On the Placement of Internet Instrumentation

Sugih Jamin
Department of EECS
University of Michigan
Ann Arbor, MI 48109-2122
jamin@eecs.umich.edu

Cheng Jin
Department of EECS
University of Michigan
Ann Arbor, MI 48109-2122
chengjin@eecs.umich.edu

Yixin Jin
Computer Science Department
University of California Los Angeles
Los Angeles, CA 90095
yjin@cs.ucla.edu

Danny Raz
Bell Labs
Lucent Technologies
Holmdel, NJ 07733-3030
raz@research.bell-labs.com

Yuval Shavitt
Bell Labs
Lucent Technologies
Holmdel, NJ 07733-3030
shavitt@ieee.org

Lixia Zhang
Computer Science Department
University of California Los Angeles
Los Angeles, CA 90095
lixia@cs.ucla.edu

Abstract— **The IDMaps project aims to provide a distance map of the Internet from which relative distances between hosts on the Internet can be gauged [1]. Many distributed systems and applications can benefit from such a distance map service, for example, a common method to improve user perceived performance of the Internet is to place data and server mirrors closer to clients. When a client tries to access a mirrored server, which mirror should it access? With IDMaps, the closest mirror can be determined based on distance estimates between the client and the mirrors. In this paper we investigate both graph theoretic methods and *ad hoc* heuristics for instrumenting the Internet to obtain distance maps. We evaluate the efficacy of the resulting distance maps by comparing the determinations of closest replica using known topologies against those obtained using the distance maps.**

I. Introduction

The IDMaps project [1] aims at providing a distance[1] map of the Internet from which relative distances between hosts on the Internet can be gauged, i.e. is host a closer to host b or host c? We envision that many distributed systems and applications can greatly benefit from such a distance information service. For example, placing mirrors of popular Web sites has become a common method to improve user-perceived performance and to reduce network load. While strategically placed Web caches may improve performance of static content, using server mirrors has been proposed to improve performance of dynamic content. Given a set of mirrors, which one should a client access? Aside from the many *ad hoc* solutions, such as round robin DNS and geography- or IP-address-based partitioning, there also are proprietary systems that address this question[2] [2], [3]. One factor to consider in determining the mirror closest to a client is the topological distances between the client and the set of mirrors, the information IDMaps intends to provide.

The architecture of IDMaps [1] consists of a network of instrumentation boxes, which we call *Tracers*, distributed across the Internet. Tracers measure distances among themselves and between themselves and regions of the Internet to build a distance map. IDMaps aims at providing long-term measures of relative distances between hosts across the global Internet. Generally speaking, given a host a and a list of some other hosts on the Internet, IDMaps can return an ordered list of these other hosts by their distances to host a. Such a simple service suffices for applications, like the one described above, that do not necessarily need a precise or absolute measure of distances between hosts.

The regions of the Internet that Tracers measure are called *AP*s ("Address Prefixes"). Tracers measure distances among themselves and to APs. We call each such measurement a *virtual link* and the process of measuring, *tracing*. Tracers advertise their measured distances on one or more multicast channels. Collectively, these measured distances form the *distance map* or, equivalently, the *virtual topology* of the Internet. From the distance map, distance between any two hosts on the Internet can be estimated.

The efficacy of IDMaps may be evaluated based on how accurate the estimated distances are compared to actual distances, but for applications such as closest server selection, only the ordering of distances between a client and server mirrors is needed. We define an evaluation metric called P_{app}, for *application level performance metric* that compares the "correctness" of closest server selection using the distance map provided by IDMaps against selection based on the underlying topology. A more precise definition of the metric is provided in Section III-A. In this paper, we explore the following questions:

1. how many Tracers must be deployed?
2. where should these Tracers be placed?
3. must the distance map be a full-mesh?

Although our goal in this paper is biased towards the IDMaps project, we believe that the results should be generally applicable to the placement of measurement instrumentation boxes on the Internet.

In the next section, we review two graph theoretic approaches that can be used to determine the number and placement of Tracers. Both of these approaches require *a priori* knowledge of the network topology. Since the topology of the Internet is not known and is continually changing, we call these the *idealized* IDMaps placement algorithms and use their performance as yard sticks to evaluate the performance of our heuristics pre-

This project is funded in part by NSF grant number ANI-9876541. Sugih Jamin's is further supported by the NSF CAREER Award ANI-9734145 and the Presidential Early Career Award for Scientists and Engineers (PECASE) 1998. Additional funding is provided by MCI Worldcom, Lucent Bell-Labs, and Fujitsu Laboratories America, and by equipment grants from Sun Microsystems Inc. and Compaq Corp.

[1] We define *distance* in the generic graph-theoretic sense of path cost: the sum of the link-cost of all links along a path.

[2] These systems are not publicly disclosed and hence cannot be evaluated in this paper.

sented in Section III-B. In Section IV, we examine algorithms used to construct the distance map. Our experiments examine the tracer placement problem and the virtual link construction using P_{app} on artificially generated network topologies. We describe our topology generation process in Section V and our performance results in Section VI.

Notations: We adopt the following notations for this paper: the network is a graph $G(V, E)$, where V is the set of nodes, and $E \subseteq V \times V$ is the set of links. We use $N = |V|$ to denote the number of nodes in G, and $\mathcal{T}$ to denote the number of Tracers we place in the graph. We denote the distance between nodes u and v in the graph G, $d_G(u, v)$; we will omit G when it can be deduced from the context.

II. PLACEMENT ON KNOWN TOPOLOGY

In this section we review two graph theoretic approaches that can help us determine the number and the placement of Tracers when the network topology is known. We use the generic term "center" in place of "Tracer" in the following descriptions. We study two variants of the center placement problem: in the first case, the maximal center-node distance is given, and one is required to find the minimal number of centers needed to satisfy the maximal distance constrain; in the second case, the number of centers is given, and one needs to decide the locations of these centers such that the maximum distance between a node and the nearest center is minimized. In the following paragraphs, we give the formal definitions of the two variants.

Number of Centers: Given a network G with N nodes, (that is, the topology is *a priori* known), a bound $\mathcal{D}$, one has to find a smallest set of centers S_C such that the distance between any node i and its closest center $C_i \in S_C$ is bounded by $\mathcal{D}$. The performance metric (P_{diam}) is the size of this set ($|S_C|$). More formally find the minimum N such that there is a set $S_C \subset V$ with $|S_C| = N$ and $\forall v \in V : d(v, C_v) \leq \mathcal{D}$.

Center Placement: For the placement of a given number of centers, one could consider the following metric (P_{minK}): minimize the maximum distance between a node and the nearest center. This problem is known as the minimum K-center problem, which states: given an undirected graph $G = (V, E)$ with link costs satisfying the triangle inequality,[3] and an integer K, find a set of K nodes such that the maximum distance between a node on the graph and the nearest center is minimized.

We present two algorithms below, each of which can be used to solve both the *Number of Centers* problem and the *Center Placement* problem. The two algorithms were designed to solve somewhat different problems than Tracer placement. Our tacit assumption in applying these algorithms to Tracer placement is that by minimizing the maximum distance between a node and its closest Tracer, we will obtain the best distance estimate.

A. k-HST

We present in this subsection an algorithm based on the *k-hierarchically well-separated trees* (k-HST) [5]. The k-HST algorithm consists of two phases. In the first phase, the graph is recursively partitioned as follows: A node is arbitrarily selected from the current (parent) partition, and all the nodes that are within a random radius from this node form a new (child) partition. The value of the radius of the child partition is a factor of k smaller than the diameter of the parent partition. This process recurses for each partition, until each node is in a partition of its own. We then obtain a tree of partitions with the root node being the entire network and leaf nodes being individual nodes in the network. In the second phase, a virtual node is assigned to each of the partitions on each level. Each virtual node in a parent partition becomes the parent of the virtual nodes of the child partitions. The length of the links from a virtual node to its children is half the partition diameter. We embed the virtual nodes in the original graph based on a technique developed by Awerbuch and Shavitt [6]. Together, the virtual nodes also form a tree.

Algorithm 1 (Greedy placement)

1. $\mathcal{L} \leftarrow \mathcal{N}_r$
2. while $(diam(H_{\mathcal{L}}) > \mathcal{D})$
3. $\quad h \leftarrow H_{\mathcal{L}}$
4. $\quad \mathcal{L} \leftarrow \mathcal{L} - H_{\mathcal{L}}$
5. $\quad \mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{N}_h$

Fig. 1. Greedy placement of centers on an k-HST tree.

The randomization of a partition radius is done so that the probability of a short link being cut by partitioning decreases exponentially as one climbs the tree. Hence nodes close together are more likely to be partitioned lower down the tree. We take advantage of this characteristic of the resulting k-HST tree to devise the following greedy algorithm to find the number of centers needed when the maximum center-node distance is bounded by $\mathcal{D}$.

Let node r be the root of the partition tree, $\mathcal{N}_i$ be the children of node i on the partition tree, and $\mathcal{L}$ be a list of partitions sorted in the decreasing order of the partition diameter at all times. $H_{\mathcal{L}}$ denotes the partition at the head of the list, and $diam(H_{\mathcal{L}})$ its diameter. Fig. 1 presents our greedy algorithm on the k-HST tree. The algorithm pushes the centers down the tree until it discovers a partition with diameter $\leq \mathcal{D}$. The number of partitions, $|\mathcal{L}|$, is the minimum number of centers required to satisfy the performance metric P_{diam}. To select the actual centers, we can simply set the virtual nodes of these partitions in $\mathcal{L}$ to be the centers.

The k-HST-based greedy placement algorithm presented above tells us the number of centers needed to satisfy the performance metric P_{diam}. For any given budget of centers, the algorithm above can also be used to determine their placement. For example, to place K centers, we simply change line 2 in Fig. 1 with "while ($|\mathcal{L}| < K$)". Obviously, the performance metric P_{diam} may no longer be satisfied for K below a certain number.

B. Minimum K-Center

The placement of a given number of centers such that the maximum distance from a node to the nearest center is min-

[3] A cost function c satisfies the triangle inequality if for all vertices u, v, w in V, $c(u, w) \leq c(u, v) + c(v, w)$ [4].

Algorithm 2 (2-approximate minimum K-center [8])

1. Construct $G_1^2, G_2^2, \ldots, G_m^2$
2. Compute M_i for each G_i^2
3. Find smallest i such that $|M_i| \leq K$, say j
4. M_j is the set of K centers

Fig. 2. Two-approximate algorithm for the minimum K-center problem.

imized, known as the minimum K-center problem, is NP-complete [7]. However, if we are willing to tolerate inaccuracies within a factor of 2, i.e. the maximum distance between a node and the nearest center being no worse than twice the maximum in the optimal case, the problem is solvable in O($N|E|$) [8] as follows:

Given a graph $G = (V, E)$ and all its edges arranged in non-decreasing order by edge cost, c: $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$, let $G_i = (V, E_i)$, where $E_i = \{e_1, e_2, \ldots, e_i\}$. A *square graph* of G, G^2 is the graph containing V and edge (u, v) wherever there is a path between u and v in G of at most two hops, $u \neq v$—hence some edges in G^2 are pseudo edges, in that they don't exist in G. An *independent set* of a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that, for all $u, v \in V'$, the edge (u, v) is *not* in E. An independent set of G^2 is thus a set of nodes in G that are at least three hops apart in G. We also define a *maximal* independent set M as an independent set V' such that all nodes in $V - V'$ are at most one hop away from nodes in V'.

The outline of the minimum K-center algorithm from [8] is shown in Fig. 2. The basic observation is that the cost of the optimal solution to the K-center problem is the cost of e_i, where i is the smallest index such that G_i has a dominating set[4] of size at most K. This is true since the set of center nodes is a dominating set, and if G_i has a dominating set of size K, then choosing this set to be the centers guarantees that the distance from a node to the nearest center is bounded by e_i. The second observation is that a star topology in G_i, transfers into a clique (full-mesh) in G_i^2. Thus, a maximal independent set of size K in G_i^2 implies that there exists a set of K stars in G, such that the cost of each edge in it is bounded by $2e_i$: the smaller the i, the larger the K. The solution to the minimum K-center problem is the G_i^2 with K stars. Note that this approximation does not always yield a unique solution.

The 2-approximate minimum K-center algorithm can also be used to determine the number of centers needed to satisfy the performance metric P_{diam} by picking an index k such that $c(e_k) \leq \mathcal{D}/2$. The maximum distance between a node and the nearest center in G_k is then at most $\mathcal{D}$, and the number of centers needed is $|M_k|$.

The graph theoretic approaches described above assume known network topologies. However, the topology of the Internet may not be known to all parties at any one time. Furthermore, the Internet topology changes continuously, from physical and algorithmic causes. Nevertheless, there are scenarios in which the above algorithms would be applicable on the Internet. For example, an Internet Service Provider (ISP) may know the topology of its network and be able to use the algorithms to distribute Tracers within its own network, or companies may use the algorithms to distribute Tracers on their intranets. In this paper, results from the graph theoretic algorithms are used as yard sticks to evaluate the performance of our Tracer placement heuristics in the next section.

[4] a dominating set is a set of D nodes such that every $v \in V$ is either in D or has a neighbor in D.

III. Tracer Placement on the Internet

A. Number of Tracers and Performance Metric

The number of instrumentation boxes needed to cover the Internet depends greatly on the performance metric these boxes must satisfy. In this subsection we discuss various candidate performance metrics for IDMaps and the implications on the number of Tracers required.

One could apply the P_{diam} metric and require that each nodes is within a certain distance of the nearest Tracer, or require that the distances between nodes and Tracers to be minimized (P_{minK}). Neither of these metrics, however, tells us how good a service IDMaps provides. A higher level metric, such as requiring that $x\%$ of IDMaps estimates be within a factor of α of actual distances, may be more useful. Depending on the value of x and α, this performance metric may not be achievable for any reasonable number of Tracers. In the worst case, the number of Tracers required may be O(N). For example, in order to get a factor of 2 estimation in all cases, we need to put Tracers in a way that for any node u the distance from u to its closest Tracer does not exceed the shortest distance from u to any other node v. In many practical networks, this will require a large number of Tracers ($N/2$).

IDMaps does not intend to provide precise estimates of distances between hosts on the Internet; what it does provide is estimates of relative distances between a source (e.g. a client) and a set of potential destinations (e.g. server mirrors). Hence we adopt an application-level performance metric, P_{app}, which measures how often the determination of the closest mirror to a client, using the information provided by IDMaps, results in a correct answer. A quantitative evaluation of P_{app} requires a definition of some target accuracy, α. A good placement is one that maximizes the number of possible node pairs for which $\alpha\%$ of client nodes will correctly select the closest mirror using the distance map induced by IDMaps. We later introduce a practical lax version of this measure.

As we will show in Section VI, this rather lax metric still allows IDMaps to provide useful services with a small number of Tracers. Furthermore, in reality, the number of Tracers we can place, and where we can place them on the Internet, will be constrained by both administrative and budgetary considerations, our goal in this paper is not to determine the minimum number of Tracers required to provide distance estimates to a given precision of accuracy, but rather to evaluate the effectiveness of various placement strategies for varying number of available Tracers, according to the performance metric P_{app}.

B. Tracer Placement Heuristics

Given a number of Tracers and an unknown topology, we devise the following heuristics for Tracer placement:

Stub-AS: Tracers are placed only on stub Autonomous Systems (ASs). This would most likely reflect the initial deployment of Tracers on the Internet, when Tracers would be run from end hosts.

Transit-AS: Tracers are placed only on transit ASs, i.e. ASs connected to more than one other AS. This reflects deployment of IDMaps on ISP backbones. As IDMaps becomes more popular, we hope that there will be enough incentives for network providers and institutions with private networks to deploy it. For networks that do not have IDMaps deployed, Tracers could still be run from end hosts.

Mixed: Tracers are uniformly placed on the network. On one hand, this is the simplest placement method and does not assume any knowledge of network characteristics. On the other, it means Tracers are placed on both stub and transit ASs. In terms of deployment, this placement reflects IDMaps being partially deployed on some ISPs.

Idealized: For the purposes of comparative study, we also consider Tracer placement using the two algorithms presented in Section II. Since these algorithms require *a priori* knowledge of network topologies, we call this the *idealized* IDMaps.

IV. The Distance Map

Once Tracers are placed on the Internet, they start tracing to each other and to regions of the Internet, called APs (see definition in Section IV-B.) The resulting traces are advertised to IDMaps' clients. Clients of IDMaps, such as SONAR [9] or HOPS [10] servers, collect the advertised traces and construct distance maps. In this section, we first discuss Tracer-to-Tracer part of the distance map; then we discuss Tracer-to-AP virtual links.

A. Tracer-Tracer Virtual Links

As of the writing of this paper, there are about 60,000 routing address prefixes on the Internet [11]. Assuming we have 5% as many Tracers, and each Tracer traces to every other Tracer, there will be about 9 million virtual links to be continually traced and advertised. In [1], we identified cases in which IDMaps may reduce the number of virtual links traced and still maintains good distance estimates. We now generalize this result by applying the t-spanner algorithm [12] to distance map construction. A t-spanner of a graph is a subgraph where the distance between any pair of nodes is at most t times larger than in the original graph [13], [14]. Formally, given a graph, $G(V, E)$, a t-spanner is a subgraph $G'(V, E')$, $E' \subseteq E$ such that $d_{G'}(u, v) \leq t \cdot d_G(u, v), \forall u, v \in V$. The number of edges required to build a 5-spanner, for example, on a graph with N nodes is O($N^{3/2}$). For $t = \log N$, the number of edges required is O(N). We examine the effect of using different t values on the performance metric P_{app} in Section VI.

The design of IDMaps allows its clients to provide feedback to Tracers on which virtual links form their t-spanners and should be continually traced. To capture topological changes, instead of completely disregarding virtual links not currently used in building distance maps, Tracers will simply reduce the frequencies at which they trace and advertise these links. We next describe the t-spanner algorithm.

Algorithm 3 (t-spanner [12])

1. sort E by cost c in non-decreasing order
2. $G' \leftarrow (V, E'), E' \leftarrow \emptyset$
3. for each edge (u, v) in E do
4. if $(t * c((u, v)) < d_{G'}(u, v))$
5. $E' \leftarrow (u, v) \cup E'$

Fig. 3. The t-spanner algorithm.

Cai [15] showed that the minimum t-spanner (a t-spanner with the minimum number of edges) is an NP-complete problem. However, asymptotically, the algorithm of Althöfer et al. generates, from a graph $G(V, E)$, a t-spanners whose edge count is in the same order of magnitude as the optimal t-spanner [12].

Fig. 3 presents the t-spanner algorithm of Althöfer et al. [12]. It first sorts, in increasing order, all the edges in G by the edge cost. The edges are examined starting with the shortest. An edge (u, v) is added to the spanner G' if it improves the distance between u and v by at least a factor of t.

To apply the t-spanner algorithm described above would require IDMaps clients to first collect and store all $\mathcal{T}^2$ virtual links advertised by the $\mathcal{T}$ Tracers. It also assumes that once a t-spanner is computed, it will remain static. In reality, we do not expect all IDMaps clients to be able to store $\mathcal{T}^2$ virtual links. As the underlying Internet topology changes, we further expect the set of virtual links that makes up the t-spanner to change from time to time. Hence Tracers continually trace and advertise all $\mathcal{T}^2$ virtual links—albeit at different frequencies, with higher frequencies for those used by the t-spanner and those that are less stable; accordingly, IDMaps clients must be able to examine each new advertisement of a virtual link and continually update their t-spanners. To allow for this incremental update, we have developed an incremental t-spanner algorithm.[5]

B. Tracer-AP Virtual Links

We define an *AP* (Address Prefix) as a consecutive address range within which all assigned addresses are equidistant (with some hysteresis) to the rest of the network. We investigate various alternatives to discovering APs on the Internet in a related work [16]. In this paper, we look at whether it is sufficient for each AP to be traced by a single Tracer. If an AP has more than one path to the rest of the Internet, having a single Tracer tracing to that AP could result in bad distance estimates between this AP and hosts that are not sharing paths with the Tracer. Fig. 4 shows a network of four ISPs and three APs. One Tracer each is placed in ISP1, ISP2, and ISP3, i.e., T1, T2, and T3, respectively. The label on each link denotes the distance of the link. Consider the following scenario. Mirrors M1 and M2 of a service are placed in AP3 and AP2, respectively. Assume that Tracer T1 traces to

[5] Space limitation prevents us from including our incremental t-spanner algorithm in this paper.

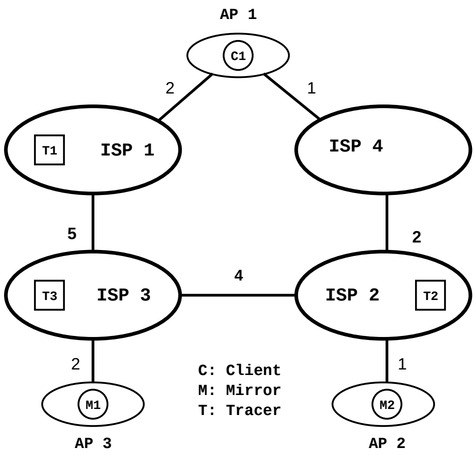


Fig. 4. Network with multiple connections to the Internet.

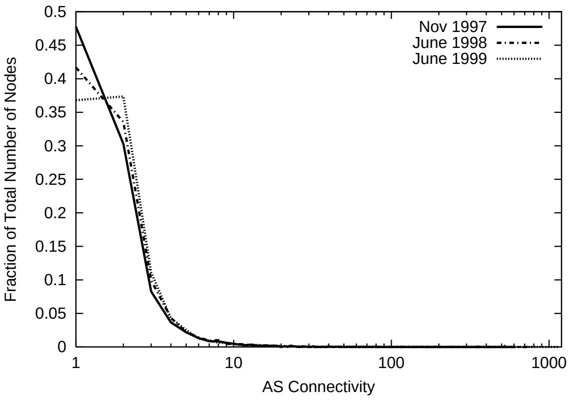


Fig. 5. Distribution of degree of inter-AS connectivity on the Internet.

AP1, T2 traces to AP2, and T3 traces to AP3. Client C1 in AP1 will then be directed to mirror M1 in AP3 instead of M2 in AP2. Had Tracer T2 also traced to AP1, the client would have been directed to M2. We investigate the effect of having more than one Tracer tracing to each AP in Section VI-F.

V. Simulation Setup

To study the various algorithms presented in this paper prior to the deployment of IDMaps on the Internet, we conduct some simulations on generated network topologies. In this section, we give an overview of the three topology generation processes used in this study. Then we describe how we "deploy" IDMaps on the generated topologies. Finally, we describe how the performance metric, P_{app}, is computed.

A. Topology Generation

For the purposes of building distance maps, we model the Internet as a flat network consisting of regions, all hosts within a region are considered equidistant to the rest of the Internet. In our simulated networks, each node represents such a region (AP). To have a rough estimate of the number of APs, we need to take a look at ASs (Autonomous Systems) on the current Internet. As of this writing, there are about 5,300 ASs on the Internet, with about 30% of them announcing only one routing prefix [11]. Fig. 5 shows the distribution of inter-AS connectivity degree observed from BGP peering data collected by the NLANR (National Laboratory for Applied Network Research) for the months of Nov. 97, Jun. 98, and Jun. 99 [17]. Maximum inter-AS connectivity was 590 in Nov. 1997, 812 in Jun. 1998, and 1,161 in Jun. 1999. We make the observation that large ASs, such as those belonging to ISPs and large companies, connect together many networks that are topologically dispersed. From the published ISP backbone data [18], we estimate that large ISPs have about 20 exchange points each. To the first order of approximation, we consider ASs with degree of connectivity less than 20 as APs.

TABLE I

Network diameter and max node degree

Model	Hop Count	E2E Distance	Max Node Degree
Waxman	21	77,655	8
Tiers	35	14,635	20
Inet	18	78,379	24

We use three models to generate network topologies: the Waxman model [19], Tiers [20], and a model based on AS-connectivity ("Inet") observed from data collected on the Internet. All three models work by first placing a given number of nodes, N, on a plane of dimensions s distance units by s distance units. In this paper, we always choose s that is one order of magnitude larger than N. The cost of each edge in the generated network is the Euclidean distance between the two end nodes, the cost of a path between two nodes is the sum of the edge costs. The three models differ in their determination of (1) how nodes are placed on the plane, (2) how many other nodes each node should directly connect to (a.k.a. *node connectivity degree*, or simply, *node degree*), and (3) which edges are selected to form the network. These three models produce networks with very different characteristics, including the node degree distribution, the end-to-end delay distribution, and the network diameter.

We use the three models to generate three 1,000 node networks.[6] Fig. 6 shows the distribution of node degree of the three generated topologies (we will explain the graph labeled "Inet Phase 1" later). Table I lists the diameter and the maximum node degree of each network, and Figs. 7 and 8 show the distribution of hop counts and end-to-end distance between all node pairs on the three networks, respectively. In Fig. 7 (8) the hop counts (e2e distances) are normalized by the diameter of the respective network. We next describe the topology generation process in greater details.

Waxman. The Waxman model is widely used in the literature for generating random topologies. To use this model, we select two parameters: β, the average node degree, and α, the density of short edges in the network. Given N nodes, the Waxman model places them on the plane by randomly generating each node's x and y coordinates from a uniform distribution over the interval $[0, s]$. We then compute L, the maximum Euclidean distance between any two nodes on the network. The probability of having an edge between nodes u and v is then calculated according to the probability $P(u, v) = \beta e^{(-d(u,v)/L\alpha)}$, where $d(u, v)$ is the

[6] We continue to experiment with more networks from each model.

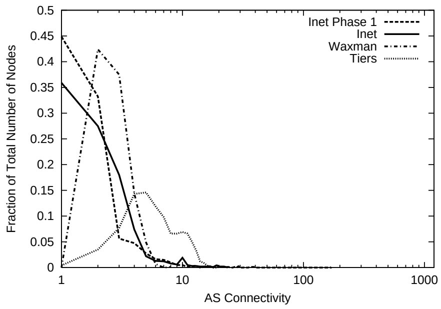


Fig. 6. Distribution of node degree.

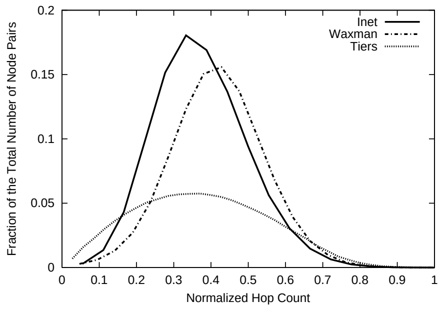


Fig. 7. Distribution of hop count.

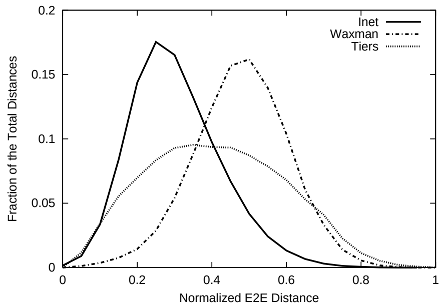


Fig. 8. Distribution of end-to-end distance.

Euclidean distance between u and v. A random number between 0 and 1 is generated. The edge is added if this number is smaller than the computed $P(u, v)$. We use $\alpha = .6$ and $\beta = .0015$, which give us a mean node degree of around 2.2. Finally, a spanning tree is computed, adding edges where necessary, such that the final network is a connected graph.

Tiers. Tiers was designed to generate networks whose topology resembles that of typical internetworks; in particular, Tiers was designed to capture the presence of locality and hierarchy in internetworks. Since we model the Internet as a flat network of APs, we use Tiers to generate networks consisting of only WAN (wide area network) nodes (in Tiers terminology).[7] Tiers places the given N nodes on the $s \times s$ plane ensuring that nodes are not within a radius of $.5s/\sqrt{N}$ of each other. The connectivity degree of each node is randomly distributed with a maximum node degree (WAN redundancy, in Tiers parlance) of r. Fig. 6 shows the distribution of the resulting node degree on our 1,000 node network for $r = 20$. Edges are added by first computing a minimum spanning tree connecting the nodes. Each node is then connected to as many other nodes as specified by its node degree. In selecting a node's neighbors, the ones closest to the node that have not yet met their node degrees are selected first.

Inet. We construct this model based on results reported in [21] and the following empirical observations we made on the inter-AS connectivity data from NLANR:

1. The top most connected ASs form a full-mesh.
2. The most common peers of the top most connected ASs are ASs with connectivity degree of two.
3. The top most connected ASs generally do not directly connect to other ASs with connectivity degree of one, even though one is the most common degree of connectivity.

The above observations about AS connectivities coincide with the findinging of tree depth in [21]. As of June 1st, 1999, the number of ASs that qualify for "the top most connected ASs" using each of the three criteria is 10, 15, and 5, respectively. Fig. 5 indicates an increase in AS connectivity between 1997 and 1999, both in terms of the increase of dual-connected ASs and the increase of maximum inter-AS connectivity. Maximum inter-AS connectivity was 590 in Nov. 1997, 812 in Jun. 1998, and 1,161 in Jun. 1999. Hence we expect the above observations to remain true for the future.

[7] Tiers is also capable of generating hierarchical networks; however the resulting topology looks like a single-backbone ISP network instead of an internetwork of ISPs.

Our Inet model is constructed in two phases. In the first phase, we model AS connectivity by randomly placing $N - \eta\kappa$ nodes on an $s \times s$ plane, we define η and κ in the description of the second phase below. Since our edge selection process (described below) does not add edges based on Euclidean distance, we do not need to ensure that nodes are at least a certain distance apart. We then assign the node degree such that the frequency of each degree occurring agrees with the power law presented in [21]. Edges are then added to simulate the characteristics of inter-AS connectivity we observed from the NLANR data, as follows: first we connect the top τ nodes into a full-mesh; in this paper $\tau = 5$. For these τ nodes, we then connect 10% of their edges to randomly selected nodes with degree of connectivity two. Finally, to achieve full connectivity, we fill the node degree requirement of the remaining nodes, starting from the ones with highest degree of connectivity, by ensuring that each node is either connected to these τ nodes or connected to another node that can reach these τ nodes.

In the second phase, we expand the most connected κ nodes into networks with η nodes each. This is to simulate the observation we made earlier that ASs with the largest degrees of connectivity usually belong to ISPs with geographically dispersed networks. In this paper, we use $\kappa = 10$ for $N = 1,000$, $\kappa = 35$ for $N = 4,200$, and $\eta = 20$ in all cases. We call the nodes generated in the first phase of the Inet model, "phase-1 nodes" and those generated in the second phase, "phase-2 nodes." Reflecting the geographically dispersed nature of networks connected to large ISPs, phase-2 nodes are randomly placed on the plane. Connectivities between phase-2 nodes replacing the same phase-1 node are determined using the Waxman model. These phase-2 nodes also inherit, from the phase-1 node they replace, and divide among themselves, connectivity to the rest of the network. In Fig. 6, the graph labeled "Inet phase 1" shows the distribution of the node degree of the $N - \eta\kappa$ phase-1 nodes, the graph labeled "Inet" shows the distribution of the node degree of all N nodes, after the second phase expansion; in this case $N = 1,000$. The maximum node degree before the second phase expansion is 170, afterwards, 24.

B. IDMaps Infrastructure

Once a network is generated, we build an IDMaps infrastructure on it. In this section, we describe how the various Tracer placement and distance map computation algorithms and heuristics are implemented.

Tracer Placement. In Section III-B, we list four Tracer placement heuristics: *Stub-AS, Transit-AS, Mixed*, and *Idealized*. Given $\mathcal{T}$ Tracers, to implement *Stub-AS* Tracer placement, we pick $\mathcal{T}$ nodes with the lowest degrees of connectivity to host Tracers. Conversely, for *Transit-AS* placement, we pick $\mathcal{T}$ nodes with the highest degrees of connectivity. We implement *Mixed* Tracer placement by giving equal probability to all nodes on the generated network to host a Tracer. We implement the *Idealized* placement by computing Tracer placement using the algorithms described in Section II.

Distance Map Computation. A distance map consists of two parts: Tracer-Tracer virtual links and Tracer-AP virtual links. Each Tracer advertises the virtual links it traces. Clients of IDMaps collect these advertisements and build a distance map out of them. We do not simulate virtual link tracing and advertisement in this study, and we only simulate a single IDMaps client. The simulated IDMaps client has a full list of Tracers and their locations. The Tracer-Tracer part of the distance map is computed either assuming a full-mesh among all Tracers, or by executing the original t-spanner algorithm shown in Fig. 3.

Each AP (node) can be traced by one or more Tracers. When each AP is traced by a single Tracer, the Tracer closest to an AP is assigned to trace to the AP. If an AP is to be traced by more than one Tracer, Tracers are assigned to the AP in the order of increasing distance. In our simulations, we assume all edges are bidirectional, and paths have symmetric and fixed costs. We study the effect of measurement error and stability on IDMaps' performance in a related work [16].

Once a distance map is built, the distance between two nodes, A and B is estimated by summing up the distance from A to its nearest Tracer TA, the distance from B to its nearest Tracer TB, and the distance between TA and TB. When a full-mesh is computed between Tracers, the TA to TB distance is exactly the length of the shortest path between them on the underlying network. Otherwise, they are computed from the t-spanner. If A and/or B have multiple Tracers tracing to them, the distance between A and B is the shortest among all combinations of Tracer-AP and Tracer-Tracer distances for the Tracers and APs involved.

C. *Performance Metric Computation*

Recall that we evaluate the efficacy of IDMaps by computing the "correctness" of closest server selection using the distance map provided by IDMaps against selection based on the underlying topology (P_{app}). Considering that on the Internet a client served by a server 15 ms away would probably not experience a perceptible difference from being served by a server 35 ms away, or that a server 200 ms away will not appear much closer than one 150 ms away, we consider IDMaps' server selection correct as long as distance between the client and the closest server determined by IDMaps is within a factor of λ times the distance between the client and the actual closest server (in this paper, we use $\lambda = 2$).

To compute the efficacy of IDMaps, we first place 3 server mirrors on our simulated network. We place the mirrors such that the distance between any two of them is at least 1/3 the diameter of the network. We consider all the other nodes on the network as clients to the server. We then compute for each client the closest server according to the distance map obtained from IDMaps, and according to the actual topology. For a given 3-mirror placement, we compute P_{app} as the percentage of correct IDMaps' answers over total number of clients, with the correctness criterion defined above. We repeat this experiment for 1,000 different 3-mirror placements, obtaining 1,000 P_{app} values. In the next section, we present our simulation results by plotting the complementary distribution function[8] of these P_{app} values. We plan to study the performance of IDMaps when the client population is not uniformly distributed across all nodes in a related work [16].

TABLE II
SIMULATION PARAMETERS

Topology	Placement	$\mathcal{T}$	T-T Map	T/AP
Waxman	Stub-AS	10	full-mesh	1
Tiers	Transit-AS	20	2-spanner	2
Inet	Mixed	40	10-spanner	3
	Min K-center	100		
	k-HST			

VI. SIMULATION RESULTS

Table II summarizes the parameters of our simulations. The heading of each column specifies the name of the parameter, and the various values tried are listed in the respective column. The column labeled "Topology" lists the three models we use to generate random topologies. The "Placement" column lists the Tracer placement algorithms and heuristics. The "$\mathcal{T}$" columns lists the number of Tracers we use on 1000-node networks. The "T-T Map" column lists the methods used to compute the Tracer-Tracer part of the distance map. The "T/AP" column lists the number of Tracers tracing to an AP. We experimented with almost all of the 540 possible combinations of the parameters on 1,000 node networks and several of them on 4,200 node networks. The major results of our study are:

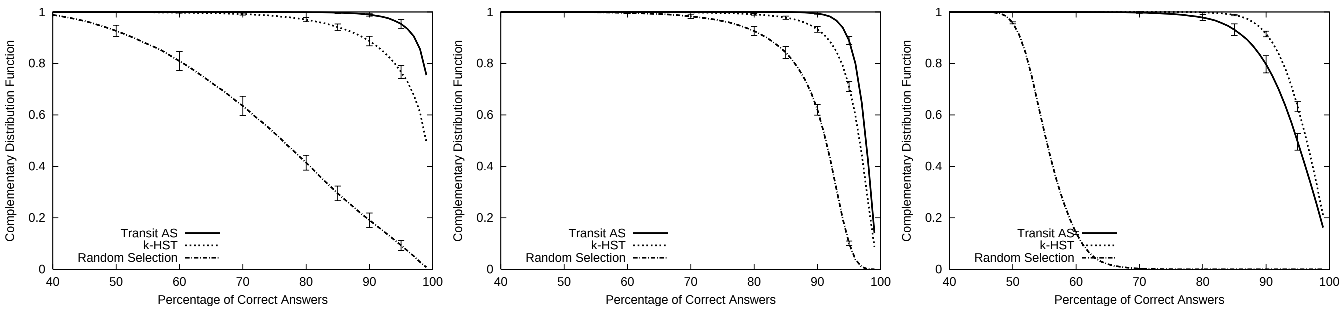
1. Mirror selection using IDMaps gives noticeable improvement over random selection.
2. Network topology can affect IDMaps' performance.
3. Tracer placement algorithms that rely on knowing the network topology do not always outperform heuristics that do not.
4. Adding more Tracers gives diminishing return.
5. Number of Tracer-Tracer virtual links required for good performance can be O($\mathcal{T}$) with a small constant multiplier.
6. Increasing the number of Tracers tracing to each AP improves IDMaps' performance with diminishing return.

These results apply to both the 1000-node and 4200-node networks. We present simulation data substantiating each of the above results in the following subsections.

A. *Mirror Selection*

Results presented in this subsection are obtained from simulations on a 1,000-node network with topology generated using the Inet model. In all cases, the number of Tracers deployed is 10 (1% of nodes), the distance maps are built by computing full-meshes between the Tracers, with only a single Tracer tracing to each AP.

[8]The complementary distribution function, $F'(x) = 1 - F(x)$, where $F(x)$ is the cumulative distribution function of the random variable x.



a. Inet b. Waxman c. Tiers

Fig. 9. Mirror selection on 1,000-node network with 10 Tracers.

We compare the results of randomly selection against selection using the distance map generated by IDMaps. The metric of comparison is P_{app}. Each line in Fig. 9a shows the complementary distribution function of 1,000 P_{app} values as explained in the previous section. For example, the line labeled "Transit-AS" shows that when mirrors are selected based on the distance map computed from Tracers placed by the Transit-AS heuristic, the probability that 80% of all clients will be directed to the "correct" mirror is 100% (recall our definition of correctness from the previous section); however, the probability that 98% of all clients will be directed to the correct mirror is only 85%. We start the x-axis of the figure at 40% to increase legibility. The line labeled "k-HST" is the result for idealized IDMaps when the k-HST algorithm is used to place Tracers. The k-HST algorithm requires knowledge of the topology (see Section II-A). The line labeled "Random Selection" is the result when mirrors are randomly selected without using a distance map. As expected, it performs well for less than 40% correctness and the performance deteriorates beyond 60% correctness. Mirror selection using distance maps outperforms random selection regardless of the Tracer placement algorithm. We ran the same experiment with Inet topologies generated using 31 different random seeds and computed the 95% confidence interval (shown as error bars) for the tail distribution in Fig. 9a. We include only the best and worst performing Tracer placement algorithms in Figs. 9 for legibility of the graphs. The relative performance of the various placement algorithms is presented in Section VI-C.

B. Effect of Topology

Figs. 9b and 9c show the results of running the same set of simulations as in the previous section, but on topologies generated from the Waxman and Tiers models, respectively. Again, the error bars on each figure shows the 95% confidence interval computed from 31 randomly seeded topologies. While mirror selection using a distance map provides better performance than random selection in all cases, performance on the Tiers generated topology exhibit a qualitatively different behavior than those on the other two topologies. For example, the Transit-AS heuristic gives better IDMaps performance than the k-HST algorithm on topologies generated from the Inet and Waxman models, but not so on the topology generated from Tiers.

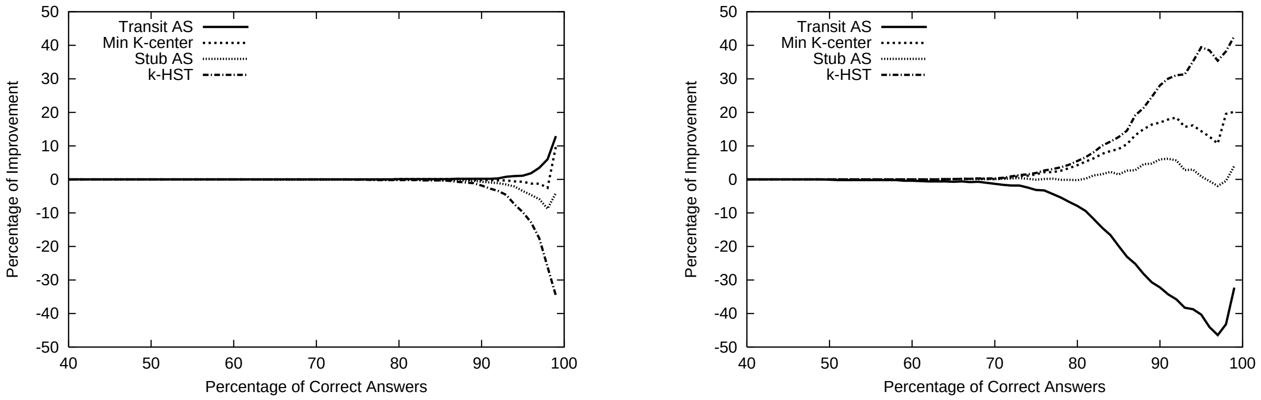
We offer a hypothesis for the relatively poor performance of random mirror selection on Tiers topology. Fig. 8 shows that almost all the end-to-end distances in Inet generated network fall between 20% and 60% of the network diameter. When we randomly pick two distances from this network, it is highly likely that they will fall within this range. Consequently, one distance will be no more than 3 times longer than the other. So given our definition of the performance metric, even the random selection can give acceptable performance. As can be seen by comparing Fig. 9b against Figs. 9a and 9c, this is more evident in the network generated from the Waxman model, where the distances fall between 30% and 70% of the network diameter. However, the distance distribution for the Tiers topology is much more dispersed, and the range is between 10% and 70% of the diameter. It is much harder for two randomly picked distances to be close within a factor 3. This is corroborated by the poor results "Random Selection" returns. We note again that despite the significant differences in the three models, IDMaps is able to provide noticeable improvements in mirror selection in all three cases.

C. Performance of Placement Algorithms

To compare the relative performance of the various Tracer placement algorithms and heuristics, we repeat the same simulations as in the previous two subsections, once for each placement algorithm. Then using the complementary distribution function of the P_{app} values obtained from running the Mixed placement algorithm as the baseline, we compute the improvement of the placement algorithms relative to Mixed placement. The results are presented in Figs. 10a and 10b for networks generated using the Inet and Tiers model, respectively. For example, Fig. 10b shows that for the Tiers topology, while the Min K-center, k-HST, and Stub-AS placement algorithms outperform the Mixed algorithm, the Transit-AS placement algorithm underperforms it. While no placement algorithm convincingly outperform any other, it is interesting to note that the mere existence of IDMaps, regradless of the placement algorithm, already provides significant improvement over random selection.

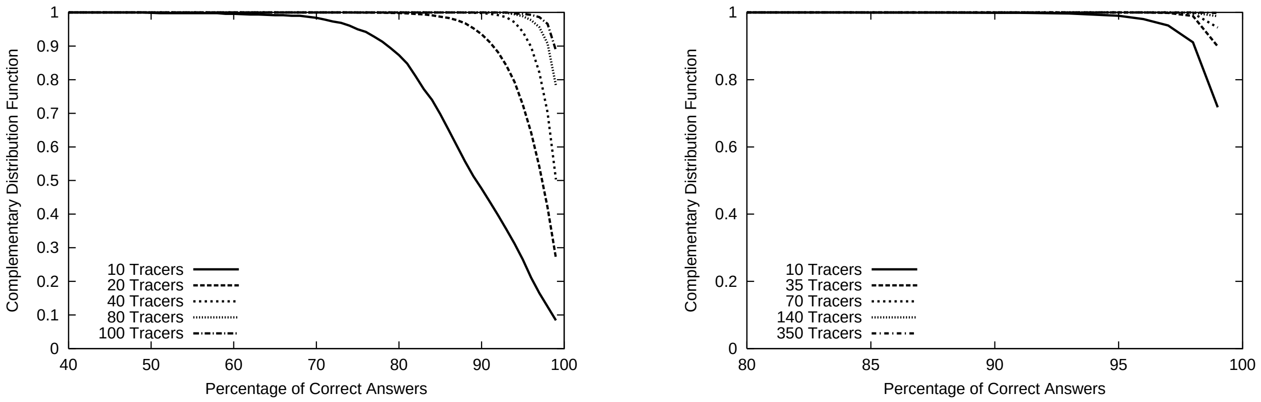
D. Having More Tracers

In this subsection, we study the effect of increasing the number of Tracers on IDMaps' performance. Fig. 11a shows the results of running the Transit-AS placement algorithm on a 1,000-node network generated using the Tiers model. Increasing the number of Tracers from 10 to 20 improves performance perceptibly, with diminishing improvements for further increases. Comparing Fig. 11a against Fig. 9c from Section VI-B we see that increasing the number of Tracers from 10 to 20 makes the performance IDMaps using the Transit-AS placement algorithm



a. Inet   b. Tiers

Fig. 10. Improvement of placement algorithms over the “Mixed” algorithm on 1,000-node network, 10 Tracers.



a. 1,000-node Tiers network   b. 4,200-node Inet network

Fig. 11. Mirror selection using IDMaps with varying number of Tracers.

comparable to that using the k-HST algorithm with 10 Tracers.

Fig. 11b shows the results of running the Transit-AS placement algorithm on a 4,200-node network generated using the Inet model. Again, we see a perceptible improvement in IDMaps performance when the number of Tracers increases from 10 to 35, with diminishing improvements for further increases. Also of significance is that having only .2% of all nodes serving as Tracers already provides correct answer 90% of the time with very high probability. Overall, it is clear that we do not necessarily need a large scale IDMaps deployment to realize an improvement in the current metric of interest, P_{app}.

E. Distance Map Reduction

In all the simulations reported so far, the distance maps are built by computing full-mesh Tracer-Tracer virtual links. Figure 12 shows the results of running the Transit-AS algorithm to place 100 Tracers on a 1,000-node network generated using the Inet model, with Tracer-Tracer virtual links computed as a full-mesh and as t-spanners. For $t = 2$, there is no perceptible difference in performance; for $t = 10$, the performance is worse. Qualitatively similar results are observed for topologies generated by the Waxman and Tiers models, with worse performance for $t = 10$ in the Tiers case.

Using a t-spanner in place of a full-mesh can significantly reduce the number of Tracer-Tracer virtual links that must be traced, advertised, and stored. Table III shows that for all the

TABLE III

NUMBER OF VIRTUAL LINKS USED BY t-SPANNER.

Placement	2-spanner	10-spanner
Inet		
Stub	628	198
Mixed	520	200
Transit	434	198
Min K-center	402	198
Waxman		
Stub	654	198
Mixed	466	198
Transit	386	202
Min K-center	434	196
Tiers		
Stub	268	202
Mixed	264	200
Transit	262	198
Min K-center	266	202

topologies we experimented with, the number of virtual links used by both 2- and 10-spanners are O($\mathcal{T}$) with a small constant multiplier. In contrast, the number of virtual links required to maintain a full-mesh for $\mathcal{T} = 100$ is 4,950 edges.

F. Multiple Tracers per AP

In all our simulations so far we have assumed that only a single Tracer traces each AP. On the 1000-node network generated using the Waxman model, we experimented with having two and three Tracers tracing to each AP. Tracers are placed us-

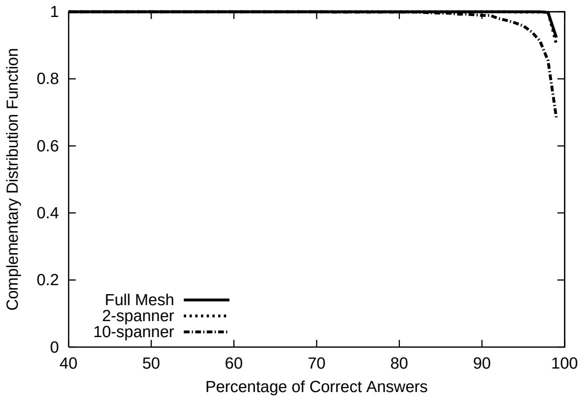


Fig. 12. Effect of t-spanner on 1,000-node Inet network with 100 Tracers.

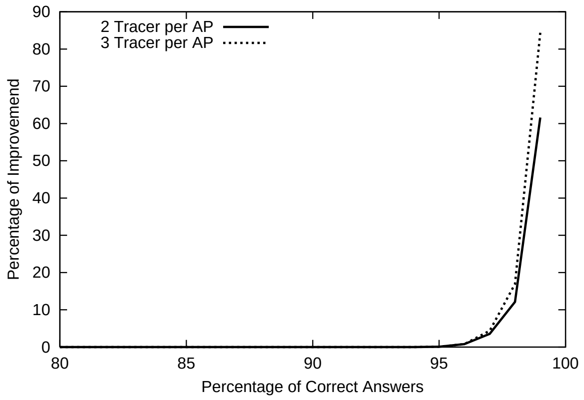


Fig. 13. Mirror selection on 1,000-node Waxman network, 100 Tracers, increasing number of Tracers per AP.

ing the Transit-AS algorithm, and a full-mesh is computed for Tracer-Tracer virtual links. Using the one Tracer per AP performance as the baseline, we compute the percentage improvement of increasing the number of Tracers per AP. We only consider up to 3 Tracers per AP since currently 85% of ASs on the Internet have degree of connectivity of at most 3 (see Fig. 5). Figs. 13 shows the results for IDMaps with 100 Tracers. The figure shows clearly the diminishing return of having multiple tracers per AP.

VII. Conclusion

It has become increasingly evident that some kind of distance map service is necessary for distributed applications on the Internet. However, the question of how to build such a distance map remains largely unexplored. In this paper, we tackle the question of how a measurement network can be placed on the Internet to collect distance information.

In the context of closest server selection for clients, we showed that, significant improvement over random selection can be achieved using placement heuristics that do not require the full topological knowledge. In addition, we showed that the IDMaps overhead can be minimized by applying spanners to the Tracer-Tracer virtual links, which results in linear measurement overhead with respect to the number of Tracers. Furthermore, we looked at some theoretical approaches to the well-known center placement problem, which can potentially provide a theoretical basis for our on-going research. Overall this study has provided positive results to show that an Internet distance map service is indeed useful.

References

[1] 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*, Mar. 1999.

[2] Akamai Inc., "Freeflow," URL: http://www.akamai.com/, 1998.

[3] Sandpiper Networks, "Footprint 2.0," URL: http://www.sandpiper.net/, 1998.

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

[5] Yair Bartal, "Probabilistic approximation of metric space and its algorithmic applications," in *37th Annual IEEE Symposium on Foundations of Computer Science*, Oct. 1996.

[6] Baruch Awerbuch and Yuval Shavitt, "Topology aggregation for directed graphs," in *Third IEEE Symposium on Computers and Communications*, June 1998, pp. 47 – 52.

[7] Michael R. Garey and David S. Johnson, *Computers and Intractability*, NY, NY: W.H. Freeman and Co., 1979.

[8] Vijay Vazirani, *Approximation Methods*, Springer-Verlag, 1999.

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

[10] Paul Francis, "Host proximity service (hops)," URL: http://www.ingrid.org/hops, July 1998.

[11] T. Bates, "The cidr report," URL: http://www.employees.org/˜tbates/cidr-report.html, June 1998.

[12] I. Althöfer, G. Das, D. Dopkin, D. Joseph, and J. Soares, "On sparse spanners of weighted graphs," *Discrete and Computational Geometry*, vol. 9, pp. 81 – 100, 1993.

[13] David Peleg and Eli Upfal, "A tradeoff between space and efficiency for routing tables," in *20th ACM Symposium on the Theory of Computing*, May 1988, pp. 43 – 52.

[14] David Peleg and Alejandro A. Schäffer, "Graph spanners," *Journal of Graph Theory*, vol. 13, no. 1, pp. 99 – 116, 1989.

[15] Leizhen Cai, "NP-completeness of minimum spanner problems," *Discrete Applied Mathematics*, vol. 48, pp. 187 – 194, 1994.

[16] IDMaps Project, "Work in progress," URL: http://idmaps.eecs.umich.edu/, 1999.

[17] NLANR, "Measurement and operations analysis team," URL: http://moat.nlanr.net/, 1997-1999.

[18] Boardwatch, "Directory of internet service providers," URL: http://boardwatch.internet.com/isp/, 1999.

[19] Bernard M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[20] K. Calvert, M.B. Doar, and E.W. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, June 1997.

[21] Christos Faloutsos Michalis Faloutsos, Petros Faloutsos, "On power-law relationships of the internet topology," *Proc. of ACM SIGCOMM*, Aug. 1999.