Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space

Bo Zhang†, T. S. Eugene Ng†, Animesh Nandi†‡, Rudolf Riedi$, Peter Druschel‡, Guohui Wang†
†Rice University, USA ‡Max Planck Institute for Software Systems, Germany $EIF-Fribourg, Switzerland

***Abstract*—Understanding the characteristics of the Internet delay space (i.e., the all-pairs set of static round-trip propagation delays among edge networks in the Internet) is important for the design of global-scale distributed systems. For instance, algorithms used in overlay networks are often sensitive to violations of the triangle inequality and to the growth properties within the Internet delay space. Since designers of distributed systems often rely on simulation and emulation to study design alternatives, they need a realistic model of the Internet delay space.**

In this paper, we analyze measured delay spaces among thousands of Internet edge networks and quantify key properties that are important for distributed system design. Our analysis shows that existing delay space models do not adequately capture these important properties of the Internet delay space. Furthermore, we derive a simple model of the Internet delay space based on our analytical findings. This model preserves the relevant metrics far better than existing models, allows for a compact representation, and can be used to synthesize delay data for simulations and emulations at a scale where direct measurement and storage are impractical. We present the design of a publicly available delay space synthesizer tool called DS^2 and demonstrate its effectiveness.

***Index Terms*—Internet delay space, measurement, analysis, modeling, synthesis, distributed system, simulation.**

I. Introduction

Designers of large-scale distributed systems rely on simulation and network emulation to study design alternatives and evaluate prototype systems at scale and prior to deployment. To obtain accurate results, such simulations or emulations must include an adequate model of the *Internet delay space*: The all-pairs set of static round-trip propagation delays among edge networks. Such a model must accurately reflect those characteristics of real Internet delays that influence system performance. For example, having realistic clustering properties is important because they can influence the load balance of delay-optimized overlay networks, and the effectiveness of server placement policies and caching strategies. Having realistic growth characteristics [16] is equally important, because the effectiveness of certain distributed algorithms depends on them. Many distributed systems are also sensitive to the inefficiency of IP routing with respect to delay. Such inefficiency manifests itself as triangle inequality violations in the delay space, and must be reflected in a model as well.

This research was sponsored by the NSF under CAREER Award CNS-0448546, an Alfred P. Sloan Research Fellowship, and by the Texas Advanced Research Program under grant No.003604-0078-2003. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, the Alfred P. Sloan Foundation, the state of Texas, or the U.S. government.

Currently, two approaches are used to obtain a delay model. The first approach, adopted for instance by the p2psim simulator [24], is to collect actual delay measurements using a tool such as King [14]. However, due to limitations of the measurement methodology and the quadratic time requirement for measuring a delay matrix, measured data tends to be incomplete and there are limits to the size of a delay matrix that can be measured in practice. To its credit, p2psim provides a 1740×1740 delay space matrix, which is not a trivial amount of data to obtain.

The second approach is to start with a statistical network topology model (e.g., [44], [47], [8], [10], [18]) and assign artificial link delays to the topology. The delay space is then modeled by the all-pair shortest-path delays within the topology. The properties of such delay models, however, tend to differ dramatically from the actual Internet delay space. This is because these models do not adequately capture rich features in the Internet delay space, such as those caused by geographic constraints, variation in node concentrations, and routing inefficiency.

A delay space model suitable for large-scale simulations must adequately capture the relevant characteristics of the Internet delay space. In addition, the model must have a compact representation, since large-scale simulations tend to be memory-bound. The naive approach of storing 16-bit delay values for all pairs of a 100K node network, for instance, would require 20GB of main memory! Finally, to enable efficient simulation, generating a delay for a given pair of nodes must require very little computation and no disk accesses.

One approach is to build a *structural* model of the Internet, using BGP tables, traceroute, ping and other measurements to capture the routing policies, the topology of the Internet and the associated static link delays [22]. Given such a model, the delay for a given pair of IP addresses can be estimated by adding the link delays on the predicted route through the topology. However, it remains unclear how all the required information (e.g., intra-domain and inter-domain routing policies, Internet topology and link weights) can be accurately obtained and how detailed such a model has to be to preserve the relevant characteristics. On the other hand, many distributed systems (e.g., structured overlays, server selection systems) only need end-to-end delays for their simulations; routing information is not necessary.

Another approach is to build a *statistical* model, designed to preserve the statistical characteristics of a measured Internet delay data set. Unlike a structural model, a statistical model cannot predict the delay between a particular pair of real

Internet IP addresses. For the purposes of distributed systems simulations, however, it suffices that the statistical properties of the model adequately reflect those of the measured delay data. Statistical models lend themselves to a compact representation and can enable efficient generation of delay data at large scale. Since we are primarily interested in enabling accurate, efficient, large-scale simulations, we decided to pursue this approach in this paper.

We have measured a sample of the Internet delay space among 3,997 edge networks. We then characterize the measured sample with respect to a set of properties that are relevant to distributed system design. Based on the analytical findings, we develop the methods and tool to model and synthesize artificial Internet delay spaces. A synthesized delay space model has a compact $O(N)$ representation (as opposed to the $O(N^2)$ matrix representation) and adequately preserves the relevant characteristics of the Internet delay space. We make two primary contributions in this work:

- We quantify the properties of the Internet delay space with respect to a set of statistical, structural, and routing metrics relevant to distributed systems design. This leads to new insights into Internet delay space characteristics that may inform future work.
- We develop a set of building block techniques and a publicly available tool called DS^2 to model and synthesize the Internet delay space compactly, while accurately preserving the relevant metrics. The compact representation enables accurate and efficient simulations at large scale. We show the benefits of DS^2 through several applications.

II. Methodology and Models

We begin by describing our measurement methodology and the existing delay space models we use in this study.

A. Measured Internet Delay Space

We use the King tool [14] to measure the all-pairs round-trip static propagation delays among a large number of globally distributed DNS servers, where each server represents a unique domain and typically one edge network. To our best knowledge, the King tool is the only tool that can accurately measure the delays among a large number of DNS servers. In order to measure the delay between two DNS servers D_1 and D_2, the King tool first measures the amount of time it takes to issue a recursive query to D_1 for a name whose authoritative name server is D_2 (the time it takes is denoted as $T(King, D_1, D_2)$), and then it measures its delay to D_1 by using an iterative DNS query (this measured delay is denoted as $T(King, D_1)$). By subtracting $T(King, D_1)$ from $T(King, D_1, D_2)$, it can obtain the estimated delay between D_1 and D_2. To choose DNS servers, we start with a list of 100,000 random IP addresses drawn from the prefixes announced in BGP as published by the Route Views project [31]. For each IP address i, we perform a reverse DNS lookup to determine the associated DNS servers. Each reverse lookup returns a set of DNS servers S_i. We keep only the DNS server sets in which at least one server supports recursive queries, since King requires it. If two DNS server sets S_i and S_j overlap, then only one of the two sets is kept since they do not represent distinct domains. If there is more than one server in a set, the set is kept only if all the servers in the set are topologically close. We check this by performing traceroutes from our machine to all the servers in the set to make sure the minimum delays to them differ less than 5% and the traceroute paths have at least 90% overlap with each other. By making sure the servers in the set are physically co-located, we ensure different measurement samples are measuring the same network. Among the remaining DNS server sets, we choose one server per set that supports recursive query. We then use 5,000 such DNS servers to conduct our measurements.

To ensure the subsequent analysis is based on accurate data, we adopt a fairly stringent methodology. We measure the round-trip delay between two DNS servers, D_1 and D_2, from both directions by using either server as the recursive server. For each direction, we make up to 50 attempts to measure the direct delay to D_1 and the recursive delay to D_2 via D_1, and up to 50 attempts to measure the direct delay to D_2 and the recursive delay to D_1 via D_2. At least 20 measurement samples must be obtained in each case. The minimum value across the samples is used as the propagation delay. After the subtraction step, if the delay is greater than 2 seconds (most likely the recursive delay measurement is inflated too much) or smaller than 100 microseconds (most likely the direct delay measurement is inflated too much), it is discarded. Also, if the obtained round-trip delay between D_1 and D_2 measured in each direction disagrees by more than 10%, we consider the measurement problematic and then discard it. Finally, we remove data from DNS servers that are consistently failing to provide valid measurements. After we assemble the delay space matrix, if any row/column has more than 25% of the values missing, the entire row/column is removed.

We collected the measurements in October 2005. Among the collected 5,000×5,000 delay data, 16.7% have insufficient measurements samples, 8.1% have inconsistent samples, 0.16% are smaller than 100 microseconds, and 0.51% are larger than 2 seconds. After removing suspicious measurement values, the remaining delay matrix has 3,997 rows/columns with 13% of the values in the matrix unavailable. To characterize the distribution of the missing values, we partition the delay matrix into its three largest clusters. These clusters correspond to IP hosts in North America, Europe and Asia. We find that the percentage of missing values are distributed as follows:

From/To	North America	Europe	Asia
North America	14%	11%	12%
Europe	11%	15%	11%
Asia	12%	11%	18%

To understand the properties in the data set under scaling, we consider four different random sub-samples of the measured data with the sizes 800, 1,600, 2,400 and 3,200. Then, for each sub-sample size, we consider five random sample. Results presented in this paper are averages over the five samples.

The data set has some limitations. First, the measurements are among DNS servers. The data set thus represents the delay space among edge networks in the Internet. No explicit measurements were collected among hosts *within* a local area network. Therefore, this study addresses only the delay space properties among edge networks in the wide area, but not the delay space properties within a local area network. Secondly, to increase our confidence in the data, we have discarded questionable measurements. We therefore proceed with the assumption that the missing delay values do not have significantly different properties than the available data.

B. Topology Model Delay Spaces

We also generate delay matrices based on existing topology models and compare them against the measured Internet delay space. The two generators we use are Inet [45] and GT-ITM [47]. The Inet generator creates a topology that has power-law node degree distribution. The GT-ITM generator is used to generate a topology based on the Transit-Stub model. We include the Inet and GT-ITM topology models in this study because they are often used in distributed system simulations.

For Inet, we create a 16,000-node topology. To generate the delays, we use the standard method of placing nodes randomly in a plane and then use the Euclidean distance between a pair of connected nodes as the link delay. All-pairs shortest delay routing is then used to compute end-to-end delays. Finally, we extract the generated delays among the 5,081 degree-1 nodes in the graph in order to model the delays among edge networks. No triangle inequality violations are introduced. For GT-ITM, we create a 4,160-node transit-stub topology. Note that GT-ITM annotates links with routing policy weights and artificial delays. Shortest path routing is performed over the topology using routing policy weights as the link costs. End-to-end delays are then computed by summing the artificial link delays along the selected paths. Some triangle inequality violations are then introduced artificially in the resulting delay space. Finally, we extract the delays among 4,096 stub routers to model the delays among edge networks.

We scale the delays in the two artificial delay matrices such that their average delay matches the average delay in the measured delay data. This constant scaling does not affect the structure of the generated delay spaces. We do this only to simplify the presentation of results.

III. Internet Delay Space Analysis

In this section, we first identify a set of metrics that are known to significantly influence the performance of distributed systems. Then, we analyze measured delay data with respect to these and other statistical and structural properties. The results give insight into the characteristics of the Internet delay space, and they inform the design of an appropriate model.

A. Systems-Motivated Metrics

The metrics presented below are known to strongly influence distributed system performance and capture a wide range of important issues in distributed system design and evaluation.

Global clustering - This metric characterizes clustering in the delay space at a macroscopic level. For instance, the continents with the largest concentration of IP subnetworks (North America, Europe and Asia) form recognizable clusters in the delay space. This global clustering structure is, for instance, relevant to the placement of large data centers and web request redirection algorithms (e.g., [28]).

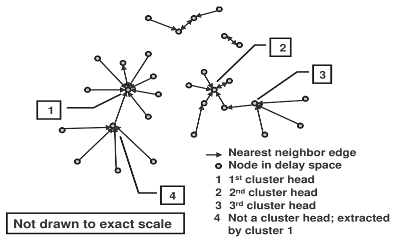


Fig. 1. Nearest neighbor directed graph analysis technique.

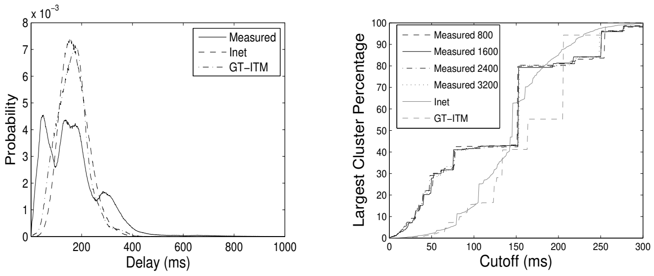
Our algorithm to determine the global clustering works as follows. Given N nodes in the measured delay data, it first treats each node as a singleton cluster. Then it iteratively finds two closest clusters to merge. The distance between two clusters is defined as the average distance between the nodes in the two clusters. A cutoff value determines when to stop the merging process. If the distance between the two closest clusters is larger than the cutoff, the merging process stops. By varying the cutoff value and monitoring the resulting cluster sizes, the global clustering properties can be determined.

Local clustering - This metric characterizes clustering in the delay space at the local level. It is based on analyzing the in-degree distribution of the directed graph formed by having each node point to its nearest neighbor in the delay space. Moreover, we use the graph to identify a set of local cluster heads (or centers). We select the node with the highest in-degree as a local cluster head and remove it and its immediate children from the graph. This step is applied repeatedly to identify the next local cluster head until no more nodes remain. Since a local cluster resembles a star graph, we sometimes simply call it a star. The process is illustrated in Figure 1. The importance of the local cluster heads will become clear in subsequent sections.

Local clustering is relevant, for instance, to the in-degree and thus the load balance among nodes in delay-optimized overlay networks (e.g., [5]). For example, dense local clustering can lead to an overlay node having an unexpectedly high number of neighbors and can potentially create a load imbalance in the overlay.

Growth metrics - Distributed nearest neighbor selection is a hard problem, but efficient algorithms have been identified to solve the problem for growth-restricted metric spaces [16]. These algorithms are used, for instance, in Tapestry [49] and Chord [40] to select overlay neighbors. Growth-constrained metric spaces satisfy the property that for any node i and distance r, the number of nodes within distance $2r$ of i, denoted as $B_i(2r)$, is at most a constant factor larger than the number of nodes within distance r of i, denoted as $B_i(r)$. We characterize the growth properties of a delay space by evaluating the function $B(2r)/B(r)$.

A related metric is the $D(k)$ metric. Let $d(i,k)$ be the average delay from a node i to its k closest nodes in the delay space and N be the set of nodes, then $D(k) = \frac{1}{|N|}\sum_{i \in N} d(i,k)$. Structured overlay networks like Chord, Tapestry and Pastry employ proximity neighbor selection (PNS) to reduce the



(a) (b)

Fig. 2. Global clustering properties. (a) Delay distribution. (b) Clustering.

expected delay stretch S, i.e., the ratio of the delay of an overlay route over the direct routing delay averaged over all pairs of nodes [13][4][29][5]. We choose to include the $D(k)$ metric because analysis has shown that in Tapestry and Pastry, the expected delay stretch S in the overlay can be predicted based on the function $D(k)$ [5].

Triangle inequality violations - The triangle inequality states that given points x, y and z, the distance d_{ij} between points i and j satisfies $d_{xz} \leq d_{xy} + d_{yz}$. The Internet delay space, however, does not obey the triangle inequality, since Internet routing may not be optimal with respect to delay. Unfortunately, many distributed nearest neighbor selection algorithms rely on the assumption that the triangle inequality holds (e.g., [32] [16] [43]). Thus, it is important to understand the characteristics of the violations in the Internet delay space.

B. Analysis Results

We begin with a comparison of the delay distribution. In Figure 2(a), we can observe that the delay distribution of the measured data set has characteristic peaks at roughly 45 ms, 135 ms, and 295 ms. This suggests that the nodes form clusters in the data. In contrast, the delay distributions for the topology models do not indicate such behavior. Clearly, there are rich features in the Internet delay space that are not captured in the delays derived from these topology models.

To quantify the global clustering properties in the measured data set, we apply the described global clustering algorithm and plot the percentage of nodes in the largest cluster against different clustering cut-off thresholds in Figure 2(b). Regardless of the sample size, the largest cluster's size increases sharply at cutoff values 155 ms and 250 ms. These sharp increases are caused by the merging of two clusters at these thresholds. The steps suggest that there are three dominant clusters. By setting the threshold to 120 ms, nodes can be effectively classified into the three major clusters that account for 45%, 35% and 9% of the nodes, respectively. By using traceroute and the WHOIS database, we find that 94.9% of the nodes in the first cluster are from North America, 99.4% of the nodes in the second cluster are from Europe and 99.5% of the nodes in the third cluster are from Asia. The remaining 11% nodes have large delays to other nodes either because they are located in other continents (e.g., Oceania, South America and Africa) or because they have large access delays. These global clustering properties can be used to guide the global placement of servers and the design of load-balancing algorithms. In contrast, there is no clear clustering structure in the Inet model. The clustering structure of the GT-ITM model does not resemble that of the measured data.

Sample size	# of samples	# Cluster heads	Avg. proportion	Variance
800	20	179	22.38%	4.13%
1,600	20	356	22.25%	4.21%
2,400	20	528	22.00%	4.26%
3,200	20	703	21.97%	4.56%
3,997	1	884	22.12%	0

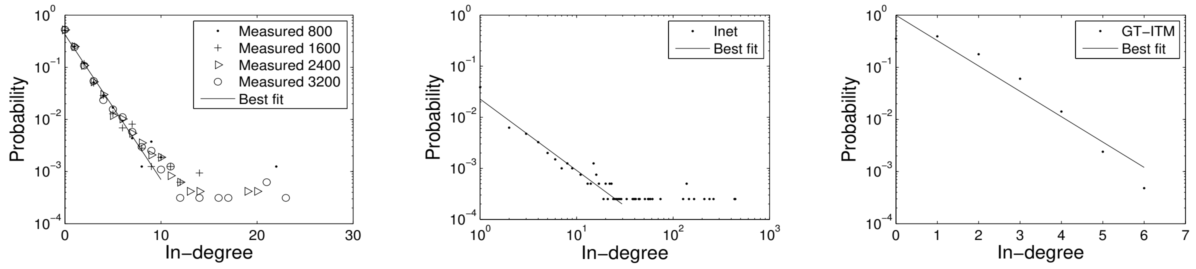
TABLE I
AVERAGE PROPORTION OF NODES CLASSIFIED AS CLUSTER HEADS.

The global clustering reveals the coarse-grained structure of the delay space. To understand the fine-grained structure, we conduct the nearest neighbor directed graph analysis on the measured data. We emphasize that these results characterize the properties among edge networks in the Internet; they *do not* characterize the properties among end hosts within local area networks. Figure 3(a) shows the in-degree distributions for different sample sizes. Observe that the in-degree distribution for the measured data has an exponential decay (note the log-linear scale). Interestingly, we discover that the distribution is consistent across different sample sizes. If a straight line is fitted over 99.9% of the distribution (i.e., ignoring the 0.1% of nodes with the largest in-degrees), the line has a y-intercept of -0.8565 and a slope of -0.6393. These parameters can be used to model the nearest-neighbor in-degree distribution among edge networks in the Internet. In the future, when delay data for hosts within local area networks become available, the model can be hierarchically extended by assigning end hosts appropriately to each edge network in the model.

We classify the nodes into local cluster heads (or star heads) and non-heads using the procedure described in III-A. Table I shows that when the sample size increases 4 times from 800 to 3,200, the average proportion of cluster heads decreases by 0.41%. However, the decreasing rate is very small compared to the variance ($> 4\%$), so we will proceed with the assumption that the proportion is stable across sample sizes for simplicity. Note that local cluster heads are not simply a random subset of nodes because local cluster sizes vary. Moreover, we find that the delays among cluster heads at different sample sizes follow a stable distribution, suggesting a scaling invariant.

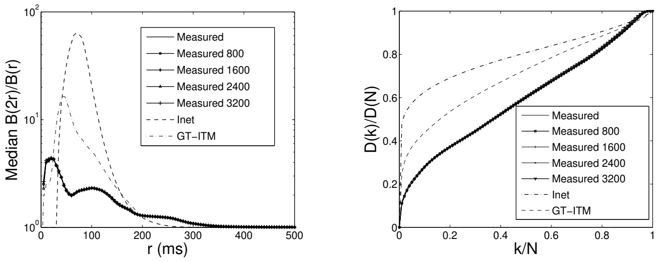
In contrast, as shown in Figure 3(b), the in-degree distribution for the Inet topology follows closely the power-law (note the log-log scale). If a straight line is fitted over 99.9% of the distribution, the line has a y-intercept of -3.7852 and a slope of -1.3970. Thus, the Inet model does not reflect the local clustering properties among edge networks in the measured data. For the GT-ITM topology, as shown in Figure 3(c), the distribution is close to exponential, the best fit line in the log-linear plot has y-intercept of -0.0080 and slope of -1.1611. Thus, this distribution is also different from that found in the measured data. In addition, the maximum in-degree in the measured data is much larger than that in the GT-ITM model.

Next we analyze spatial growth. Figure 4(a) shows the median $B(2r)/B(r)$ growth of the data sets. We plot the median because, unlike the mean, it is insensitive to the extreme outliers and can better characterize the dominant trends. As can be seen, the topology models have far higher peak spatial growth than the measured data (note the log-



(a) (b) (c)

Fig. 3. Local clustering analysis. (a) Exponential-like in-degree distribution for measured data (log-linear scale). (b) Power-law-like in-degree distribution for Inet (log-log scale). (c) Exponential-like in-degree distribution for GT-ITM (log-linear scale).



(a) (b)

Fig. 4. Growth metrics. (a) $B(2r)/B(r)$ (log-linear scale). (b) $D(k)/D(N)$.

linear scale) and have different trends. In the measured data, the initial growth is higher when the ball is expanding within a major cluster. As soon as the ball radius covers most of the nodes within the same major cluster, growth slows down as expected. When the ball radius reaches a size that begins to cover another major cluster, the growth increases again. Eventually most of the nodes are covered by the ball and the growth ratio steadily drops to one. This growth trend in the measured data is invariant across different sample sizes. These findings can help fine tune distributed system algorithms that are sensitive to the ball growth rate. On the other hand, the growth trends in the Inet and GT-ITM topology models do not reflect the structure of the measured data.

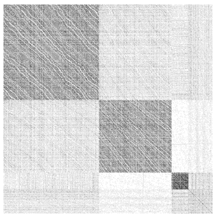
In terms of the $D(k)$ metric, we also observe dramatic differences between topology models and the measured data. Figure 4(b) indicates that in the Inet and GT-ITM topology models, from the perspective of an observer node, there are very few nodes whose delays are substantially smaller than the overall average delay. In contrast, in the measured data, from an observer node, we can find many more nodes whose delays are substantially smaller than the overall average. The bigger the fraction of small delays is, the more likely we can find a close-by node by randomly probing the same number of nodes. Thus, a random probing strategy for finding a close-by neighbor would be much more successful in the real Internet than in the Inet and GT-ITM topology models. This is an example of how using an inadequate delay space model for simulation can potentially lead to misleading results. Finally, it can be observed that the $D(k)$ metric is also invariant across different sample sizes. This empirical $D(k)$ function can be applied to compute the expected delay stretch in the Pastry and Tapestry overlays when deployed over the global Internet [5].

We next analyze the measured data set with respect to properties related to triangle inequality violations. We say that an edge ij in the data set causes a Type 1 triangle inequality violation if for some node k, $\frac{d_{ik}+d_{kj}}{d_{ij}} < 1$, and it causes a Type 2 violation if $\frac{|d_{ik}-d_{kj}|}{d_{ij}} > 1$. Intuitively, better overlay paths can be found for edges that cause Type 1 violations, and edges that cause Type 2 violations can potentially provide short-cut overlay paths.

For each edge, we count the number of Type 1 violations it causes. To show how the triangle inequality violations are distributed over the major clusters, we present a matrix in Figure 5[a] for the measured data. To produce this figure, we first reorganize the original matrix by grouping nodes in the same clusters together. The top left corner has index (0,0). The matrix indices of the nodes in the largest cluster (North America) are the smallest, the indices for nodes in the second largest cluster (Europe) are next, then the indices for nodes in the third largest cluster (Asia), followed by indices for nodes that did not get classified into any of the 3 major clusters.

Each point (i, j) in the plot represents the number of Type 1 violations that the edge ij is involved in as a shade of gray. A black point indicates no violation and a white point indicates the maximum number of violations encountered in the analysis. Missing values in the matrix are drawn as white points.

It is immediately apparent that clustering is very useful for classifying triangle inequality violations. It can be seen that edges within the same cluster (i.e., the 3 blocks along the diagonal) tend to have significantly fewer Type 1 violations (darker) than edges that cross clusters (lighter). Also, the number of violations for edges connecting a given pair of clusters is quite homogeneous. The fact that TIVs exist even within the same cluster is consistent with the effects of policy based Internet routing. On the other hand, the fact that TIVs are much less severe within the same cluster implies that the routing path between two nodes in the same continent is most likely contained within the continent without detouring through other continents. Note that the white wavy lines roughly parallel to the diagonal are simply showing the missing data. Our measurement methodology measures the data in sequences parallel to the diagonal to evenly spread the traffic among the probed DNS servers. Thus, when a measurement station fails, an entire diagonal can be missing. The lines are not straight because whole rows and columns are removed from the data set if they have more than 25% of the values missing. Due to space limitations, we do not include the matrix picture for Type 2 violations, but as expected, the relative shades are the reverse of those in Figure 5[a]. These results imply that, if



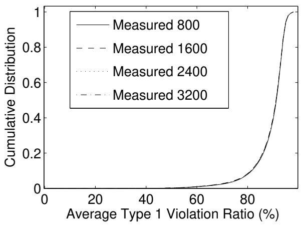


Fig. 5. (a) Clustered Type 1 TIVs (white color is most severe). (b) Type 1 TIV ratio distributions.

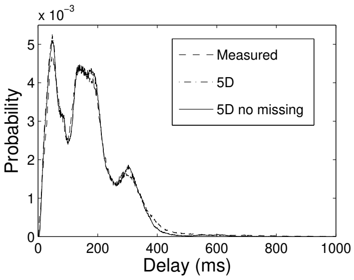


Fig. 6. Delay distribution of the 5D Euclidean map.

two nodes are within the same major cluster, then finding a shorter overlay path is more difficult than when the nodes are in different clusters. Interestingly, observe that it is hardest to find better overlay routes for paths within the Asia cluster, but it is easiest to find better overlay routes for paths across the Asia and Europe clusters.

We show in Figure 5(b) the cumulative distributions of Type 1 violation ratios for different sample sizes. Observe that the distribution is stable across sample sizes. Intuitively, since triangle inequality violation is an inherent property of the inefficiency of Internet routing, the amount of triangle inequality violations observed is not expected to depend on the number of data samples. This invariant is useful in synthesizing the Internet delay space.

IV. Internet Delay Space Modeling and Synthesis

Using measured Internet delay data to drive distributed system simulations allows system designers to evaluate their solutions under realistic conditions. However, there are two potential concerns. First of all, our ability to measure a large portion of the Internet delay space is limited by the time required and the difficulty of dealing with network outages, measurement errors and accidentally triggered intrusion alerts. The second concern is that the $O(N^2)$ storage requirement of a measured delay matrix representation does not scale.

To address these concerns, we develop techniques to model and synthesize realistic delay spaces based on the characteristics of a measured Internet delay space. An overview of the proposed techniques and the relationships between the techniques and the delay space properties are summarized in Table II. The synthesized delay space adequately preserves the relevant properties of the measured data while it has only $O(N)$ storage overhead. The goal is to allow synthesis of realistic delay spaces at scales that exceed our capability to measure Internet delays. Such a tool is valuable for distributed system design and evaluation.

A. Building Block Techniques

Technique 1: Low-dimensional Euclidean embedding - The first technique we use is to model an Internet delay space using a low-dimensional Euclidean embedding. That is, we compute Euclidean coordinates for each node and use Euclidean distances to model the delays in the delay space. Such a Euclidean map contains N low-dimensional Euclidean coordinates and has a scalable $O(N)$ representation.

Several techniques have been proposed to compute a Euclidean embedding robustly (e.g., [23], [7], [34], [6], [19], [42], [39], [38]). Previous studies have shown that a Euclidean embedding can well approximate an Internet delay space with as little as 5 dimensions. However, studies (e.g., [17]) have also shown that such an embedding tends to inflate the small values (< 10 ms) in the delay space significantly.

In order to create a model that also preserves small values, we first use the Vivaldi algorithm to create an Euclidean embedding of the measured delay space, then we explicitly adjust the Euclidean coordinates of nodes as follows. First, extract the set S of all node pairs (i, j) with measured delay d_{ij} less than 10 ms. Next, among these node pairs, select the node pair (m, n) whose Euclidean distance $\hat{d}_{mn}$ in the embedding is smallest. If $\hat{d}_{mn} > 30$ ms, the procedure terminates. Otherwise, the coordinates of nodes m and n are adjusted so that $\hat{d}_{mn}$ becomes identical to d_{mn}. Then, (m, n) is removed from S and the procedure repeats. The 30 ms threshold is empirically derived which allows a similar amount of small delays as in the measured data to be reproduced by the coordinates adjustments.

The effect of this procedure is that small values in the measured delay space that are mildly distorted in the initial Vivaldi Euclidean embedding are well preserved by the final set of adjusted Euclidean coordinates. These adjusted Euclidean coordinates serve as the starting point for our model.

Figure 6 shows the delay distributions for (1) the measured data, (2) all the delays in the 5D Euclidean map, including the modeled values for the missing data, and (3) the delays in the 5D Euclidean map corresponding to the available measured data. We can see that the 5D Euclidean map preserves the distribution of the measured data well. In addition, the modeled values for the missing data do not skew the overall distribution.

However, the Euclidean map is at the same size as the measured data so it still cannot support large scale simulation. In addition, the Euclidean map cannot preserve triangle inequality violations. Finally, it also fails to preserve the high in-degree of some nodes in the nearest neighbor directed graph because a node cannot have a high number of nearest neighbors in a low-dimensional Euclidean space.

To address these limitations of the basic Euclidean model, we use four additional techniques in order to enable synthesis of a larger delay space while preserving the properties lost as a result of the Euclidean embedding.

Technique 2: Euclidean map synthesis - This technique exploits the node-growth properties found in the measured Internet delay space to enable the synthesis of a larger delay space. The node-growth properties relate to the spatial

Techniques	Input	Output	Relationship to delay properties
Euclidean embedding	Measured delay matrix	A Euclidean map	Preserve global clustering properties & growth properties
Euclidean map synthesis	(1) A Euclidean map (2) A scaling factor	A synthesized Euclidean map	Preserve global clustering properties & growth properties
Global distortion	(1) A synthesized Euclidean map (2) Global distortion statistics	Synthesized delays after global distortion	Recreate TIVs characteristics
Local cluster size assignment	A list of synthesized cluster centers	A list of assigned cluster sizes	Preserve the in-degree distribution
Local distortion	(1) A synthesized Euclidean map (2) Local distortion statistics (3) Synthesized delays after global distortion	Synthesized delays after global distortion and local distortion	Recreate local clustering properties

TABLE II
SUMMARY OF THE PROPOSED TECHNIQUES

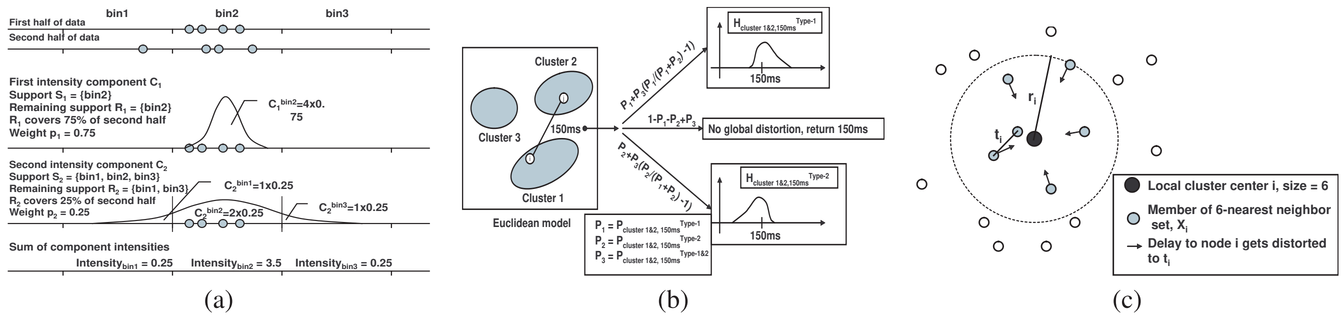


Fig. 7. (a) Euclidean map synthesis technique. (b) Global distortion technique. (c) Local distortion technique.

distribution and density of nodes in a Euclidean map. Given a Euclidean map of an Internet delay space, we seek to capture its node-growth properties so that we can synthesize an artificial map and create realistic structure in the synthesized delay space.

A simple idea is to divide the Euclidean space into equal sized hyper-cubes, count the number of points in each hyper-cube, and use these counts as relative intensities. With appropriate scaling of the relative intensities, one can synthesize an artificial map of a certain size by generating random points in each hyper-cube using an inhomogeneous Poisson point process[1] [20][30]. Indeed, this simple method can mimic the point distribution of the original map and generate a realistic overall delay distribution and global clustering structure. However, this method ignores the node-growth properties in the data. As a result, synthetic points can only appear in hyper-cubes where points were originally found.

To incorporate the node-growth properties, the idea is to introduce uncertainties in the locations of each point and compute intensities that predict the node-growth. The idea is best explained with a simple example in Figure 7(a). In the example, there are 8 points in a 1-dimensional Euclidean space divided into 3 equally sized bins. We randomly divide the points into two halves, the first half happens to lie in bin2, while the other half is spread across bin1 and bin2. Let us define an *intensity component* as a vector of intensities for the bins. We then iteratively compute the i^{th} intensity component C_i using the first half of the points to predict the node-growth observed in the second half of the points. Each component is weighted according to how well it predicts the second half. The location uncertainty of a point in the first half is represented by a Gaussian probability distribution with a certain variance or width. To compute the first intensity component C_1, we place a Gaussian with a small *width* w_1 that represents a low level of uncertainty in the center of each bin and scale it by the number of first half points in the bin. As a result, the 99% bodies of the Gaussians lie within bin2. We call the bins occupied by the 99% bodies of the Gaussians the *support* of the first component, S_1. We also define the *remaining support* of a component to be the support of the current component subtracted by the support of the previous component, i.e., $R_i = S_i \backslash S_{i-1}$. For the first component, R_1 is simply S_1.

The *intensity* I_1 generated by the Gaussian is spread over the three bins as 0, 4, 0, respectively. Now we ask, how well does R_1 cover the second half of the points? If all points in the second half are covered by R_1 then I_1 can account for the node-growth in the second half and we are done. However, in the example, R_1 only covers 75% of the points in the second half. As a result, we weight the intensity I_1 by a factor $p_1 = 0.75$ to obtain the intensity component C_1. Since we have not completely accounted for the node-growth in the second half, we need to increase the location uncertainty and compute the second intensity component C_2. To do so, we use a wider Gaussian (width w_2) for the second iteration. The aggregate intensity is still 4, but this time, it is spread across all 3 bins. Suppose the intensities generated in the 3 bins are 1, 2, 1, respectively. The 99% body of these wider Gaussians occupy all three bins, thus the support of the second component S_2 is the set $\{$bin1, bin2, bin3$\}$. The remaining support R_2 is $S_2 \backslash S_1$, i.e., $\{$bin1, bin3$\}$. The fraction of the second half covered by R_2 is 25%. Thus, the intensity I_2 is weighted by $p_2 = 0.25$ to obtain C_2. This iterative process continues until either all points in the second half are covered by R_i, or when a maximum Gaussian width has been reached. The intensity of each bin is simply the sum of all the intensity components

[1]The number of points lying in any two disjoint sets in space are independent random numbers distributed according to a Poisson law with mean given by the intensity.

C_i. Finally, we repeat the procedure to use the second half to predict the node-growth in the first half and use the average intensity of each bin as the final intensity. In practice, we divide the space into 100 bins in each dimension and vary the Gaussian width from one-tenth to ten times the bin width.

Technique 3: Global distortion - The basic technique to create triangle inequality violations in the Euclidean model is to distort the delays computed from the embedding. Since the frequency of triangle inequality violations in the measured data is relatively small, it suffices to distort only a small subset of node pairs or edges.

Recall that edges between different pairs of global clusters have very different triangle inequality violation behaviors (as can be seen in Figure 5(a)). Thus, we first identify the edges in each pair of clusters that cause violations above a certain severity threshold, and then characterize the distortion distribution for these edges when they are mapped into the Euclidean model, finally we use this same distortion distribution to introduce distortions when delays are generated from the embedding. To ensure that the model always produces the same delay for a given pair of nodes, it uses the node identifiers to generate deterministic pseudo-random distortions. By choosing different severity thresholds, we can vary the number of edges that get distorted in the model and experimentally determine the threshold that best matches the empirical data. An overview of the technique is illustrated in Figure 7(b).

We define a violation severity threshold R. A violation caused by an edge ij is severe if for some node k, $\frac{d_{ik}+d_{kj}}{d_{ij}} < R$ (called Type 1 violation), or if $\frac{|d_{ik}-d_{kj}|}{d_{ij}} > \frac{1}{R}$ (called Type 2 violation). For each global cluster pair g, all edges with the same Euclidean model delay l (rounded down to the nearest 1ms) form a subgroup. For each subgroup (g, l), we compute the fraction of edges in this subgroup that are involved in severe Type 1 violations in the measured data, $P_{g,l}^{\text{Type-1}}$, and a histogram $H_{g,l}^{\text{Type-1}}$ to characterize the real delay distribution of those severe violation edges. Similarly, for Type 2 violations, we compute the fraction $P_{g,l}^{\text{Type-2}}$ and the histogram $H_{g,l}^{\text{Type-2}}$. We also compute the fraction of edges that incur severe Type 1 and Type 2 violations simultaneously, $P_{g,l}^{\text{Type-1\&2}}$. This extra statistical information incurs an additional constant storage overhead for the model.

With these statistics, the delay between node i and j is then computed from the model as follows. Draw a pseudo-random number ρ in [0,1] based on the IDs of i and j. Let the Euclidean distance between i and j be l_{ij} and the cluster-cluster group be g. Based on $P_{g,l_{ij}}^{\text{Type-1}}$, $P_{g,l_{ij}}^{\text{Type-2}}$, $P_{g,l_{ij}}^{\text{Type-1\&2}}$, and using ρ as a random variable, decide whether the edge ij should be treated as a severe Type 1 violation (with probability $P_{g,l_{ij}}^{\text{Type-1}} + P_{g,l_{ij}}^{\text{Type-1\&2}} \cdot (\frac{P_{g,l_{ij}}^{\text{Type-1}}}{P_{g,l_{ij}}^{\text{Type-1}}+P_{g,l_{ij}}^{\text{Type-2}}} - 1)$), or a severe Type 2 violation (with probability $P_{g,l_{ij}}^{\text{Type-2}} + P_{g,l_{ij}}^{\text{Type-1\&2}} \cdot (\frac{P_{g,l_{ij}}^{\text{Type-2}}}{P_{g,l_{ij}}^{\text{Type-1}}+P_{g,l_{ij}}^{\text{Type-2}}} - 1)$), or to return the value l_{ij} without distortion. If the edge ij is treated as a severe Type 1 violation, then we use the histogram $H_{g,l_{ij}}^{\text{Type-1}}$ and ρ to draw a value from the histogram and return that value. Similarly, if the edge is treated as a severe Type 2 violation, then we use the histogram $H_{g,D_{ij}}^{\text{Type-2}}$ instead.

By experimenting with different threshold values R, we have determined that a value of 0.85 produces Type 1 and Type 2 violation distributions similar to those observed in the measured data. This is also the threshold we use in the remainder of this paper.

In order to preserve realistic local clustering properties, we will first assign a cluster size to each local cluster center (Technique 4) and then perform the local distortion technique (Technique 5) to create the needed in-degree.

Technique 4: Local cluster size assignment - Recall the synthesizer knows the empirical exponential distribution of cluster sizes (as computed in Section III-B). Thus, it can draw the cluster sizes from the distribution to approximate the local cluster size distribution of the measured data. What remains unclear is how to assign different cluster sizes to the synthesized cluster centers. Should they be assigned randomly to the cluster centers? Would that be realistic?

It turns out cluster sizes are related to node densities in the measured data. Figure 8 plots the average local density at the cluster centers, i.e., the number of nodes within 15ms of the cluster centers, versus the local cluster size (or star size) for different sample sizes. As can be seen, the size of a local cluster is roughly linearly related to the local node density around the cluster center.

Therefore, the synthesizer assigns cluster sizes as follows. First, the synthesizer computes the local node densities for the synthesized cluster centers and ranks them according to the densities. The synthesizer also ranks the cluster sizes drawn from the exponential distribution. Then, the synthesizer assigns a cluster center of local density rank r the cluster size of rank r. This way, the linear relationship between cluster size and local density is preserved.

Technique 5: Local distortion - The idea of this technique is to simply pull some nodes within a radius around a local cluster center closer to create the needed in-degree, as illustrated in Figure 7(c). Suppose a local cluster center node i has a cluster size of s_i assigned by Technique 4. We identify the set of its s_i nearest neighbors, X_i, in the synthetic data after global distortion. Then, we compute a radius r_i as $\max_{j \in X_i}(d_{ij})$, and a threshold t_i as $\min_{j,k \in X_i}(d_{jk}) - \epsilon$. Currently, ϵ is set to $0.01 \cdot \min_{j,k \in X_i}(d_{jk})$. Then we associate the values r_i and t_i with node i. r_i is essentially the radius within which distortion may be necessary. t_i is the delay needed to beat the smallest delay among the nodes in X_i.

The delay between node i and j is then computed as follows. Suppose the delay for the edge ij after global distortion is l_{ij}. If neither i nor j is a local cluster center, l_{ij} is returned. Suppose i is a local cluster center and j is not, then if $l_{ij} \leq r_i$, we return $\min(t_i, l_{ij})$; otherwise, we return l_{ij}. The t_i threshold is used to ensure that the nodes in X_i cannot choose one another as their nearest neighbors. After the distortion, they will choose i as their nearest neighbor unless there is a closer node outside of the radius r_i. If both i and j are local cluster centers, we pick the one with the smaller node

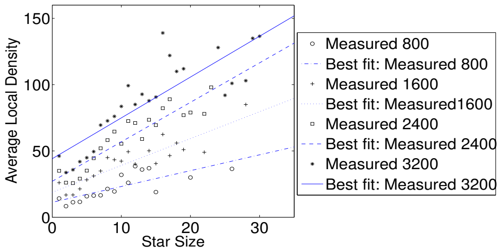


Fig. 8. Average local density vs local cluster size for different sample sizes.

identifier as the center and perform the above steps.

B. Delay Space Synthesizer DS^2

Based on the techniques described above, we have implemented a delay space synthesizer called DS^2. At a high level, DS^2 works as follows: **Step 1.** Perform global clustering on the measured data to assign nodes to major clusters. Perform nearest neighbor directed graph analysis to identify local cluster centers. **Step 2.** Compute a 5D Euclidean embedding of the measured data using a robust method such as the Vivaldi algorithm, which can handle missing data. Then, adjust coordinates to preserve small values. **Step 3.** For each cluster-cluster group g and Euclidean delay l, compute the global distortion statistics $P_{g,l}^{\text{Type-1}}$, $P_{g,l}^{\text{Type-2}}$, $P_{g,l}^{\text{Type-1\&2}}$, $H_{g,l}^{\text{Type-1}}$, $H_{g,l}^{\text{Type-2}}$ using a severe violation threshold R. **Step 4.** At this point, the original measured data is no longer needed. Split the 5D Euclidean map into two, one containing only local cluster centers, and one containing all other nodes. Recall that cluster heads are not simply a random sample of the full data, so they are treated separately. Then each of the two maps is further divided according to which global cluster each node belongs. Assuming there are three major global clusters and the remaining un-clustered nodes form another group, then the splitting procedure produces eight sub-maps. Based on these eight maps, separately synthesize Euclidean maps of each part to the appropriate scale using the Euclidean map synthesis technique. Merge the eight resulting synthesized maps back into one synthesized map. In the final synthesized map, for each node, we now know whether it is a local cluster center and which major cluster it belongs to. **Step 5.** Assign a local cluster size to each synthesized center using the local cluster size assignment technique. For each local cluster center i, compute the local distortion statistics r_i and t_i. **Step 6.** To compute the synthesized delay between node i and j, we first compute the Euclidean delay. In order to reproduce TIVs characteristics, we always apply global distortion according to the global distortion statistics from the measured data, and finally apply local distortion according to the local distortion statistics from the measured data . Return final value.

Note that a lower bound can be enforced on the synthesized delays to mimic some sort of minimum processing delay incurred by network devices. DS^2 provides this as an option.

V. Design and Evaluation of DS^2

In this section, we describe the design and implementation details of the DS^2 software (see [9] for further information).

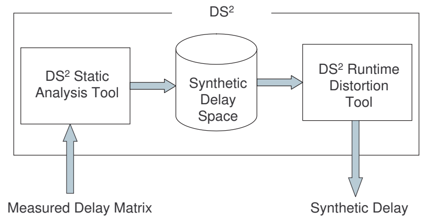


Fig. 9. Architecture of DS^2 software.

A. Design of DS^2

The architecture of DS^2 is shown in Figure 9. The DS^2 software comprises two separate tools: the DS^2 static analysis tool, which analyzes the input delay matrix to generate a synthetic delay space, and the DS^2 runtime distortion tool, which generates synthetic delays upon requests at runtime. By decoupling its functionalities into two separate tools, DS^2 can be used in simulations more efficiently. First of all, the DS^2 static analysis tool allows users to perform time-consuming analysis to generate a synthetic and reusable delay space model offline. Secondly, the resulting synthetic delay space can be recorded in an intermediate file that can be used by different DS^2 runtime distortion implementations. Thirdly, in order to use DS^2 in simulations, users only need to incorporate a runtime distortion implementation into the simulators and then load the generated synthetic delay space into the runtime distortion implementation.

The DS^2 static analysis tool is implemented in C++. Specifically, it implements step 1 to step 5 in Section IV-B. The DS^2 runtime distortion tool is currently implemented as both a C++ class and a Java class so it can be incorporated into C++ based or Java based simulators easily. It implements step 6 in Section IV-B. Using DS^2 in simulations involves the following steps: **Step 1.** Generate a synthetic delay space using the DS^2 static analysis tool offline. **Step 2.** Instantiate a DS^2 runtime distortion object (C++ version or Java version) in the simulator. **Step 3.** Load the generated synthetic delay space into the DS^2 runtime distortion object. **Step 4.** When a delay between two nodes is needed, the simulator just simply calls the DS^2 runtime distortion object to generate the delay.

B. Evaluating the Synthesized Delay Space Model

To evaluate the effectiveness of the synthesized delay model, we first extract a 2,000 node random sub-sample from the measured data. Then, we feed DS^2 with just this 2,000 node sub-sample to synthesize delay spaces with 2x, 4x, and 50x scaling factors. If DS^2 correctly predicts and preserves the scaling trends, then the synthetic 2x delay space should have properties very similar to those found in the measured data from 3,997 nodes. The larger scaling factors (4x and 50x) are presented to illustrate how the synthesizer preserves various properties under scaling. Note that at the scaling factor of 50x, a 100,000 node delay space is synthesized. Unfortunately, at this scale, we do not have efficient ways to compute global clustering (requires $O(N^3)$ space) and triangle inequality violation ratios (requires $O(N^3)$ time) and thus results for these two metrics are calculated based on a 16,000 node

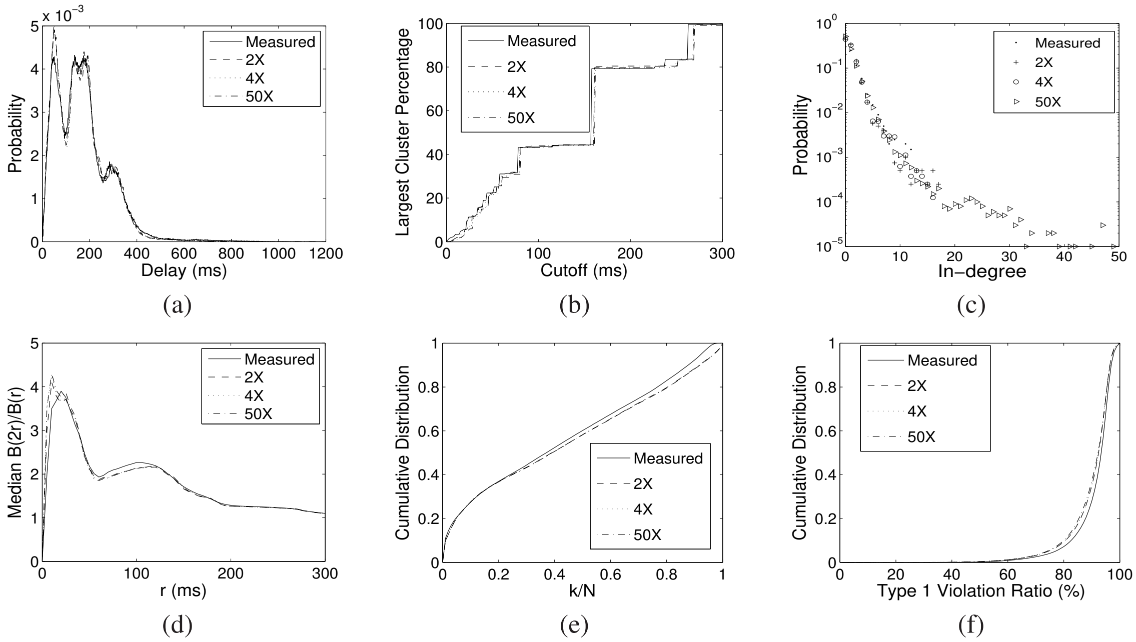


Fig. 10. DS^2 vs measured data. (a) Delay distribution. (b) Clustering cutoff. (c) In-degree distribution. (d) Median B(2r)/B(r). (e) D(k)/D(N). (f) Triangle inequality violation ratio distribution.

random sample out of the 50x synthetic delay space.

The results in Figure 10 show that, even though the synthesis is based on a 2,000 node subset of data, the 2x synthesized data is able to match the characteristics of the 3,997 node measured data very well. As expected, there are a few differences. However, these differences are small and we will show in Section VI that they do not negatively affect the application of the synthesized delay model in distributed system simulations. It is also worth noting that the scaling invariants observed in the measured data are maintained by the synthesizer. In summary, the synthesis framework implemented by DS^2 is highly effective in creating realistic delay spaces with compact $O(N)$ storage requirement.

C. Performance of DS^2

In this section, we present the computational performance of DS^2. All experiments are done on a Dell Dimension 9100 desktop with a Pentium 4 3.0 GHz CPU and 2GB of memory.

Performance of the DS^2 static analysis tool - The DS^2 static analysis tool is applied to our 3,997×3,997 measured delay matrix to synthesize a synthetic delay space with a scaling factor of 10. The computation times for all components in the DS^2 static analysis tool are shown in Table III.

Component Name	(min.)	(sec.)
Euclidean embedding	5	36
Adjust coordinates	3	42
Calculate relative errors	0	34
Calculate triangle inequality violations	44	30
KNN analysis	0	12
Global clustering	13	57
Re-organize matrices	2	45
Synthesis	3	43
Extract global distortion statistics	0	58
Extract local distortion statistics	66	24
Delete temporary files	0	2
Total time	142	23

TABLE III
COMPUTATION TIME OF COMPONENTS IN DS^2 STATIC ANALYSIS TOOL.

"Extract local distortion statistics", "Calculate triangle inequality violations" and "Global clustering" are the three most time-consuming components. Let us denote the scaling factor as F and the size of input matrix as N, then the computation complexity of the component "Extract local distortion statistics" is $O((N \times F)^2)$. The computation complexities of "Calculate triangle inequality violations" and "Global clustering" are both $O(N^3)$. Fortunately, these expensive components can be done offline and only need to be done once.

The resulting synthetic delay space is a 40MB ASCII text file, which can be easily loaded into any modern computer's memory. In contrast, if we use a full matrix to represent the delay space of a 40k-node network and assume we use 4 bytes to hold each value, we need 40k×40k×4 =6.4 GB of memory.

Performance of DS^2 runtime distortion tool - A delay space model with 4k nodes is generated using the DS^2 static analysis tool and loaded into the DS^2 runtime distortion tool. Then a C program, which simply keeps querying the DS^2 runtime distortion tool, is used to test the standalone performance of DS^2. 10^8 random delay requests are issued to the DS^2 runtime distortion tool and it takes DS^2 5.6 microsecond on average to handle each delay request. For comparison, another C program simply loads a 4k×4k delay matrix into memory and then issues 10^8 random memory access on the delay matrix. It takes 0.24 microsecond to perform one memory access on average. The comparison results are summarized in Table IV. Although DS^2 is slower than direct memory access, such delay calculations are only a small part of a simulation. In addition, DS^2 features constant overhead, i.e., the computation time does not increase with the size of the simulated network. In contrast, the approach of using a full delay matrix does not scale. To further improve the efficiency of DS^2, a caching system can be implemented in DS^2 to cache the recently calculated delays. This can benefit those simulations that have good delay request locality.

Models	Average Time (ms)	Memory (MB)
DS^2: 4k nodes	5.6	30
Delay Matrix: 4k nodes	0.24	64
DS^2: 40k nodes	5.6	40
Delay Matrix: 40k nodes	105	6,400

TABLE IV
PERFORMANCE OF STANDALONE DS^2 RUNTIME DISTORTION TOOL.

Table IV compares the performance of using DS^2 and using a delay matrix to simulate a 40k-node network. DS^2 only requires 40 MB of memory to simulate a 40k-node network and the runtime simulation overhead remains the same as simulating a 4k-node network. While if a 40k×40k delay matrix is used, the delay matrix of 6.4 GB cannot be completely loaded into memory so pages have to be swapped in and out, which is very inefficient. This is why the average access time of the direct matrix approach increases 438 times to 105 microseconds per request.

D. Limitations of DS^2

The strength of DS^2 is that it synthesizes delay space models based on the empirical characteristics of measured Internet delays. This approach produces realistic delay spaces at scales where direct measurement and storage are impractical.

One down side of this approach is that DS^2 is designed based on a set of assumptions that are empirically derived from delays among edge networks in the Internet. That is, it is not designed to synthesize delays within a local area network, although such a capability can be incorporated into DS^2 as future work. Another limitation is that it does not model delay dynamics (e.g., dynamic convergence of routing protocols and congestion events). In fact, modeling delay dynamics is orthogonal to our work on modeling static all-pairs delays. We believe the two models can be integrated when the delay dynamics model becomes available. On the other hand, the static all-pairs delays generated by DS^2 are sufficient for simulations of many distributed systems. Taking Vivaldi as an example, each Vivaldi node periodically probes its neighbors and keeps the minimum delay to each neighbor, i.e, it is designed to filter out the delay dynamics. Our goal is to provide a way for stressing the *scalability* of distributed systems under realistic delay models. For evaluations that concern adaptability to delay dynamics, our model does not apply and Planet-lab based experiments could be used.

We have experimented with PlanetLab delay data as well as p2psim delay data and found that DS^2 can correctly synthesize the characteristics of these data sets. However, DS^2 may not work correctly on arbitrary delay data inputs that violate the following empirical assumptions:

• A low-dimensional Euclidean embedding can model the input delay data with reasonable accuracy, ignoring triangle inequality violations and local clustering properties. Some recent studies (e.g., [21], [17]) have shown that Euclidean embedding has difficulties in predicting pairwise Internet delays very accurately. Note, however, that we do not aim at predicting pairwise delays, we only use the Euclidean embedding as a compact model of the statistical properties of the input data.

• The in-degree distribution of the nearest neighbor graph computed from the input data is exponential. The current implementation of DS^2 automatically fits the in-degree distribution of the input data to an exponential distribution.

• Due to the limited amount of data available, we cannot yet accurately model the slow decreasing trend for the proportion of local cluster heads. The current implementation of DS^2 assumes the proportion of cluster heads is stable under scaling. However, this can be easily changed once an accurate model becomes available.

• The input data has a coarse-grained clustering structure. And the delay edges across the same coarse-grained cluster pair exhibit similar triangle inequality violation characteristics.

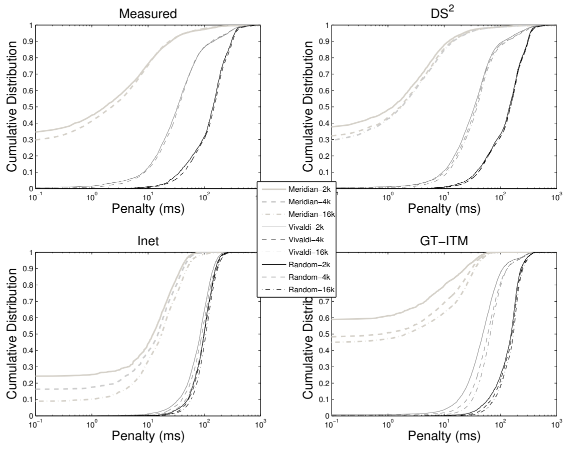


Fig. 11. Performance comparison of three server selection algorithms.

VI. Applications

In this section, we demonstrate the importance of using a realistic delay model for simulation-based evaluation of distributed systems.

A. Server Selection

A number of server selection systems [46], [11], [7] have been proposed recently. In this section, the performance of Meridian [46], Vivaldi [7] and random server selection is evaluated using four different delay spaces: measured data, DS^2, Inet and GT-ITM.

We evaluate the accuracy of the three server selection algorithms using the delay penalty metric, which is defined as the difference between the delay to the chosen server and the delay to the closest server. We run each algorithm on all of the following data sets: for measured data, in addition to the full 3,997-node data, we also use a 2k sample; for DS^2 data, we synthesize 2k data from a 1k sample of the measured data, and synthesize 4k and 16k data from a 2k sample of the measured data; for both Inet and GT-ITM, we generate 2k, 4k and 16k data sets, respectively, using the same methodology described in Section II. In all server selection experiments, we assume that there is only one service available in the network, and all the nodes act as clients and servers simultaneously. Clients are not allowed to select themselves as their servers. For each data set, we run five experiments with different random seeds and the cumulative distributions of server selection penalties are presented in Figure 11.

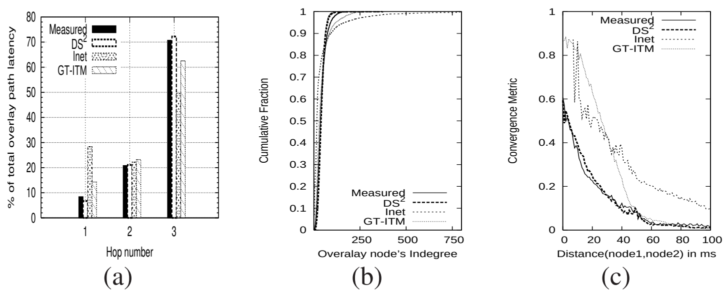
First of all, the synthesized 2k and 4k DS^2 data sets yield virtually identical results as the 2k and 3,997-node measured data, even though they are synthesized from only 1k and 2k measured data samples, respectively. Second, using the Inet model significantly underestimates the performance of Vivaldi. The results suggest that Vivaldi performs no better than random server selection, while Vivaldi performs much better than random selection if it is evaluated using the measured data or DS^2 data. Thus, using Inet model could lead to false conclusions about the performance of Vivaldi. Third, although the relative performance rank of the three algorithms is the same across all four delay models, the absolute performance estimated with Inet and GT-ITM differs dramatically from that achieved with the measured data or DS^2 data.

Finally, the experiment based on the 16k DS^2 synthetic data indicates that the performance of Vivaldi should almost remain constant under scaling, but this is not the case with Inet and GT-ITM delay models. Similarly, Meridian's performance degrades more rapidly on Inet and GT-ITM than on DS^2 data with increasing network size. This illustrates that it is important to have good delay space models that are beyond our ability to measure since important performance trends sometimes only show at scale.

B. Structured Overlay Networks

Here, we show the importance of using a realistic delay space in simulation of structured overlay networks. Unless otherwise stated, the results in this section have been evaluated on a 4,000-node overlay network using FreePastry [12], where the delay space used was either based on measured data, DS^2, Inet, or GT-ITM.

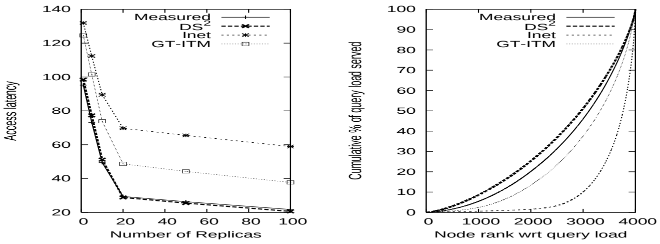
Overlay Metrics - We firstly evaluate three important metrics in overlay network: *Hop Length Distribution* of overlay route, which determines the latency of overlay lookups. *Overlay Indegree* of a node, which is the number of overlay nodes that have the node in their routing tables. *Route Convergence* of overlay routes, which, given two nodes located at distance d from each other, measures what fraction of their overlay paths to a given destination is shared.



(a) (b) (c)

Fig. 12. Overlay properties. (a) Hop Length Distribution. (b) Overlay Indegree. (c) Route Convergence.

Figure 12 shows that the results agree very well for the measured delay data and DS^2 data on all three metrics, while the results with the Inet and GT-ITM models differ significantly. For the hop length distributions, we observe that the first hop of overlay routes with the Inet model is significantly larger than the first hop obtained with measured delay data.



(a) (b)

Fig. 13. Proactive replication. (a) Query latency. (b) Query load.

The Inet model yields different results on indegree because of its power-law connectivity. Finally, the route convergence with Inet/GT-ITM is higher than with the measured data. The deviations of these properties are rooted in the differences of the D(k)/D(N) growth metric and the local clustering in-degree metric among the delay models.

Effectiveness of PNS on Eclipse Attacks - Recently Singh et al. [36] argue that Proximity Neighbor Selection (PNS) alone is a weak defense against Eclipse attacks [3]. While earlier work has shown that PNS is effective against Eclipse attacks based on simulations with a GT-ITM delay model [15], Singh et al. demonstrate that the defense breaks down when using measured delay data in simulations. We have repeated their simulations using DS^2 data and confirmed that DS^2 yields same results as the measured data. This shows again how using inadequate delay models can lead to wrong conclusions.

Performance of Proactive Replication - Proactive replication is very effective in reducing overlay lookup hops and latency in structured overlays [27]. We experimented with a simple proactive replication network that consists of 4,000 nodes (3,997 nodes for measured data) and a total of 10,000 objects.

Figure 13(a) shows that the average query latency for a given number of replicas is significantly lower with measured delay, especially when compared to Inet. This is an artifact of the different distributions of hop length as shown earlier in Figure 12(b). In the absence of realistic models, one would overestimate the number of replicas required to achieve a certain target lookup latency.

We then evaluate the distribution of query traffic to the replicas. The objects were replicated on all nodes that matched at least one digit with the object identifier. Figure 13(b) shows the distribution of query load among the replicas. A point (x, y) in the plot indicates that the x lowest ranked nodes with respect to the amount of query traffic they served, together serve $y\%$ of the overall query traffic. The figure shows a huge imbalance in the load distribution for the Inet topology model, wherein 5% of the nodes serve over 50% of the traffic. This imbalance is caused due to the highly skewed overlay indegree distribution of nodes in the Inet topology.

In conclusion, the DS^2 delay model allows realistic evaluation of the effectiveness and performance of distributed systems at scale. In contrast, simulation results based on GT-ITM and Inet delay models are often not realistic.

VII. Related Work

This paper is based on [48] and contains significant revisions and extensions. In this extended paper, we present the design of the DS^2 delay synthesizer tool. We show that decomposing DS^2 into the analysis and runtime components can lead to an efficient implementation and ease integration into existing network simulators. We have quantified the performance of DS^2 and showed that it has a low runtime overhead and can support large scale simulations highly efficiently. The proposed modeling and synthesis techniques are presented in an integrated fashion. We also explain the importance of synthesizing the local cluster heads separately. The data presented and the DS^2 tools are available at [9].

Our work on modeling the Internet delay space is complementary to existing work on modeling network connectivity topologies. There is an opportunity for future work to incorporate delay space characteristics into topology models.

Early artificial network topologies had a straight-forward connectivity structure such as tree, star, or ring. A more sophisticated topology model that constructs node connectivity based on the random graph model was proposed by Waxman [44]. However, as the hierarchical nature of the Internet connectivity became apparent, solutions that more accurately model this hierarchy, such as Transit-Stub by Calvert *et al* [47] and Tier by Doar [8], emerged. Faloutsos *et al* [10] studied real Internet topology traces and discovered the power-law node degree distribution of the Internet. Li *et al* [18] further showed that router capacity constraints can be integrated with the power-law node degree model to create even more realistic router-level topologies.

There are many on-going projects actively collecting delay measurements of the Internet, including Skitter [37], AMP [2], PingER [26], and Surveyor [41] to name just a few examples. Some of these projects also collect one-way delays and hop-by-hop routing information. These projects typically use a set of monitoring nodes, ranging roughly from 20 to 100, to actively probe a set of destinations. The active monitoring method can probe any destination in the network, but the resulting measurements cover only a small subset of the delay space as observed by the monitors. Many of these measurements are also continuously collected, allowing the study of changes in delay over time. Our work uses the King tool to collect delay measurements, which restricts the probed nodes to be DNS servers, but produces a symmetric delay space matrix, which lends itself to a study of the stationary delay space characteristics.

Some of the delay space properties reported in this paper have been observed in previous work. For example, triangle inequality violations and routing inefficiencies have been observed in [33] and [23]. [50] explored some of the ways in which routing policies can cause TIVs. Some of the characteristics of delay distributions and their implications for global clustering have been observed in Skitter. However, many of the observations made in this paper are new. These include the local clustering properties, and in particular the approximately exponential in-degree distribution, spatial growth properties, detailed properties of triangle inequality violations of different types and across different clusters, and the examination of these properties under scaling. In addition to the "static" properties of delay, previous work have also studied the temporal properties of Internet delay [1]. Incorporating temporal properties into a delay space model is an area for future work.

One key technique used in our work is computing a low dimensional Euclidean embedding of the delay space to enhance the scalability of the delay space representation. Many approaches for computing such an embedding have been studied [23], [7], [34], [6], [19], [42], [35], [25]. We have not considered the impact of using different computation methods or using different embedding objective functions. This represents another area for future work.

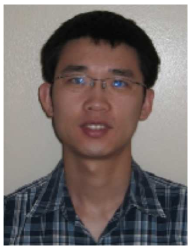
VIII. Conclusions

To the best of our knowledge, this is the first study to systematically analyze, model, and synthesize the Internet delay space. We quantify the properties of the Internet delay space with respect to a set of metrics relevant to distributed systems design. This leads to new understandings of the Internet delay space characteristics which may inform future work. We also develop a set of building block techniques to model and synthesize the Internet delay space compactly while accurately preserving all relevant metrics. The result is an Internet delay space synthesizer called DS^2 that can produce realistic delay spaces at large scale. DS^2 requires only $O(N)$ memory, where N is the number of nodes, and requires only simple run-time calculations to generate the delay between a pair of nodes. This helps to address the memory requirement barrier of conducting large-scale simulations. DS^2 provides an important mechanism for simulating and emulating distributed systems at large-scale, which complements other evaluation methodologies. See [9] for further information on DS^2.

References

[1] A. Acharya and J. Saltz. A Study of Internet Round-Trip Delay. Technical Report CS-TR-3736, University of Maryland, 1996.
[2] Active measurement project, NLANR. http://watt.nlanr.net.
[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D.S. Wallach. Security for Structured Peer-to-Peer Overlay Networks. In *USENIX OSDI*, December 2002.
[4] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting Network Proximity in Peer-to-Peer Overlay Networks. Technical Report MSR-TR-2002-82, Microsoft Research, May 2002.
[5] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Proximity Neighbor Selection in Tree-based Structured Peer-to-peer Overlays. Technical Report MSR-TR-2003-52, Microsoft Research, June 2003.
[6] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. Technical Report MSR-TR-2003-53, Microsoft Research, September 2003.
[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *ACM SIGCOMM*, August 2004.
[8] M. Doar. A Better Model for Generating Test Networks. In *IEEE GLOBECOM*, November 1996.
[9] DS^2. http://www.cs.rice.edu/∼eugeneng/research/ds2/.
[10] C. Faloutsos, M. Faloutsos, and P. Faloutsos. On Power-law Relationships of the Internet Topology. In *ACM SIGCOMM*, August 1999.
[11] M. Freedman, K. Lakshminarayanan, and D. Mazieres. OASIS: Anycast for Any Service. In *USENIX NSDI*, May 2006.
[12] FreePastry. http://freepastry.rice.edu/.
[13] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *ACM SIGCOMM*, August 2003.
[14] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Latency Between Arbitrary Internet End Hosts. In *ACM IMW*, November 2002.
[15] K. Hildrum and J. Kubiatowicz. Asymptotically Efficient Approaches to Fault Tolerance in Peer-to-Peer Networks. In *17th International Symposium on Distributed Computing*, October 2003.
[16] D. R. Karger and M. Ruhl. Finding Nearest Neighbors in Growth Restricted Metrics. In *ACM STOC*, May 2002.

[17] S. Lee, Z. Zhang, S. Sahu, and D. Saha. On Suitability of Euclidean Embedding of Internet Hosts. In *ACM SIGMETRICS*, June 2006.
[18] L. Li, D. Alderson, W. Willinger, and J. Doyle. A First-Principles Approach to Understanding the Internet's Router-level Topology. In *ACM SIGCOMM*, August 2004.
[19] H. Lim, J. Hou, and C.-H. Choi. Constructing Internet Coordinate System Based on Delay Measurement. In *ACM IMC*, October 2003.
[20] J. Møller and R. Waagepetersen. Statistical Inference and Simulation for Spatial Point Processes. Chapman and Hall/CRC, 2004.
[21] E. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the Accuracy of Embeddings for Internet Coordinate Systems. In *ACM IMC*, October 2005.
[22] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *USENIX OSDI*, November 2006.
[23] T. S. E. Ng and H. Zhang. Predicting Internet Networking Distance with Coordinates-Based Approaches. In *IEEE INFOCOM*, June 2002.
[24] p2psim. http://www.pdos.lcs.mit.edu/p2psim/.
[25] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. In *IPTPS*, February 2003.
[26] PingER. http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html.
[27] V. Ramasubramanian and E.G Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *USENIX NSDI*, March 2004.
[28] S. Ranjan, R. Karrer, and E. Knightly. Wide Area Redirection of Dynamic Content by Internet Data Centers. In *IEEE INFOCOM*, 2004.
[29] S. Ratnasamy, S. Shenker, and I. Stoica. Routing Algorithms for DHTs: Some Open Questions. In *IPTPS*, March 2002.
[30] R. Reiss. A Course on Point Processes. Springer Series in Statistics. Springer, 1993.
[31] Route views. http://www.routeviews.org/.
[32] B. Bhattacharjee S. Banerjee and C. Kommareddy. Scalable Application Layer Multicast. In *ACM SIGCOMM*, August 2002.
[33] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-end Effects of Internet Path Selection. In *ACM SIGCOMM*, August 1999.
[34] Y. Shavitt and T. Tankel. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In *IEEE INFOCOM*, March 2003.
[35] Y. Shavitt and T. Tankel. On the Curvature of the Internet and Its Usage for Overlay Construction and Distance Estimation. In *IEEE INFOCOM*, March 2004.
[36] A. Singh, T. W. Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *IEEE INFOCOM*, 2006.
[37] Skitter. http://www.caida.org/tools/measurement/skitter/.
[38] A. Slivkins. Distributed Approaches to Triangulation and Embedding. In *16th ACM-SIAM SODA*, January 2004.
[39] A. Slivkins, J. Kleinberg, and T. Wexler. Triangulation and Embedding Using Small Sets of Beacons. In *IEEE FOCS*, October 2004.
[40] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, August 2001.
[41] Surveyor. http://www.advanced.org/csg-ippm/.
[42] L. Tang and M. Crovella. Virtual Landmarks for the Internet. In *ACM IMC*, October 2003.
[43] M. Waldvogel and R. Rinaldi. Efficient Topology-Aware Overlay Network. In *ACM HotNets-I*, October 2002.
[44] B. Waxman. Routing of Multipoint Connections. *IEEE J. Select. Areas Commun.*, December 1988.
[45] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report UM-CSE-TR-456-02, University of Michigan, 2002.
[46] B. Wong, A. Slivkins, and E. Sirer. Meridian: A Lightweight Network Location Service Without Virtual Coordinates. In *ACM SIGCOMM*, August 2005.
[47] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *IEEE INFOCOM*, March 1996.
[48] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang. Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space. In *ACM IMC*, October 2006.
[49] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Wide-area Fault-tolerant Location and Routing. *U.C. Berkeley Technical Report UCB//CSD-01-1141*, 2001.
[50] H. Zheng, E. Luo, M. Pias, and T. Griffin. Internet Routing Policies and Round-Trip Time. In *PAM*, March 2005.



Bo Zhang is a Ph.D. student in Computer Science at Rice University. He received a B.S. in Computer Science in 2004 from the University of Science and Technology of China and a M.S. in Computer Science in 2007 from Rice University. His research interest lies in network management and network measurement.

T. S. Eugene Ng is an Assistant Professor of Computer Science at Rice University. He is a recipient of a NSF CAREER Award (2005) and an Alfred P. Sloan Fellowship (2009). He received a Ph.D. in Computer Science from Carnegie Mellon University in 2003. His research interest lies in developing new network models, network architectures, and holistic networked systems that enable a robust and manageable network infrastructure.

Animesh Nandi got his Ph.D. from Rice University in 2009. He previously got his M.S. from Rice University in 2004, and his B.Tech. from Indian Institute of Technology, Kharagpur, India in 2001. All the above degrees were in Computer Science. He is currently visiting the Max Planck Institute for Software Systems, Germany, working in the networked and distributed systems group. His research interests lie in the area of Internet-scale networked and distributed systems, content distribution networks and middleware for communication systems.

Rudolf Riedi is a Professor of Mathematics at EIF Fribourg, Switzerland. Prior to joining the EIF in Sept 2007, he was an Associate Professor of Statistics at Rice University. He received his Ph.D. in Mathematics in 1993 from ETH Zurich, Switzerland. His research focuses on the development of multiscale methodologies for modeling, estimation, inference and simulation, emphasizing on applications to complex systems with evolutionary components.

Peter Druschel is on the faculty of the Max Planck Institute for Software Systems (MPI-SWS). Prior to joining MPI-SWS in 2005, he was a Professor of Computer Science at Rice University. He received a Ph.D. from the University of Arizona (1994), an NSF CAREER Award (1995), a Alfred P. Sloan Fellowship (2000) and the Mark Weiser Award (2008). His research interests are in understanding, designing and building distributed systems.

Guohui Wang is a Ph.D. student in Computer Science at Rice University. He received a B.S. in Electrical Engineering from University of Science and Technology of China in 2002, a M.S. in Computer Science from Chinese Academy of Science in 2005 and a M.S. in Computer Science from Rice University in 2008. His research interests are in networking and distributed systems.