract— **Distance estimation is important to many Internet applications, most notably for a WWW client that needs to select a server among several potential candidates. Current approaches to distance (i.e., time delay) estimation in the Internet are based on placing Tracer stations in key locations and conducting measurements between them. The Tracers construct an approximated map of the Internet after processing the information obtained from these measurements.**

This work presents a novel algorithm, based on Algebraic tools, that computes *additional* distances, which are not explicitly measured. As such, the algorithm extracts more information from the same amount of measurement data.

Our algorithm has several practical impacts. First, it can reduce the number of Tracers and measurements without sacrificing information. Second, our algorithm is able to compute distance estimates between locations where Tracers cannot be placed. This is especially important when unidirectional measurements are conducted, since such measurements require specialized equipment which cannot be placed everywhere.

To evaluate the algorithm's performance, we tested it both on randomly generated topologies and on real Internet measurements. Our results show that the algorithm computes up to 50-200% additional distances beyond the basic Tracer-to-Tracer measurements.

I. Introduction

A. Background

The Internet is growing at a remarkable rate. However, there is no central registry that allows users or planners to track this growth. The basic characteristics of the Internet structure are only starting to be revealed [7], [5], [4]. Learning the exact structure of the network seems to be an unrealistic target.

In many cases, however, an estimate of the distances between nodes in the network is good enough. The most obvious case is when a client needs to select a service from one of several servers located in distant locations. The WWW is an example of such a situation, as more and more popular sites open mirror sites that are geographically scattered. Mirror sites are used to balance the computation load among the servers, and also to reduce the average response time to the clients. Other applications where distance estimation is useful for intelligent server selection include: distributed object repositories, hierarchical caching, etc.

IDMaps [8] is a project that attempts to solve this problem by placing measurement stations (Tracers) at key locations in the Internet. These Tracers periodically measure the distances among themselves and to other regions of the network. The Tracers advertise their measurement information to clients such as SONAR [12] or HOPS [8] servers, and these servers use the measurements to construct an estimated distance map of the network.

A distance measurement between two IDMaps Tracers can be done in two basic ways. The first is to use the `traceroute` program, which generates a series of packets with increasing TTL values. The TTL field in the IP header is decreased by one in every router that handles the packet. When the packet travels a distance equal to its initial TTL, its TTL value is decreased to zero, and the packet is dropped. When the packet is dropped, an ICMP packet is sent from the dropping node to the packet originator. By this repeating this process until the packets reach the destination, the source node learns the round-trip time (rtt), and also the IP addresses of all the nodes on the route to the destination. A second way to measure distance between Tracers is by using `ping`. In this method, a packet is sent to the destination, which replies immediately, resulting in a single rtt measurement.

It is important to note that `traceroute` consumes more network resources than `ping`: the number of message$\times$hops of `traceroute` is quadratic in the hop distance, while `ping` uses a linear number of message$\times$hops (a single packet on every hop in each direction). Due to this, and other reasons which are beyond the scope of this paper, `ping` is a preferred measurement method. However, since the route stability between two Tracers is an important characteristic, `traceroute` is also expected to be used but at a much lower rate than `ping`.

B. Contributions

In this paper, we present an algorithm that increases the effectiveness of end-to-end distance measurements. Given end-to-end distance measurements (e.g., using `ping`) and the routes along which the measurements where conducted (e.g., using `traceroute`) our algorithm computes *additional* distances, to intermediate nodes. The result is a more detailed network map, and thus provides better distance estimation. Furthermore, our algorithm can compute delay estimates between sites in which it is not possible to place Tracers.

Another area where our results can be used is in unidirectional measurement of properties [10], [3]. To measure unidirectional delay, special hardware devices are placed in the network. These devices are synchronized to achieve the required measurement accuracy (e.g., by GPS [10]). Applying our algorithm to the unidirectional measurements produces additional unidirectional distances between nodes in the networks where such special devices are not (or even cannot be) placed.

The main idea behind our approach is that using the mea-

[1] Also with the Dept. of Electrical Engineering - Systems, Tel-Aviv University, Tel-Aviv 69978, Israel.

[2] Research supported by NSF grant CCR-9700239, by DIMACS, and by NSF STC 91-19999.

surement routes, one can identify nodes through which routes between several Tracers pass. We refer to these nodes as *crossing points*. A favorable arrangement of these points may enable us to calculate the distances to the crossing points and between them from the end-to-end measurements. In the worst case, the number of crossing points can be zero or they can be arranged in a way where no additional delay can be calculated. However, we show that in the Internet this is not the case, using both simulated networks and recent Internet traces (from Oct. 1999).

Our algorithm is adapted to handle noisy measurements using least-squares approximation algorithm (cf. [6, Ch. 31]).

To evaluate our algorithm, we studied randomly generated networks of two varieties, and recently collected data from the Internet. In both cases, our algorithm succeeded to compute a significant number of distances between Tracers and crossing points, and between different crossing points. For the Internet data, the algorithm discovered additional distances to as many crossing point as the original Tracers, a 100% gain. For randomly generated networks, the gain was over 200% when we used the Waxman method [17], and about 50% when we used networks generated according to the recently discovered power-law on the node connectivity [7]

C. Related work

Francis et al. described the IDMaps architecture and philosophy of operation in [8]. This paper did not focus on optimizing the measurement overhead, and made only some simple observations about situations in which measurements can be saved. Theilmann and Rothermel [16] suggested to use hierarchical Tracer structure to reduce the measurement overhead. A better approach was suggested by Jamin et al. [9] that showed how spanners [14] can effectively reduce the amount of measurement without sacrificing too much of the estimation effectiveness. Our algorithm complements all of these approaches, by extracting more information from the same data that is collected by the various tracing strategies.

The MINC project (cf. [11]) aims at using multicast inference to characterize network loss and queueing delay (rather than propagation delay). To the best of our knowledge, all their published results are for a single multicast tree, where they succeed, like us, in calculating non measured parameters (queueing delay and loss).

Organization The rest of the paper is organized as follows. In the next Section II we present the model and a simple example of the idea behind the algorithm. In Section III, we show that on a network with a tree topology one can compute the distances between all the crossing points. In Section IV we present the details of algorithm itself. In Sections V and VI we describe the evaluation of the algorithm using Internet data, and using synthetic data. We conclude with Section VII.

II. The model

A. Definitions

For simplicity of presentation, the network is modeled as an undirected graph. The directed case is similar, and is discussed in Section IV-A. The graph structure or size is unknown to the algorithm, and is used only for the purpose of analysis. We assume that measurement stations (Tracers) are placed at some nodes of the graph. The routes between Tracers are assumed to be quasi-static, i.e., they change slowly enough to make their knowledge valuable. On the other hand, the distance between nodes is assumed to be dynamic. The distance may be the propagation delay [9], [8], the average delay [16], hop count, or any other measurable route characteristics. Since delay is the most commonly pursued characteristic, we interchangeably use the terms distance, length, and delay.

We make no assumptions about the routing in the network. The algorithm is easier to explain when the routing is symmetric (and we assume this for the generated networks), but it works just as well for asymmetric routing.

Definition 1: A **measurement path** is the route (list of nodes) between two different Tracers as defined by the network topology and the underlying routing protocol.[1]

Definition 2: The *measurement graph* is the union of all the measurement paths. I.e., the graph nodes are the union of all the nodes along the measurement path, and the graph edges are the union of all the links comprising the measurement path.

Definition 3: A non-Tracer node that has at least three different neighbors in the measurement graph is called a **crossing point**.

Definition 4: A **segment** is a maximal sub-path of a measurement path, whose end-points are either Tracers or crossing points, that does not include an internal crossing point.

Definition 5: The **segment graph** is a graph whose nodes are the Tracers and crossing points, and has a link between two nodes if there is a segment between these two nodes in the measurement graph.

The problem: Given a set of end-to-end delays between Tracers with their associated routes, find all the possible segments or groups of consecutive segments whose length can be derived from the data.

B. An example

The following simple example explains the terms defined above, and the problem statement. For simplicity, we assume that the routing is symmetric, and that the measurements are for the round trip delay. Thus all the delays are expressed as round trip times.

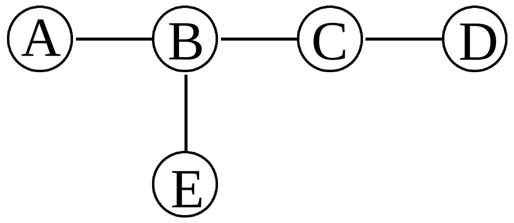

Fig. 1. A five node network example.

Consider the five node network of Figure 1, where Tracers are placed at nodes A, D, and E. Suppose that the following three (round trip) distances are measured: A-D, E-D, and A-E. Using `traceroute` the three routes, A-B-C-D, E-B-C-D, and A-B-E, are obtained.

[1] In the unidirectional case a measurement path between Tracer A and Tracer B is simply the route from A to B. For the bidirectional case, it is the concatenation of the two unidirectional paths between A and B.

Note that it is clearly impossible to compute the distance on the link B-C separately from the distance on the link C-D, since every end-to-end measurement path that contains one of these links also contains the other. This is the motivation for the definitions of segments and crossing points.

In the example, only node B is identified as a crossing point. This defines three segments: s_1=A-B, s_2=B-C-D, and s_3=E-B. Suppose that, using `ping`, the distances A-D, E-D, and A-E were measured to be 4, 7, and 5, respectively.

The following three equations express the `ping` measurement data, using the segments identified from the `traceroute` information as variables:

$$\begin{array}{ccccccl} s_1 & + & s_2 & & & = 4 \\ & & s_2 & + & s_3 & = 7 \\ s_1 & & & + & s_3 & = 5 \end{array} \qquad (1)$$

In this case, we have three linearly independent equations with three variables, which we can solve to obtain the delay in each of the three segments: $s_1 = 1, s_2 = 3, s_3 = 4$. Thus, we are able to compute all the distances to the crossing point B, even though no Tracer was placed in it. The gain for this example is 100%: from 3 measurements we were able to compute 3 additional distances, and discover distances to one additional non-Tracer node (33%).

III. The Tree Case

Suppose initially that each Tracer measures the distances to all other Tracers (we will show later that this assumption is stronger than necessary). Suppose, further, that the resulting measurement graph is a tree, and let t denote the number of Tracers. We prove that in this case, one can find the delay on all the segments using a simple linear algorithm. For simplicity, we assume that there is no noise in the measurements. The noise is easily treated by using least-squares approximation to obtain a solution that is the closest to all measurement points (more on this issue is Section IV).

We first note that, with no loss of generality, all the Tracers can be assumed to be placed in leaves of the tree, and not in internal nodes. Otherwise, the tree can be cut at the internal node which is a Tracer (with this node duplicated to all the resulting sub-trees) and each sub-tree can be treated independently. An internal node in the tree with degree greater than 2 is a cross point, and must be part of, at least, two measurement routes by its definition. However, the existence of two routes indicates that, at least, a third route passes through the internal node, as stated in the following lemma.

Lemma 6: The number of measurement routes passing through a crossing point is at least three in a measurement graph with tree topology.

Proof: Consider a crossing point c. By its definition there are, at least, three sub-trees connected to it. Let l_1, l_2, and l_3 be three leaves each in a different sub tree. Obviously the routes between any pair of these leaves must pass through c. ■

Fact 7: If the measurement graph is a tree, then the segment graph derived from the measurements is also a tree, which we call the segment tree. By definition of a segment, internal nodes of the segment tree cannot have degree 2.

Fact 8: In every segment tree there exist at least one internal node with degree $d \geq 3$ that is connected directly to, at least, $d-1$ leaves. We refer to such an internal node as an outpost.

Theorem 9: The lengths of all the segments in the segment tree can be computed.

Proof: Consider a crossing point, c, with degree d that is an outpost. By Fact 8, at least one such a crossing point exists, and it is connected directly to $d-1$ leaves $l_2, \ldots, l_d$. Let l_1 be some leaf node in the part of the tree other than $\{l_2, \ldots, l_d, c\}$. The routes between every pair of the leaves $l_1, \ldots, l_d$ passes through c. Let s_i be the length of the route between c and l_i, $i = 1, 2, 3, \ldots, d$; and let the measurements between l_i and l_{i+1} be b_i for $i = 1, 2, 3, \ldots, d-1$, and the measurements between l_1 and l_d be b_d. Obviously we can solve the following linear system and obtain the length of $s_2, s_3, \ldots, s_d$, which are the segments that connect c to the leaves $l_2, l_3, \ldots, l_d$.

$$\begin{array}{cccccccl} s_1 & + & s_2 & & & & & = b_1 \\ & & s_2 & + & s_3 & & & = b_2 \\ & & & & \ddots & & & \\ s_1 & & & & & + & s_d & = b_d \end{array}$$

This way one can obtain the length of all the segments that connect all the leave nodes to the crossing point c.

Removing nodes $l_2, \ldots, l_d$ from the tree does not change the degree of any internal node (except for the outpost) and thus Fact 8 holds for this tree as well, enabling the repetative application of the above procedure.

After the leaves $l_2, \ldots, l_d$ are removed, we are left with a tree with a leaf node (c) which is not a Tracer. However, we can use any of its (removed) leaves, say l_2, as a measurement proxy. The distance between some Tracer z and c can be computed by subtracting the newly computed distance between c and l_2 from the distance between the Tracers z and l_2.

In the final stage of the algorithm we are left with a star, and there all the star segments can be easily calculated with the same equation. ■

Theorem 10: All the segment lengths in the tree can be found using $O(t)$ measurements.

Proof: In the proof of Theorem 9 we used d measurements to obtain the length of every $d-1$ segments iteratively. The binary tree is the case that maximizes the number of measurements we need due to two reasons. First, it maximizes the number of segments in a tree with t leaves, which is $2t$. Second, the binary tree gives us the worst measurement to gain ratio, i.e., $d/(d-1) = 3/2$. Thus, we need no more than $3t$ measurements. ■

Note, of course, that not every set of $O(t)$ measurements is sufficient for finding all the segement lengths.

IV. The Algorithm

In this section we describe our algorithm for general networks. We reiterate that the only information available to the algorithm is the set of end-to-end measurements. We do not make any assumptions about the structure, connectivity, or size of the network. The algorithm comprises of several phases, which we describe in the following sections.

A. Interpreting the measurements

Before the algorithm itself can begin its work, we need to decide how we wish to interpret the measurements. In particular, we need to define our variables, so that we can write equations that correspond to the measurements. For a link (A,B), two choices exist. We can either define two unidirectional variables, one for the delay from A to B and one for the delay from B to A; or we can define a single bidirectional variable, for the round-trip delay A-B-A.

The decision depends on the nature of the measurements available to us. If the measurements are truely unidirectional, then we should clearly use unidirectional variables. If the measurements are round-trip measurements, and the routing is symmetric, then we can use bidirectional variables. For round trip measurements with asymmetric routing, which is the situation most appropriate for `traceroute` and `ping` Internet measurements, we can use either unidirectional or bidirectional variables. Both possibilities have pros and cons. In Section V we describe how we interpreted the measurements in our experimentation.

Another point to consider is the interpretation of the delay values that are measured. The simplest and least informative case is when a measurement value is simply the result of a single `ping`. In this case our algorithm would compute a "snapshot" of the delays in the system. However, as discussed in [8], it is more likely that a set of measurements will be taken between every two Tracers. Then the delay value that appears on the right-hand-side of our equations can be either the average delay in the set, or the minimal delay in the set (the latter is appropriate when we are trying to estimate the propagation delay and to ignore the queueing delays). Our algorithm is essentially indifferent to the meaning of the delay value, however this issue has some implications when dealing with noisy data (see Section IV-D).

B. Segmentation, and writing the equations

Once we decided upon the definitions of our basic variables, in principle, we can write a linear system of equations that describes the measurements. The left-hand-side of each equation is the sum of all the variables (uni- or bidirectional) corresponding to the links that appear in a particular measurement path. The right-hand-side of each equation is the measured (unidirectional or round-trip) delay for this path.

However, as we remarked in the discussion of the example in Section II-B, there are variables that clearly cannot be solved. Thus, we need to switch from dealing with individual links to dealing with segments (recall Definition 4). For this, we need to identify all the crossing points (Definition 3). Once we identify the crossing points, we define our variables *per segment*, and write the equations in terms of these segment variables. As we discussed in the previous section, the segment variables can be either uni- or bidirectional.

Notation: Let n denote the number of measurements, and let m denote the number of segments that remain after the crossing points have been identified. We use x_j for $j = 1, \ldots, m$ to denote the variables representing the lengths of the m segments, and b_i for $i = 1, \ldots, n$ to denote the lengths of the given measurement. Let a_{ij} for $i = 1, \ldots, n$ and $j = 1, \ldots, m$ be coefficients such that $a_{ij} = 1$ if the jth segment appears on the ith measurememt path, and $a_{ij} = 0$ otherwise. The general form of the equations obtained after segmantation is as follows:

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & + \cdots + & a_{1m}x_m & = b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + \cdots + & a_{2m}x_m & = b_2 \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + \cdots + & a_{nm}x_m & = b_n \end{array} \tag{2}$$

Let $A = \{a_{ij}\}$ be the $n \times m$ matrix induced by the equations, let $\mathbf{x} = (x_1, \ldots, x_m)$ be the vector of variables, and let $\mathbf{b} = (b_1, \ldots, b_n)$ be the vector of measurements. Then we can rewrite Equation (2) in matrix form as

$$A\mathbf{x} = \mathbf{b}. \tag{3}$$

C. Solving as much as possible

It is highly unlikely that the linear system of Equation (3) is solvable. Typically, it is under-defined for some variables and over-defined for others. Our goal is to extract as much information as possible from the given measurements (we show no one can do better in Section IV-F). Therefore, rather than trying (and failing) to solve Equation (3), we transform the system of equations into a new system that isolates all that is solvable.

The transformation is performed as follows. We perform Gauss-elimination steps on the *columns* of A, until we to transform A into the matrix A' of the following form:

$$A' = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ * & 1 & \cdots & 0 & 0 & \cdots & 0 \\ * & * & \ddots & 0 & 0 & \cdots & 0 \\ * & * & * & 1 & 0 & \cdots & 0 \\ * & * & * & * & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & * & 0 & \cdots & 0 \end{pmatrix}$$

('$*$' means 'any number'). Note that the rows of A may also need to be permuted to reach this structure. This is equivalent to reordering the measurements. Let $m' \le m$ denote the number of non-zero columns in A'. Clearly, the leftmost m' columns of A' are linearly independent.

Note that performing Gauss-elimination on the columns of A is equivalent to multiplying A on the right by a regular $m \times m$ transformation matrix. Let T be this transformation matrix. Furthermore, T can be computed incrementally, as the Gauss elimination progresses, using standard linear algebra. In matrix notation, we have

$$A' = AT. \tag{4}$$

We now define a new vector of unknowns, $\mathbf{y} = (y_1, \ldots, y_m)$ using the same transformation, i.e.,

$$\mathbf{x} = T\mathbf{y}. \tag{5}$$

Then, plugging (4) and (5), and using Equation (3), we can write

$$A'\mathbf{y} = AT\mathbf{y} = A\mathbf{x} = \mathbf{b}. \tag{6}$$

We end with n equations in the y_j variables, that are defined by the matrix A'. Clearly, the y_j's with $1 \le j \le m'$ are solvable from the new equations: The top left $m' \times m'$ sub-matrix of A' is lower triangular and of rank m'. However, these are the only y_j variables that can be solved: columns $m' < j \le m$ in A' are all zero.

D. Dealing with noise

It is highly unlikely that $m' = n$ and that the new system of equations defined by

$$A'\mathbf{y} = \mathbf{b} \tag{7}$$

is solvable. Typically, $m' < n$, and the system of equations is over-defined. In an ideal situation, when the data contains no measurement noise, all n equations would be mutually consistent. In reality, however, noisy data would make the over-defined system unsolvable exactly.

To deal with the measurement noise, we solve (7) for variables $y_j (1 \leq j \leq m')$ using least-squares approximation (cf. [6, Ch. 31]). Using this method we find the values of $y_j (1 \leq j \leq m')$ that minimize the function

$$\sum_i w_i (\sum_j a'_{ij} y_j - b_i)^2, \tag{8}$$

where w_i's are some positive weights. In our implementation we used $w_i = 1$ for all i. Other choices of w_i are also possible, e.g., using $w_i = 1/b_i^2$, but using them would make sense only if we had a more detailed model of the origin of noise. We leave this issue for future work.

Note that the least-squares approximation inherently assumes that the error in the equations may be either positive (measured delay is too high) or negative (measured delay is too low). Whether this assumption is appropriate depends on the meaning of the b_i values (recall Section IV-A). In particular, if b_i is the minimal value selected from a set of measurements, then allowing for negative error may be an invalid choice. In such a case, we can solve (7) using linear programming. Let $\mathbf{e} = (e_1, \ldots, e_n)$ be a vector of error (or slack) variables. We can rewrite (7) as an error minimization problem:

$$\mathrm{M}inimize \max_i \{e_i\} \; s.t. \; \begin{cases} A'\mathbf{y} + \mathbf{e} = \mathbf{b} \,, \\ \mathbf{y} \geq 0, \mathbf{e} \geq 0 \,. \end{cases}$$

This is a linear program, which allows only positive noise. It can be solved using any LP solver. Exploring this method of dealing with noise is also left for future research.

We emphasize, though, that our algorithm introduces no additional errors. This is the case regardless of the method we use for dealing with noise. In an ideal case where measurements contain no noise, solving equations $1, \ldots, m'$ in (7) suffices to compute the exact values of y_j.

E. Back to sub-paths

At this point, we have solved (7), which gives us the values of the y_j variables for $i = 1, \ldots, m'$. However, these y_j's do not directly correspond to lengths we are actually interested in. Recall that our original x_j variables, that represent segment lengths, are related to the y_j's via Equation (5): $\mathbf{x} = T\mathbf{y}$. The elements of matrix T are not necessarily positive or even integral, since it is the byproduct of the Gauss elimination. Therefore, we need to translate the solution from the $\mathbf{y}$ domain back to the $\mathbf{x}$ domain.

It is not immediately obvious how to perform this reverse translation. Clearly, not every segment length x_j can be computed, since only $m' < m$ of the y_j's were solved. However, in many cases we can bypass this problem, using the following observation. Suppose x_j and x_k represent consecutive segments on some measurement path, between A-B and B-C, respectively. Even if we are unable to compute x_j and x_k separately, we may well be able to compute their sum $x_j + x_k$, which represents the delay on the concatenated sub-path A-C. The same observation holds for any sub-path of a measurement path.

Consider a sub-path of one of the measurement paths, which consists of several consecutive segments. Let these segments correspond to variables $x_{j_1}, \ldots, x_{j_\ell}$. Then the delay p on this sub-path can be expressed as 0-1 linear combination of $x_{j_1}, \ldots, x_{j_\ell}$. We can write this combination as a product of a 0-1 row vector $\mathbf{c}$ and the column vector $\mathbf{x}$, where $c_{j_k} = 1$ for $k = 1, \ldots, \ell$ and $c_k = 0$ elsewhere. Formally,

$$p = \mathbf{c} \cdot \mathbf{x} = \sum_{j=1}^{m} c_j x_j .$$

Note that this representation works for individual segments as well: variable x_j can be expressed using a vector $\mathbf{c}$ which is zero everywhere and has a 1 in coordinate j.

We would like to check whether the delay p on the sub-path is solvable. Using Equation (5), we can write

$$p = \mathbf{c} \cdot \mathbf{x} = \mathbf{c}T\mathbf{y} = \sum_{ij} c_i T_{ij} y_j = \sum_j (\sum_i c_i T_{ij}) y_j . \tag{9}$$

Thus, p can be solved if and only if the coefficients of y_j for $j > m'$ are all 0, since these are the y_j variables we were unable to solve. In other words, the delay p on a sub-path can be computed if and only if

$$\sum_i c_i T_{ij} = 0, \;\; \forall j > m'. \tag{10}$$

If the condition in Equation (10) holds, the solution for p is obtained by plugging in the already solved $y_j (1 \leq j \leq m')$ values in Equation (9). In summary, we need to perform the following procedure after solving variables $y_1, \ldots, y_{m'}$ in Equation (7):

```
Procedure **compute-sub-paths**:
  Foreach measurement path $M_i, i = 1, \ldots, n$
    Foreach sub-path $p$ of $M_i$ consisting of segments $x_{j_1}, \ldots, x_{j_\ell}$
      Set $c_{j_k} = 1$ for $k = 1, \ldots, \ell$ and $c_k = 0$ elsewhere.
      If $\sum_i c_i T_{ij} = 0, \forall j > m'$ Then
        Compute $p$ using (9)
      Else $p$ cannot be solved.
```

A detailed example of the algorithm operation appears in [15].

F. Completeness

A distance, by the metric definition (delay), is a linear combination of end-to-end distances. Using linear algebra, since T is non-singular, a sub path can be expressed as linear combination of the given distances if and only if it can be expressed as linear combination of entries of $A'y$. Since the rank of A' is m' and only the first m' columns of A' are nonzero, a sub path can be expressed as linear combination of the given distances if and only if it can be expressed as linear combination of y_i's $(i = 1, \ldots, m')$, which can be solved by our algorithm.

G. The algorithm complexity

Starting with t Tracers one may reveal N nodes. Assuming, that the equivalence list is kept in a hash table, converting the N nodes to their equivalent is a linear process. Using different hash tables one can identify the crossing points and identifying the segments in $O(N)$. Writing the $n \in O(t^2)$ equations is $O(mn)$ where m is the number of segments.

Triangulating the equations with the Gaussian elimination requires $O(nmm')$, where m' is the number of solvable segments. Each column triangulation requires $O(nm)$ operation, and the process stops when no more lines can be triangulated, i.e., after m' iterations.

Checking which of the segments or segment groups are solvable requires less than $O(nm^2)$. Next we give a more precise analysis. On the average, there are $\frac{m}{n}$ segments in a trace. This is a small number since $O(m) = O(t^2) = O(n)$, thus the average cannot be much different from the maximum number of segments per path. Since we check whether any possible consecutive combination of segments in a path can be solved, we perform $O(m_s^2)$ examinations of the condition in Equation (10), where m_s is the number of segments in a path. As a result, the cost of performing this stage is $O\left(\left(\frac{m}{n}\right)^2 \cdot m(m-m')\right)$, per path, and $O\left(n\left(\frac{m}{n}\right)^2 \cdot m(m-m')\right) = O\left(\frac{m^3(m-m')}{n}\right)$ in total.

Calculating the length of a segment or a segment group (using Equation 9) is $O(m')$ per solvable segment, and $O(m'^2)$ in total. If the routing does not change and new delay measurements arrive the complexity of recalculating the delays of all the solvable segments is only $O(m'^2)$.

V. Internet measurements

In this section we describe the experimentation we did with real Internet measurements. We used publically-accessible `traceroute`-ing machines as our Tracers, collected data, and then applied our algorithm to this data.

A. Preliminary issues

A.1 Node identification

When faced with multiple `traceroute`s from different nodes on the Internet, the first thing we need to address is node identification. The output of `traceroute` is normally a list of IP addresses that were encountered along the path between the end-points. However, using these IP addresses directly as node identifiers creates two problems:
1. Routers are multi-homed by definition, so the same router shows up with many different IP addresses in the `traceroute` data, depending on the direction in which the `traceroute` request packet arrived at the router. Typically (but not always) a router will report back the IP address of its interface which is closest to the `traceroute` originator. For our algorithm to give meaningful results, we need to be able to identify all these different IP addresses as belonging to the same node.
2. Many backbone carriers have clusters of routers in their major hubs. A cluster is a collection of several routers, in very close proximity (usually in the same building), connected by a very fast network (e.g., an FDDI ring or ATM mesh). From our perspective, every individual router in the cluster may show up in the `traceroute` data, with its (many) IP addresses, and often consecutive `traceroute`s between the same end-points go through different members of the cluster. Since our measurements are inherently inaccurate, and the members of the cluster are so close to each other, we argue that dealing with individual cluster-routers is too fine a granularity. The results are much more meaningful if we treat all the members of a cluster as one virtual node.

To deal with the first problem, we relied on DNS queries. We found that 94% of the IP addresses that our `traceroute` data discovered are registered in DNS. Our assumption was that usually a router has many IP addresses but only one DNS name. Thus, we translated all the IP addresses to their DNS names, and used the names as node identifiers. Our experiments showed this DNS-based node identification to be an effective heuristic.

We remark that, originally, we planned to use our algorithms on the `traceroute` data from datasets D1 and D2 of [13]. Unfortunately, we were unable to reliably identify which IP addresses belonged to the same router from the stored datasets. The datasets do not include the DNS names of the routers, and querying today's DNS failed on 61% of the IP addresses that were discovered in the 1994 and 1995 `traceroute`s. Apparently, most of the routers have been replaced or reconfigured with different IP addresses over the last five years. Our inability to use this data was the main motivation for our own data collection effort.

Our solution to the second problem, of identifying and unifying cluster-routers into virtual nodes is partly mechanized, and partly art. We relied on two sources of information. One source is that backbone carriers typically use a clear naming convention (e.g., all the routers in AlterNet's Chicago hub have DNS names ending with `chi.alter.net`). The other source is that some carriers actually make their network structure and router naming conventions publically available, (e.g., Sprintlink [2], AboveNet [1]). Combining these sources, we were able to unify all the major hubs that showed up in our data into virtual nodes.

A.2 Unidirectional or bidirectional variables?

As we discussed in Section IV-A, we needed to decide whether to use uni- or bidirectional variables. The routing in the Internet is asymmetrical, i.e., the return path from B to A may be totally or partially disjoint from the route from A to B. Unfortunately, `traceroute` only provides the list of routers on one direction of the round trip.

Using unidirectional variables with this data would have required us to take the `traceroute` from A to B and splice it with the `traceroute` from B to A to create the full round-trip path. Using bidirectional variables was simpler, but we would effectively be assuming that Internet routing is symmetric.

Our main goal was to explore the power of our algorithm, rather than to compute highly accurate distances. Furthermore, we wanted to be able to compare the algorithm's performance on Internet measurements with its performance on synthetic networks (see the next section), and the routing was assumed to be symmetric on the synthetic networks. Therefore, we chose to use bidirectional variables.

```
bungi.com
fmp.com
getnet.com
his.com
io.com
iserver.com
maps.vix.com
wvi.com
public.yahoo.com
telcom.arizona.edu
berkeley.edu
nd.edu
sdsc.edu
wisc.edu
above.net
abs.net
acadia.net
comnetcom.net
thor.csu.net
odyssesy.cwis.net
www.denver.net
erie.net
gem.net
gip.net
jet.net
fudge.nortel.net
ntrnet.net
stealth.net
structured.net
tp.net
uen.net
vineyard.net
beacon.webtv.net
```

Fig. 2. The list of domains/hosts where Tracers used in the Internet experiments resided.

```
sjc.above.com
bos.alter.net
chi.alter.net
dca.alter.net
dfw.alter.net
ewr.alter.net
hou.alter.net
lax.alter.net
nyc.alter.net
pao.alter.net
chicago.bbnplanet.net
nyc.bbnplanet.net
paloalto.bbnplanet.net
sanjose.bbnplanet.net
vienna.bbnplanet.net
sfo-bb.cerf.net
sanfrancisco.cw.net
westorange.cw.net
nchicago-core.nap.net
sl-bb*-ana-*.sprintlink.net
sl-bb*-chi-*.sprintlink.net
sl-bb*-nyc-*.sprintlink.net
sl-bb*-pen-*.sprintlink.net
sl-bb*-rly-*.sprintlink.net
sl-bb*-stk-*.sprintlink.net
sl-gw*-che-*.sprintlink.net
iad.verio.net
or.nw.verio.net
nyc.verio.net
pao.verio.net
phl.verio.net
pvu.verio.net
sjc.verio.net
```

Fig. 3. The list of domains/sites to which distances were successfully computed.

B. Data collection

In this experiment we selected a set of machines (Tracers) and conducted `traceroute` measurements between all pairs of machines in this set. We used up to 33 publicly available `traceroute` servers (see list in Figure 2), out of the 96 US sites available at `www.traceroute.org`.

Using 33 Tracers, we conducted 33×32=1056 `traceroute`s. Eight of them were not usable, e.g., one measurement had a routing loop, and were discarded. The `traceroute`s revealed the IP addresses of 2115 interfaces which we identified using DNS queries. Of these, 122 IP addresses where not in the DNS database. Using the DNS names, we unified the IP addresses into 652 virtual nodes (as described in the previous section). We then proceeded to identify crossing points and segmentize the paths. The result was a segment graph connected by 846 segments.

C. The algorithm's performance

The system we fed our algorithm with had 1048 equations $(= 1056 - 8)$ and 846 variables. The algorithm solved 593 of the $\mathbf{y}$ variables (recall Equation (6)). Using procedure `compute-sub-paths` (Section IV-E) our algorithm successfully computed 499 new distances (in addition to the original 1048 measurements).

Despite the fact that the 33 Tracer sites were selected arbitrarily, without any attempt to spread them out in any particular way, we were able to compute the distances to an additional 33 nodes. The list of these discovered "virtual-Tracer" nodes is given in Figure 3. It includes nine out of the fifteen major hub sites of AlterNet (UUNET) in North America (in Boston, MA, Chicago, IL, Washington, DC, Dallas, TX, Newark, NJ, Houston, TX, Los-Angles, CA, New-York, NY, and Palo-Alto, CA), and seven of Sprintlink's fourteen sites (in Anaheim, CA, Chicago, IL, New-York, NY, Pennsauken, NJ, Relay, MD, Stockton, CA, and Cheyenne, WY).

To give a taste of the power of our method, and also to demonstrate how rich the calculated topology is, we describe the details of the computed distances for a particular virtual Tracer that our algorithm discovered: the AlterNet site in Los-Angles, CA (`lax.alter.net`). For this site, our algorithm calculated distances to 11 other virtual Tracers: `chi.alter.net`, `dca.alter.net`, `nyc.alter.net`, `ewr.alter.net`, `dfw.alter.net`, `hou.alter.net`, `pao.alter.net`, `sfo-bb.cerf.net`, `pao.verio.net`, `nw.verio.net`, `sjc.above.net`; Our algorithm also computed distances from the same site to 17 of the original Tracers: `thor.csu.net`, `trojan.neta.com`, `sdsc.edu`, `wvi.com`, `yahoo.com`, `www.denver.net`, `maps.vix.com`, `beacon.webtv.net`, `berkeley.edu`, `xenon.gem.net`, `odyssesy.cwis.net`, `telcom.arizona.edu`, `bungi.com`, `donjon.fmp.com`, `uen.net`, `abs.net`, and `jet.net`. Overall, we managed to compute distances to 28 other nodes from this node. This is about the average for the data we collected.

VI. Synthetic networks

A. Network generation models

We used two different network generators, to generate synthetic networks with different characteristics. One generator was based on work by Waxman [17], the other on work by Faloutsos et al. [7]. The generation algorithms use the following models.

EX model [17] — In the EX model, nodes are placed on a plane, and the probability for two nodes to be connected by a link decreases exponentially with the Euclidean distance between them. This nicely models intranets, but it is now debatable how well it models the Internet structure.

PL model [7] — In the PL model the node connectivity follows a power-law rule: very few nodes have high connectivity, and the number of nodes with lower connectivity increases exponentially as the connectivity decreases. This model is based on Internet measurements, where a node is an autonomous system (AS).

We generated synthetic networks comprised of 600 and 1000 nodes for each of the network generation models. In these networks, we assigned Tracers randomly to the network nodes. We varied the number of Tracers, and re-randomized their locations in the network.

We assumed that routing is symmetric on the synthetic networks, and that the routes followed the shortest paths between Tracers. Thus, for each generated network and each random choice of Tracer locations, we solved the all-pairs-shortest-path problem (limited to pairs of Tracers).

In order to compare the results on the synthetic networks with the Internet measurements, we needed to vary the number of real Tracers. We did this by taking our original 33 Tracers, and choosing a random subset of them. We took the shortest paths between the selected Tracers, and used those as the simulated measurement paths.

To demonstrate the robustness of our algorithm, we injected noise into the simulated delay measurements along each path. We first chose a random delay d_e for every link e in the network. The delay d_e served as the true (ideal) delay, that is not known to the algorithm. We quantified the amount of injected noise by a parameter $0 \leq \lambda < 1$. For a given measurement path, we assigned a measured delay value r_e to every link e along the path, and r_e was chosen uniformly at random from the range $[d_e(1-\lambda) : d_e(1+\lambda)]$, and the measurement along the path was then $\sum_e r_e$. Note that the same link e may get different values of r_e for different paths it belongs to.

B. Results and interpretation

Figure 4 shows our algorithm's performance on the synthetic networks together with the results on the Internet measurements.

Figure 4(A) shows the number of non-Tracer nodes our algorithm was able to discover (i.e., compute at least one distance to). Figure 4(B) shows the same data as a percentage of the number of Tracers. We can clearly see that in all cases, as more Tracers are added, the algorithm discovers more non-Tracers—in both absolute and relative numbers. The gains are substantial in all cases, ranging between 70%-214%. We can also see that the network generation model makes a big difference: for 30 Tracers, on the EX-generated networks our algorithm found 59 and 64 additional nodes on average (198% and 214%), while on PL-generated networks the algorithm discovered 21 and 28 nodes on average. We can also see that the algorithm's performance on Internet data is close to its performance on PL-networks, lending some support to the argument that the power-law rule is a good model for the Internet structure.

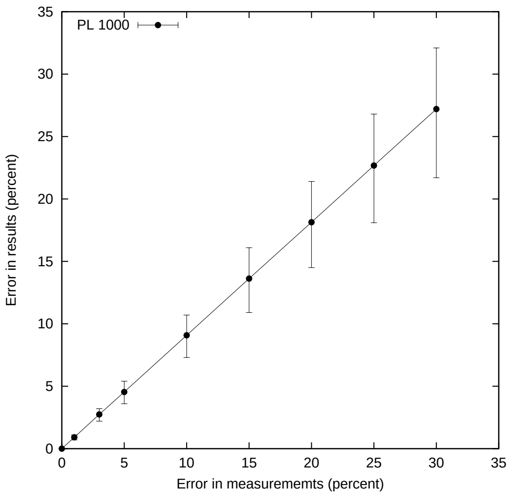

Fig. 5. The root mean square error as a function of the injected noise. Each point represents the average of ten experiments each on a 1000 node network with 30 Tracers.

Figure 4(C) shows the number of new distances that our algorithm succeed to calculate. Figure 4(D) shows the same data as a percentage of the number of measurements $\binom{t}{2}$. We see from Figure 4(C) that, again, as more Tracers are added, the algorithm computes more distances. Surprisingly, the algorithm did significantly better on the Internet measurements than on any of the synthetic networks, computing 415 additional distance—more than double the number of additional distances computed for the closest synthetic network, which is an EX network. The number of computed distances grows roughly linearly with the number of Tracers t, however, Figure 4(D) shows that the growth rate is slower than the number of measurements, which is quadratic.

Finally, Figure 5 shows the effects of the injected noise on our algorithm. Since we know the "true" distance for each link, we can compare it to the computed distance. The figure shows how the root of the mean square error in the computed distance varies with the rate of injected noise λ. For each instance, we calculated the standard deviation of the error. The length of the vertical bars are the average of the standard deviations over all the simulation conducted with the same injected noise λ. We can see clearly that on average, our algorithm slightly reduces the measurement noise: e.g., for 30% injected noise, we found an average of 27% error in the results. The significance here is that despite its algebraic components, the algorithm does not amplify measurement noise. Roughly speaking, the computed distances are as noisy as the inputs.

VII. Concluding Remarks

We presented an algorithm that extracts as much distance information as possible from end-to-end measurement data. The

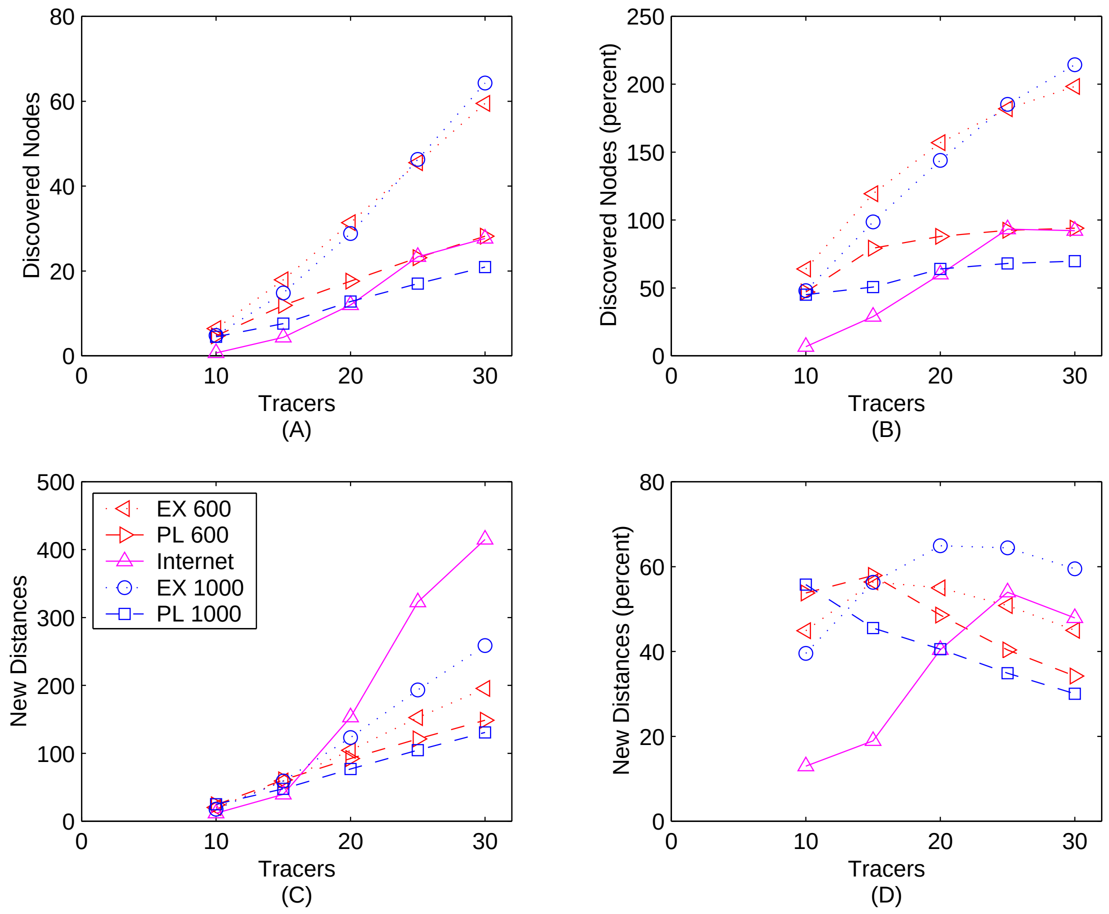

Fig. 4. Results of the algorithm testing on real and simulated data: (A) The number of virtual Tracers our algorithm discovered, as a function of the number of Tracers. (B) The percentage (out of the number of Tracers) of virtual Tracers our algorithm discovered, as a function of the number of Tracers. (C) The number of new distances that were calculated, as a function of the number of Tracers. (D) The ratio between the number of new distances that were calculated and the number of number of measurements, as a function of the number of Tracers.

algorithm performed well on real and on synthetic network measurements. This strong results are achieved using a practical and reasonable computational complexity. We believe our results can be readily used to improve mirror placement.

There are several research directions we intend to study. First, one must understand the best way to handle noise and different assumptions and noise models. Another important research direction is to understand how to place Tracers in the network in a way that will enable maximal gain from our algorithm. It is also very interesting to study the inter-relations between our algorithm and spanners [9] in order to achieve optimal data to overhead ratio.

References

[1] Abovenet—global one-hop network. `http://www.above.net/network/network.html`.

[2] Sprint internet services. `http://www.sprintlink.net/maint/`.

[3] G. Almes, S. Kalidindi, and M. Zekauskas. A one-way delay metric for IPPM, September 1999. Request for Comments: 2679.

[4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *SCIENCE*, 286:509 – 512, 15 October 1999.

[5] W. Cheswick, J. Nonnenmacher, Cenk Sahinalp, R. Sinha, and K. Varadhan. Modeling internet topology. Technical Report Technical Memorandum 113410-991116-18TM, Lucent Technologies, 1999.

[6] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

[7] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM*, August 1999.

[8] Paul Francis, Sugih Jamin, Vern Paxson, Lixia Zhang, Daniel Gryniewicz, and Yixin Jin. An architecture for a global internet host distance estimation service. In *IEEE Infocom'99*, New-York, NY, USA, March 1999.

[9] Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. On the placement of internet instrumentation. In *IEEE Infocom 2000*, Tel-Aviv, Israel, March 2000.

[10] Sunil Kalidindi and Matthew J. Zekauskas. Surveyor: An infrastructure for internet performance measurements. In *INET'99*, San Jose, CA, USA, June 1999.

[11] F. LoPresti, N.G. Duffield, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal delay distributions. UMass Computer Science Technical Report TR99-55, November 1999.

[12] K. Moore, J. Cox, and S. Green. Sonar - a network proximity service. Internet-Draft, `http://www.netlib.org/utk/projects/sonar/`, February 1996.

[13] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.

[14] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99 – 116, 1989.

[15] Y. Shavitt, X. Sun, A. Wool, and B. Yener. Computing the unmeasured: An algebraic approach to Internet mapping. DIMACS TR 2000-15, 2000.

[16] Wolfgang Theilmann and Kurt Rothermel. Dynamic distance maps of the internet. In *IEEE Infocom 2000*, Tel-Aviv, Israel, March 2000.

[17] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.