# Computing the Unmeasured: An Algebraic Approach to Internet Mapping

Yuval Shavitt, *Senior Member, IEEE*, Xiaodong Sun, Avishai Wool, and Bülent Yener, *Senior Member, IEEE*

***Abstract*—Distance estimation is important to many Internet applications. It can aid a World Wide Web client when selecting among several potential candidate servers, or among candidate peer-to-peer servers. It can also aid in building efficient overlay or peer-to-peer networks that dynamically react to change in the underlying Internet. One of the approaches to distance (i.e., time delay) estimation in the Internet is 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.**

**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.**

***Index Terms*—Delay inference, end-to-end measurements, network tomography.**

## 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 [1]–[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 World Wide Web (WWW) is an example of such a situation, as more and more popular sites open mirror sites that are geographically scattered. Many methods have been suggested to aid clients in selecting the best server (e.g., see [5]–[7]). All of them rely on network delay as one of the most important metrics to consider. In fact, Obraczka and Silva [7] have found that the single most important measure in server selection is network delay. Currently, however, the selection is usually done by the server, which directs the client to a mirror that is expected to serve it with the lowest delay (possibly also taking the server load into account). A special case of the mirror site solution is the well publicized content delivery network (CDN) [8], which in some cases replicates information to dozens of mirror sites.

Distance estimation is important also when ones wishes to optimize overlay networks. Shi and Turner [9] suggested deploying multicast via overlay services, and suggested several algorithms which optimize the end-to-end delay. Ratnasamy *et al.* [10] suggested to optimize overlay networks by binning together nodes whose relative delay is short. Other overlay applications in which distance estimation is useful for efficient operation include: application-layer anycasting [11], distributed object repositories, hierarchical caching, etc.

IDMaps [12] 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. For simplicity, IDMaps uses each autonomous system (AS) as a region.[1] The tracers advertise their measurement information to clients such as SONAR [13] or HOPS [12] servers, and these servers use the measurements to construct an estimated distance map of the network. Nodes of an overlay network can then query these topology servers regarding either distances,[2] or more likely coordinates [14], [15] to be used in optimizing their internal routing.

Measuring the distance (delay) between two IDMaps tracers can be done in many ways. A seemingly obvious choice is to use the `traceroute` program, which returns the IP addresses of all the routers on the route to the destination and the round-trip time (RTT) to each one of them. However, this approach suffers from two limitations.

Manuscript received November 15, 2002; revised May 1, 2003. The work of Y. Shavitt was supported in part by a Grant from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel. The work of X. Sun was supported in part by the Discrete Mathematics and Theoretical Computer Science Department (DIMACS), and in part by the National Science Foundation (NSF) under Grant CCR-9700239 and Grant CCR 9988526.

Y. Shavitt is with the Department of Electrical Engineering-Systems, Tel-Aviv University, Tel-Aviv 69978, Israel (e-mail: shavitt@eng.tau.ac.il).

X. Sun was with the Mathematics Department, Rutgers University, Piscataway, NJ 08854 USA. He is now with the School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540 USA (e-mail: sunxd@ias.edu).

A. Wool is with the Department of Electrical Engineering-Systems, Tel-Aviv University, Tel-Aviv 69978, Israel, and also with Lumeta Corporation, Somerset, NJ 08873 USA (e-mail: yash@acm.org).

B. Yener is with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: yener@cs.rpi.edu).

Digital Object Identifier 10.1109/JSAC.2003.818796

[1]Large ASes such as backbone providers are further divided into smaller regions.

[2]It was shown in [12] that while the triangle inequality does not hold in a large percentage of the Internet triangles, its violation is minuscule and does not hurt the distance approximation.

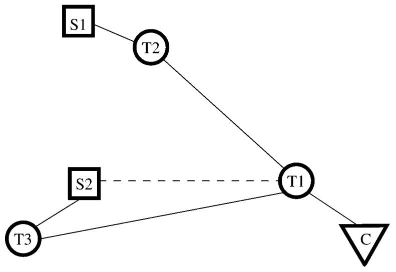

Fig. 1. A motivation for discovering extra distances.

1) The RTT estimates returned by `traceroute` are not accurate and are regarded only as very gross estimates of the real RTTs.[3]
2) The overhead of performing `traceroute` is significantly higher than that of the obvious alternative, which is `ping`.

A simple and effective way to measure delay between two end points is to use `ping`. A few time-spaced trains of `ping` probes or application level probes can be used to measure the delay between end points quite accurately. For example, Zeitoun *et al.* [16] found that using 8–12 probes is sufficient to achieve a good propagation delay estimate for a wide range of interprobe distances. More sophisticated tools such as `pathchar` (see [17] for a list) can map the delay, loss and other characteristics of an entire path between two points. However, these tools typically require hundreds of probes and several minutes to map a single multihop path with acceptable accuracy [18], and are thus not practical for massive usage.

Because of the above-mentioned reasons, the designers of the IDMaps system chose to use `ping` trains as its main measurement tool, coupled with a low frequency of `traceroute` usage to discover the routes. We refer the reader to [12] and [16] for further discussions regarding the design of IDMaps. In turn, our algorithms are designed to extract as much distance information as possible from this combination of `ping` and `traceroute` data.

## *B. Contributions*

In this paper, we present an algorithm that increases the effectiveness of end-to-end distance measurements at no additional overhead. Given end-to-end distance measurements (e.g., using `ping`) and the routes along which the measurements were conducted (e.g., using `traceroute`) our algorithm computes *additional* distances, to and between intermediate nodes. The result is a more detailed distance map, which provides better distance estimation. For example, consider the situation depicted in Fig. 1. We have three tracers, T1, T2, and T3, a client, C, and two possible servers, S1 and S2. Using tracer-to-tracer measurements and tracer-to-AS measurements, we estimate the distance between the client and the servers by the length of the concatenated measured paths $\mathrm{C-T1-T2-S1} = 40 + 160 + 50 = 250$ and $\mathrm{C-T1-T3-S2} = 40 + 200 + 40 = 280$. As a result, we will wrongly identify S1 as the closest server to C. However, if we can discover the additional distance $\mathrm{T1-S2} = 150$, we can better estimate the distance from the client C to the server S2 over the path $\mathrm{C-T1-S2} = 40 + 150 = 190$, and identify S2 as the closest server.

[3]Van Jacobson, the developer of `traceroute`, also acknowledged this point at his April 1997 talk at the Mathematical Sciences Research Institute, Berkeley, CA, and thus, motivated the introduction of `pathchar`.

An alternative way to look at the above is to notice that our algorithm discovers distances to new nodes in the network which can be used as dummy tracers in the approximated map. It was shown [12], [19] that the accuracy of pointing a client to the closest mirror increases with the number of tracers and that one can select the number of deployed tracers based on the network size and a target accuracy. Given that our algorithm discovers dummy tracers one can deploy less tracers to get the same accuracy.

The main idea behind our approach is that using the measurement 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 from the end-to-end measurements the distances to the crossing points and between them. 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 Internet traces (from Oct. 1999).

Our algorithm is computationally efficient, and can handle noisy measurements. It is adapted to handle two types of noise. Two-sided noise appears when we are interested in the average delay: in this case, a noisy measurement may be either higher or lower than the true value. To deal with two-sided noise, we use a least-squares approximation algorithm (cf. [20, Ch. 31]). One-sided noise appears when we are interested in measuring the minimal propagation delay: in this case, a noisy measurement cannot fall below the true value. To deal with one-sided noise, we use linear programming with additional slack variables.

To evaluate our algorithm, we studied its performance against two varieties of synthetic randomly generated networks and also against data we collected from the Internet. In all cases, our algorithm succeeded in computing 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 points as the original tracers, a 100% gain. For randomly generated networks, the gain was over 200% when we used the Waxman method [21] and about 50% when we used networks generated according to the recently discovered power law on the node connectivity [1], [22].

## *C. Related Work*

Francis *et al.* described the IDMaps architecture and philosophy of operation in [23]. 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 [24] suggested to use hierarchical tracer structure to reduce the measurement overhead. A different approach was suggested by Francis *et al.* [12] that showed how spanners [25] 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. [26]) aims at using multicast inference to characterize network loss and queuing 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 (queuing delay and loss) in tree segments. Rubenstein *et al.* [27] suggested techniques to identify link sharing among measurements, their techniques were later improved by Harfoush *et al.* [28].

In a recent work, published after the early versions of this paper [29], [30], Bradford *et al.* [31] suggested to use traceroute for network discovery and looked at the marginal utility of "tracerouting" to an increasing number of destinations from a limited number of origins. They claim success in revealing large portions of the Internet core, while they fail to reveal much of the "horizontal" links outside of the core. Their result can explain why the nodes we revealed in our Internet experiment were mostly core nodes.

The accuracy of using distance maps such as the ones generated by IDMaps for the mirror selection and related problems was studied by Jamin *et al.* [19]. They found that in about 85% of the cases the end to end measurements were sufficient to locate the closest server to the client. Our algorithm can potentially increase this number.

*1) Organization:* The rest of the paper is organized as follows. In the next section, we present the model and a simple example of the idea behind the algorithm. In Section III, we prove 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. 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 quasistatic, 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 [32], [23], the average delay [24], 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 2.1:* A **measurement path** is the route (list of nodes) between two different tracers as defined by the network topology and the underlying routing protocol.[4]

[4]If we assume RTT measurement and undirected variables for delay, a measurement path between tracer A and tracer B is simply the route from A to B. We can also use variables for each direction of every link and then a measurement path is the concatenation of the two unidirectional paths between A and B.

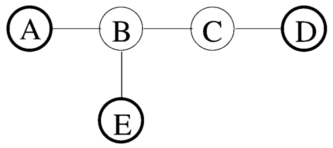

Fig. 2. Five-node network example.

*Definition 2.2:* 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 paths, and the graph edges are the union of all the links comprising the measurement paths.

*Definition 2.3:* A non-tracer node whose degree in the measurement graph is greater than two is called a **crossing point**.

*Definition 2.4:* A **segment** is a maximal subpath of a measurement path, whose end-points are either tracers or crossing points, that does not include an internal crossing point.

*Definition 2.5:* 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.

*1) 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 lengths can be derived.

### B. 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.

Consider the five node network of Fig. 2, 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`, we obtain the three routes, A-B-C-D, E-B-C-D, and A-B-E.

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: $s_1 = \mathrm{A{-}B}, s_2 = \mathrm{B{-}C{-}D}$, and $s_3 = \mathrm{E{-}B}$. 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

$$\begin{aligned} s_1 + s_2 &= 4 \\ s_2 + s_3 &= 7 \\ s_1 + s_3 &= 5. \end{aligned} \tag{1}$$

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 three measurements we were able to compute three additional distances, and discover distances to one additional non-tracer node (a 33% gain).

## III. 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 in 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 subtrees) and each subtree 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 3.1:* 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 subtrees connected to it. Let $l_1, l_2$, and $l_3$ be three leaves each in a different subtree. Obviously, the routes between any pair of these leaves must pass through $c$. ■

*Fact 3.2:* 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 3.3:* In every segment tree with more than two nodes there exists 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 3.4:* 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 3.3, 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 $l_2, l_3, \ldots, l_d$

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

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 3.3 holds for this tree, as well, if it contains more than two nodes, enabling the repetitive 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 3.5:* All the segment lengths in the tree can be found using $O(t)$ measurements.

*Proof:* In the proof of Theorem 3.4, 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 segment lengths.

## IV. 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 truly 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 [23], 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 queuing 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 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 2.4). For this, we need to identify all the crossing points (Definition 2.3). 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 unidirectional or bidirectional.

*1) 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 $j$th segment appears on the $i$th measurement path, and $a_{ij} = 0$, otherwise. The general form of the equations obtained after segmentation is as follows:

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

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 (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 (3) is solvable. Typically, it is underdefined for some variables and overdefined 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 (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. Permuting rows of $A$ is equivalent to reordering the measurements. Let $m' \leq m$ denote the number of nonzero 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 (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 $1 \leq j \leq m'$ are solvable from the new equations: The top left $m' \times m'$ submatrix of $A'$ is lower triangular and of rank $m'$. However, these are the only $y_j$ variables that can be solved: columns $m' < j \leq m$ 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 overdefined. 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 algebraically unsolvable. To deal with the measurement noise, we solve (7) for variables $y_j$ $(1 \leq j \leq m')$ using least-squares approximation (cf. [20, Ch. 31]).

In the absence of a detailed characterization of the noise, using least-squares is the standard noise elimination technique. In our case, the noise is dominated by the delays injected by each router along the path. If the router delays are independent and the path is long enough, the central limit theorem tells us that their sum approximates a normal distribution, i.e., the noise is Gaussian. For Gaussian noise it is known that the least-squares method provides the maximum-likelihood estimator (MLE). Therefore, we believe that the least-squares method is a reasonable noise elimination method for the problem at hand.

Using the least-squares method we find the values of $y_j$ $(1 \leq j \leq m')$ that minimize the function

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

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 is two-sided: the measured delay could be either too high (positive noise) or too low (negative noise). 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

$$\text{Minimize } \max_i \{e_i\} \quad \text{s.t.} \begin{cases} A'\mathbf{y} + \mathbf{e} = \mathbf{b} \\ \mathbf{y} \geq 0, \mathbf{e} \geq 0 \end{cases}.$$

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 Subpaths*

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 (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 subpath A-C. The same observation holds for any subpath of a measurement path.

Consider a subpath 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 subpath can be expressed as the sum of $x_{j_1}, \ldots, x_{j_\ell}$ (which is a 0–1 linear combination of $x_1, \ldots, x_m$). 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 one in coordinate $j$.

We would like to check whether the delay $p$ on the subpath is solvable. Using (5), we can write

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

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 subpath can be computed if and only if

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

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

```
Procedure **compute-subpaths**:
  For each measurement path $M_i, i = 1, \ldots, n$
    For each subpath $p$ 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.
```

### *F. Completeness*

A distance, by the metric definition (delay), is a linear combination of end-to-end distances. Using linear algebra, since $T$ is nonsingular, a subpath 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 subpath 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.

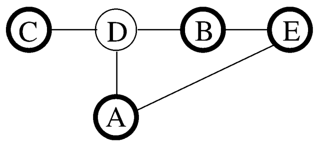

Fig. 3. Example of a network with asymmetric routing.

### G. Example of the Algorithm Operation

To get a better understanding of our algorithm, we include an annotated run of the algorithm on the network shown in Fig. 3. The network has four tracers, at A, B, C, E. The routes between the tracers A, B, and C are symmetric and pass through node D. These three tracers measure the three round-trip delays among themselves. Tracer E only measures its round-trip delay to A. However, the routing between E and A is asymmetric; the route from E to A passes through B, while the route from A to E uses the direct link A-E.

For simplicity, we assume no measurement errors in the delay. Assume that the four round-trip measurements yield the following numbers: $\mathrm{A} \leftrightarrow \mathrm{B} = 7, \mathrm{A} \leftrightarrow \mathrm{C} = 8, \mathrm{B} \leftrightarrow \mathrm{C} = 9$, and $\mathrm{A} \leftrightarrow \mathrm{E} = 11$. There are eight directional links that appear in the measurement graph: $AD, BD, CD, DA, DB, DC, EB, AE$. Let $\mathbf{x} = (AD, BD, CD, DA, DB, DC, EB, AE)$, and $\mathbf{b} = (7, 8, 9, 11)$. Then, we can write the following matrix equation:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 7 \\ 8 \\ 9 \\ 11 \end{pmatrix}. \tag{11}$$

It is easy to see that two column pairs are identical. Columns 3 and 6 correspond to links CD and DC, and are united to a segment representing C-D-C (which is bidirectional). The other identical pair, columns 7 and 8, corresponds to links EB and AE. The fact that they are identical means that these two links always appear together in the equations and are united to a (unidirectional) segment A-E-B. The result is a smaller linear equation system

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \mathbf{x}' = \begin{pmatrix} 7 \\ 8 \\ 9 \\ 11 \end{pmatrix} \tag{12}$$

where $\mathbf{x}' = (AD, DB, BD, DA, CD + DC, EB + AE)$.

Next, the matrix $(A)$ in (12) is triangulated using Gauss elimination. The result is the matrix

$$A' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{13}$$

with the transformation matrix

$$T = \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & 1 \\ 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{14}$$

Solving the equation $A'\mathbf{y} = \mathbf{b}$, we get $\mathbf{y} = (7, -1, 5, 11, *, *)$, where "$*$" means an arbitrary value.

Now, we can check the solvability of the various segments. For example, for the unidirectional segment AD $c = (1, 0, 0, 0, 0, 0)$ and, thus, $\mathbf{c}T = (1, -1, -1, -1, 1, 1)$, which shows that we cannot compute the length of the unidirectional link AD by itself; the last two entries of $\mathbf{c}T$ are nonzero. However, checking for the bidirectional segment pair $\mathrm{AD} + \mathrm{DA}$, for which $\mathbf{c} = (1, 0, 0, 1, 0, 0)$, we get $\mathbf{c}T = (1, -1, -1, 0, 0, 0)$, which is solvable since the last two entries are zero. We get $\mathrm{AD} + \mathrm{DA} = y_0 - y_1 - y_2 = 3$. In the same way, we also get: $\mathrm{BD} + \mathrm{DB} = y_1 + y_2 = 4$ and $\mathrm{CD} + \mathrm{DC} = y_3 = 5$. These are all the additional segments we can obtain beyond the original four measurements, a 75% gain, with one additional nontracer node (D) discovered. Note that the single measurement $\mathrm{E} \leftrightarrow \mathrm{A}$ did not provide us with any extra information beyond its own round-trip delay.

### H. 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 the segments in $O(N)$. Writing the $n = O(t^2)$ 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. Under the assumption that the same segment are not shared by too many traces, there are $O(m/n)$ segments in a trace on average. 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 (10), where $m_s$ is the number of segments in a path. As a result, the cost of performing this stage is $O((m/n)^2 \cdot m(m - m'))$, per path, and $O(n(m/n)^2 \cdot m(m - m')) = O((m^3(m - m'))/n)$ in total.

Calculating the length of a segment or a segment group [using (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 publicly accessible `traceroute`-ing machines as our tracers, collected data, and then applied our algorithm to this data.

## *A. Preliminary Issues*

*1) Node Identification:* When faced with multiple `traceroutes` 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 Internet protocol (IP) addresses directly as node identifiers creates two problems.

1) IP address are allocated to interfaces rather than to routers, 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 router.
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 `traceroutes` 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 domain name system (DNS) queries. Our assumption was that a router usually 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. We found that 94% of the IP addresses that our `traceroute` data discovered are registered in DNS. For the remaining 6% of IP addresses we used the IP address itself as the node identifier. Our experiments showed this DNS-based node identification to be an effective heuristic. Govidan and Tangmunarunkit [33] suggest other methods for alias resolution based on probing routers with user datagram protocol (UDP) packets destined to a nonexistent port. Using their method in the context of our algorithms is left as a topic for further research.

We remark that, originally, we planned to use our algorithms on the `traceroute` data from datasets D1 and D2 of [34]. 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 `traceroutes`. 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 publicly available, (e.g., Sprintlink [35], AboveNet [36]). Combining these sources, we were able to unify all the major hubs that showed up in our data into virtual nodes. Similar solutions were use by Paxson in his Ph.D. dissertation [37].

*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 sometimes 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 Section VI), and the routing was assumed to be symmetric on the synthetic networks. Therefore, we chose to use bidirectional variables.

## *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 33 publicly available `traceroute` servers (see list in Fig. 4), out of the 96 U.S. sites available at www.traceroute.org.

Using 33 tracers, we conducted $33 \times 32 = 1056$ `traceroutes`. Eight of them were not usable, e.g., one measurement had a routing loop, and were discarded. The `traceroutes` 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. 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 (6)]. Using procedure `compute-subpaths` (Section IV-E), our algorithm successfully computed 499 new distances (in addition to the original 1048 measurements).

```
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. 4. List of domains/host where tracers 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. 5. List of domains/sites to which distances were successfully computed.

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 Fig. 5. It includes nine out of the 15 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`, `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 [21], the other one is the Inet simulator from University of Michigan [22] which is based on work by Faloutsos *et al.* [1]. The generation algorithms use the following models.

*1) EX Model [21]:* 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.

*2) PL Model [1]:* 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 rerandomized their locations in the network.

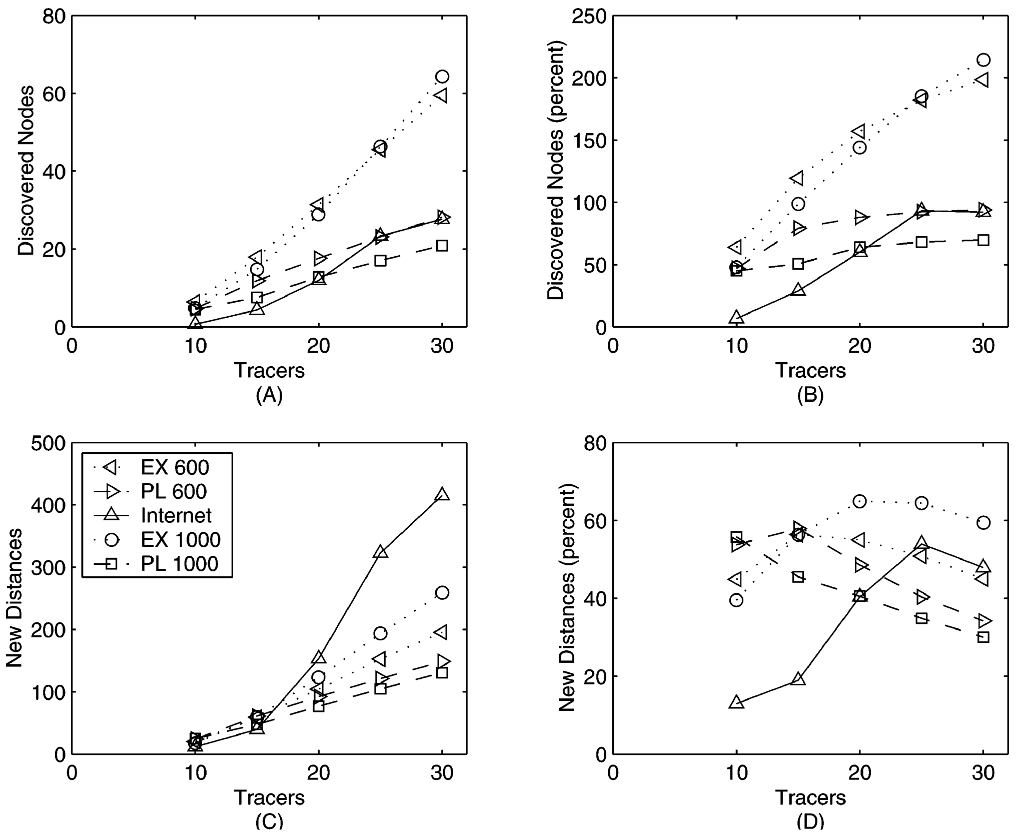

Fig. 6. Results of the algorithm testing on real and simulated data. (a) The number of virtual tracers our algorithm discovered, as a function of the number of tracers. (b) The percentage (out of the number of tracers) of virtual tracers our algorithm discovered, as a function of the number of tracers. (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 measurements, as a function of the number of tracers.

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

Fig. 6 shows our algorithm's performance on the synthetic networks together with the results on the Internet measurements.

Fig. 6(a) shows the number of non-tracer nodes our algorithm was able to discover (i.e., compute at least one distance to). Fig. 6(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.

Fig. 6(c) shows the number of new distances that our algorithm succeed to calculate. Fig. 6(d) shows the same data as a percentage of the number of measurements $\binom{t}{2}$. We see from Fig. 6(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 distances (for 30 tracers)—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, Fig. 6(d) shows that the growth rate is slower than the number of measurements, which is quadratic in $t$.

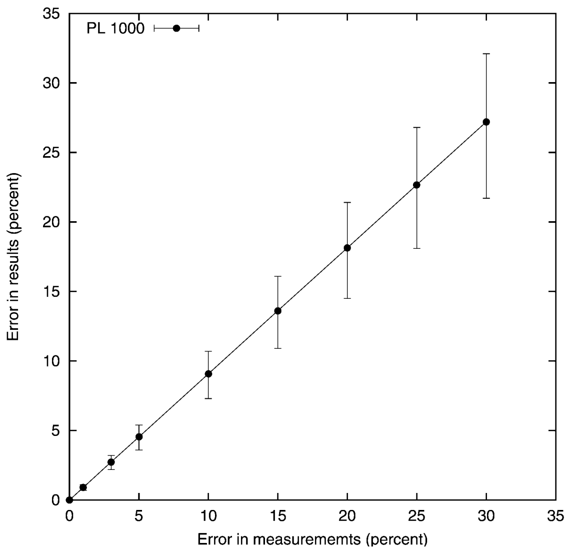

Fig. 7. The root mean square error as a function of the injected noise. Each point represents the average of ten experiments each on a 1000 node network with 30 tracers.

Finally, Fig. 7 shows the effects of the injected noise on our algorithm. Since we know the "true" distance for each link, we can compare it with 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 $\lambda$. 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 $\lambda$. 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 algorithm performed well on real and on synthetic network measurements. These strong results are achieved with 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 interrelations between our algorithm and spanners [32] in order to achieve an optimal data-to-overhead ratio.

## References

[1] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," presented at the ACM SIGCOMM, Cambridge, MA, Aug. 1999.

[2] W. Cheswick, J. Nonnenmacher, C. Sahinalp, R. Sinha, and K. Varadhan, "Modeling Internet Topology," Lucent Technologies, Tech. Memo. 113 410-991 116-18TM, 1999.

[3] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, Oct. 15, 1999.

[4] R. Albert and A.-L. Barabási, "Topology of evolving networks: Local events and universality," *Phys. Rev. Lett.*, vol. 85, p. 5234, 2000.

[5] R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," presented at the IEEE INFOCOM 1997, Kobe, Japan, Apr. 1997.

[6] S. G. Dykes, C. L. Jeffery, and K. A. Robbins, "An empirical evaluation of client-side server selection algorithms," presented at the IEEE INFOCOM 2000, Tel-Aviv, Israel, Mar. 2000.

[7] K. Obraczka and F. Silva, "Network latency metrics for server proximity," presented at the GLOBECOM, San Francisco, CA, Dec. 2000.

[8] FreeFlow (1998). [Online]. Available: http://www.akamai.com/

[9] S. Shi and J. Turner, "Routing in overlay multicast networks," presented at the IEEE INFOCOM'02, New York, NY, June 2002.

[10] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," presented at the IEEE INFOCOM'02, New York, NY, June 2002.

[11] E. W. Zegura, M. H. Ammar, Z. Fei, and S. Bhattacharjee, "Application-layer anycasting: A server selection architecture and use in a replicated web service," *IEEE/ACM Trans. Networking*, vol. 8, pp. 455–466, Aug. 2000.

[12] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A global Internet host distance estimation service," *IEEE/ACM Trans. Networking*, vol. 9, pp. 525–540, Oct. 2001.

[13] K. Moore, J. Cox, and S. Green, "SONAR—A Network Proximity Service," Internet-Draft, http://www.netlib.org/utk/projects/sonar/, Feb. 1996.

[14] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates based approaches," presented at the IEEE INFOCOM 2002, New York, NY, June 2002.

[15] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in euclidean space," presented at the IEEE INFOCOM 2003, San Francisco, CA, Mar. 2003.

[16] Z. Wang, A. Zeitoun, and S. Jamin, "Challenges and lessons learned in measuring path RTT for proximity-based applications," presented at the Passive and Active Measurement Workshop (PAM) 2003, San Diego, CA, Apr. 2003.

[17] Tools [Online]. Available: http://www.caida.org/tools

[18] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," presented at the ACM SIGCOMM, Stockholm, Sweden, Aug. 2000.

[19] E. Cronin, S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," *IEEE J. Select. Areas Commun.*, vol. 20, pp. 1369–1382, Sept. 2002.

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

[21] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1617–1622, Dec. 1988.

[22] C. Jin, Q. Chen, and S. Jamin, "Inet Topology Generator," Univ. Michigan, Ann Arbor, MI, Tech. Rep. CSE-TR-433-00, July 2000.

[23] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin, "An architecture for a global Internet host distance estimation service," presented at the IEEE INFOCOM'99, New York, NY, Mar. 1999.

[24] W. Theilmann and K. Rothermel, "Dynamic distance maps of the Internet," in *Proc. IEEE INFOCOM 2000*, Tel-Aviv, Israel, Mar. 2000, pp. 275–284.

[25] D. Peleg and A. A. Schäffer, "Graph spanners," *J. Graph Theory*, vol. 13, no. 1, pp. 99–116, 1989.

[26] F. LoPresti, N. G. Duffield, J. Horowitz, and D. Towsley, "Multicast-Based Inference of Network-Internal Delay Distributions," Univ. Massachusetts, Amherst, MA, UMass Comput. Sci. Tech. Rep. TR99-55, Nov. 1999.

[27] D. Rubenstein, J. Kurose, and D. Towsley, "Detecting shared congestion of flows via end-to-end measurement," presented at the ACM SIGMETRICS'00, Santa Clara, CA, June 2000.

[28] K. Harfoush, A. Bestavros, and J. Byers, "Robust identification of shared losses using end-to-end unicast probes," presented at the 6th IEEE Int. Conf. Network Protocols (ICNP'00), Osaka, Japan, Oct. 2000.

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

[30] ——, "Computing the unmeasured: An algebraic approach to Internet mapping," presented at the IEEE INFOCOM'01, Anchorage, AK, Apr. 2001.
[31] P. Barford, A. Bestavros, J. Byers, and M. Crovella, "On the marginal utility of network topology measurements," presented at the ACM SIGCOMM Internet Measurement Workshop, San Francisco, CA, Nov. 2001.
[32] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of Internet instrumentation," presented at the IEEE INFOCOM 2000, Tel-Aviv, Israel, Mar. 2000.
[33] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proc. IEEE INFOCOM 2000*, Tel-Aviv, Israel, Mar. 2000, pp. 1371–1380.
[34] V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Trans. Networking*, vol. 5, pp. 601–615, 1997.
[35] Sprint Internet Services [Online]. Available: http://www.sprintlink.net/maint/
[36] Abovenet—Global One-Hop Network [Online]. Available: http://www.above.net/network/network.html
[37] V. Paxson, "Measurements and analysis of end-to-end Internet dynamics," Ph.D. dissertation, Computer Sci. Div., Univ. California, Berkeley, CA, Apr. 1997.

**Yuval Shavitt** (S'88–M'97–SM'00) received the B.Sc. degree (*cum laude*) in computer engineering, the M.Sc. degree in electrical engineering, and the D.Sc. degree from the Technion—Israel Institute of Technology, Haifa, in 1986, 1992, and 1996, respectively.

From 1986 to 1991, he served in the Israel Defense Forces, first as a System Engineer and the last two years as a Software Engineering Team Leader. After graduation, he spent a year as a Postdoctoral Fellow in the Department of Computer Science, Johns Hopkins University, Baltimore, MD. From 1997 to 2001, he was a Member of Technical Staff at the Networking Research Laboratory, Bell Labs, Lucent Technologies, Holmdel, NJ. Since October 2000, he has been a Faculty Member in the Department of Electrical Engineering, Tel-Aviv University, Tel-Aviv, Israel. His recent research focuses on QoS routing and support and Internet measurement, mapping, and characterization.

Dr. Shavitt served as Technical Program Committee Member for INFOCOM 2000–2003, IWQoS 2001, and 2002, ICNP 2001, and MMNS 2001, and on the Executive Committee of INFOCOM 2000, 2002, and 2003. He is an Editor of *Computer Networks*, and served as a Guest Editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS and *Journal on World Wide Web*.

**Xiaodong Sun** received the B.Sc. degree in mathematics from Beijing University, Beijing, China, in 1990, the M.Sc. degree in applied mathematics from the Chinese Academy of Sciences, Beijing, China, in 1994, and the M.Sc. degree in computer science and the Ph.D. degree in mathematics from Rutgers University, Piscataway, NJ, in 2001 and 2002, respectively.

He is currently a Postdoctoral Fellow at the School of Mathematics, Institute for Advanced Study, Princeton, NJ. His recent research interests include Internet mapping, mobile computing and collaboration, and complexity lower bounds of problems in massive data set computation.

**Avishai Wool** received the B.Sc. degree (*cum laude*) in mathematics and computer science from Tel-Aviv University,Tel-Aviv, Israel, in 1989, the M.Sc. and Ph.D. degrees in computer science from the Weizmann Institute of Science, Rehovot, Israel, in 1992 and 1996, respectively.

He was a Member of Technical Staff at Bell Laboratories, Murray Hill, NJ. In 2000, he cofounded Lumeta Corporation, Somerset, NJ, a startup company specializing in network security. Since 2002, he has been an Assistant Professor in the Department of Electrical Engineering Systems, Tel-Aviv University. He is the creator of the Lumeta Firewall Analyzer. His research interests include firewall technology, network and wireless security, data communication networks, and distributed computing.

Dr. Wool is an Associate Editor of the *ACM Transactions on Information and System Security*. He has served on the program committees of the leading IEEE and ACM conferences on computer and network security. He is a Member of the Association for Computing Machinery (ACM), the IEEE Computer Society, and USENIX.

**Bülent Yener** (M'96–SM'03) received the B.S. and M.S. degrees in industrial engineering from the Technical University of Istanbul, Istanbul, Turkey, and the M.S. and Ph.D. degrees in computer science from Columbia University, New York, NY, in 1987 and 1994, respectively.

He is an Associate Professor of computer science at Rensellaer Polytechnic Institute (RPI), Troy, NY. Before joining RPI, he was a Member of Technical Staff at Bell Laboratories, Murray Hill, NJ. His current research interests include routing problems in wireless networks, Internet measurements, quality-of-service in the Internet protocol networks, and Internet security.

Dr. Yener has been serving on the Technical Program Committee of the leading IEEE conferences and workshops and is an Associate Editor of *IEEE Network*.