RICE UNIVERSITY

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

by

Bo Zhang

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE
Master of Science

APPROVED, THESIS COMMITTEE:

Dr. T. S. Eugene Ng (Chair),
Assistant Professor,
Computer Science

Dr. Edward W. Knightly,
Professor,
Electrical and Computer Engineering

Dr. Rudolf H. Riedi,
Associate Professor,
Statistics

Dr. Peter Druschel,
Scientific Director,
Max Planck Institute for Software
Systems

HOUSTON, TEXAS

May, 2007

UMI Number: 1441865

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 1441865

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

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

by

Bo Zhang

Understanding the characteristics of the Internet delay space 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 to study design alternatives, they need a realistic model of the Internet delay space. We analyze measured delay spaces and identify key properties that are important for distributed system design. Our analysis shows that existing models do not adequately capture important properties of the Internet delay space. Furthermore, we derive a simple model of the Internet delay space, which 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.

Acknowledgment

First and foremost, I would like to thank my advisor, Dr. T. S. Eugene Ng, for his insight in helping guide the direction of this thesis. He has been supporting me and providing me the freedom and opportunity to perform the research from the beginning. I would also like to thank Dr. Edward Knightly, Dr. Rudolf Riedi and Dr. Peter Druschel for being on my committee and providing me with valuable feedbacks that made the thesis a lot better. I am indebted greatly to both Animesh Nandi and Guohui Wang for their efforts and insightful comments regarding my research. Additionally, I would like to thank Atul Singh for providing the results for the Eclipse attack experiments, and Evan Stade for providing a simulator of Meridian. Finally, I would like to thank Angela Yun Zhu and Lily Lam for helping and encouraging me during the preparation of my thesis defense.

Contents

List of Illustrations	vi
List of Tables	viii
1 Introduction	1
2 Methodology and Models	4
2.1 Measured Internet Delay Space	4
2.2 Topology Model Delay Spaces	6
2.3 Limitations of Measured Delay Data	7
3 Internet Delay Space Analysis	8
3.1 Systems-motivated Metrics	8
3.2 Analysis Results	10
3.3 Summary	17
4 Internet Delay Space Modeling	19
4.1 Building Block Techniques	19
4.2 Modeling Framework	24
4.3 Evaluating the Model	25
5 Internet Delay Space Synthesis	26
5.1 Building Block Techniques	26
5.2 Delay Space Synthesizer DS^2	30
5.3 Evaluating the Synthesized Delay Model	31

5.4	Assumptions	32
6	Design and Implementation of DS^2	34
6.1	Design of DS^2	34
6.2	Implementation of DS^2	35
6.2.1	The DS^2 Static Analysis Tool	35
6.2.2	The DS^2 Runtime Distortion Tool	37
6.3	Performance of DS^2	38
6.3.1	Performance of the DS^2 Static Analysis Tool	38
6.3.2	Performance of the DS^2 Runtime Distortion Tool	40
7	Applications	43
7.1	Server Selection	43
7.2	Structured Overlay Networks	45
8	Related Work	50
9	Conclusions	52
	Appendices	52
A	Manual of DS^2	53
A.1	Code Structure	53
A.2	Usage of DS^2 Static Analysis Tool	53
A.3	Usage of DS^2 Runtime Distortion Tool	54
A.4	Integration DS^2 with P2psim	55
	Bibliography	56

Illustrations

3.1	Nearest neighbor directed graph analysis technique.	9
3.2	Global clustering properties. (a) Delay distribution. (b) 2D coordinates scatter plot. (c) Clustering results.	11
3.3	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).	12
3.4	Growth metrics. (a) $B(2r)/B(r)$ metric (log-linear scale). (b) $D(k)/D(N)$ metric.	14
3.5	Type 1 triangle inequality violations for Measured Data (white color is most severe).	15
3.6	Triangle inequality violation ratio distributions. (a) Type 1 violations. (b) Type 2 violations.	17
4.1	Delay distribution of the 5D Euclidean map.	20
4.2	Global distortion technique.	21
4.3	Local distortion technique.	23
4.4	Model 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.	25
5.1	Computing intensities in Euclidean map synthesis technique.	27

5.2	Average local density vs local cluster (star) size for different data sample sizes.	29
5.3	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.	31
6.1	Architecture of DS^2 software.	35
7.1	Performance comparison of three server selection algorithms.	44
7.2	Overlay properties comparison: In-degree, hop length and route convergence.	46
7.3	Performance comparison of proactive replication.	47

Tables

3.1	Average fraction of nodes classified as cluster heads in measured data.	13
6.1	Execution time for major components in DS^2 static analysis tool. . .	39
6.2	Performance of standalone DS^2 runtime distortion tool.	40
6.3	Performance of DS^2 runtime distortion tool.	41

Chapter 1

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 [KR02] in the delay space 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.

Currently, two approaches are used to obtain a delay model. The first approach, adopted for instance by the P2PSim simulator [p2p], is to collect actual delay measurements using a tool such as King [GSG02]. 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. [Wax88, ZCB96, Doa96, FFF99, LAWD04]) 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. At the same time, 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 almost 20GB of main memory! Finally, to enable efficient simulation, generating a delay value 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 coarse-grained (e.g., AS-level) topology of the Internet and the associated static link delays [MIP⁺06]. 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. If the topology model captures the coarse-grained structure of the Internet well enough, the resulting delays should preserve the characteristics of the Internet delay space. However, it remains unclear how detailed such a model has to be to preserve the relevant characteristics.

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 inter-

ested in enabling accurate, efficient, large-scale simulations, we decided to pursue this approach in this thesis.

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 metrics that are relevant to distributed system design. Based on these analytical findings, we develop a method to model measured Internet delay spaces. The resulting model has a compact $O(N)$ representation (as opposed to the $O(N^2)$ matrix representation) that adequately preserves the relevant delay space characteristics, and can model missing measurements. We then extend our model and develop a method to synthesize an artificial delay space. The method exploits the scaling characteristics found in the measurements and makes it possible to synthesize a delay space much larger than the measured delay space, while preserving the characteristics of the measured data. We make two primary contributions in this work:

- We systematically 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 fundamental insights into Internet delay space characteristics that may inform future work.
- We develop a set of building block techniques to model and synthesize the Internet delay space compactly, while accurately preserving the relevant metrics. The compact representation enables accurate and memory efficient network simulations at large scale.

We emphasize that our goal is to provide a model of the Internet delay space that enables accurate large-scale simulations. We do not attempt to provide either an *explanatory* model of Internet delay, which explains the underlying technical, economic and social forces that shape Internet delays, nor do we attempt to provide a *predictive* model that can estimate the delay between a given pair of real IP hosts. Building such models is also an interesting research direction, but is beyond the scope of this thesis.

Chapter 2

Methodology and Models

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

2.1 Measured Internet Delay Space

We use the King tool [GSG02] to measure the all-pair 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 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 [Rou]. For each IP address, we perform a reverse DNS lookup to determine the associated DNS servers. Each reverse lookup returns a set of DNS servers D_{IP_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 D_{IP_i} and D_{IP_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 to all the servers in the set. 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.

Care must be taken during the measurement process. The measured delays include the DNS server processing delays as well as network queuing delays. We collect

multiple measurement samples and keep only the minimum value to approximate the static delay. However, high packet loss rates can cause insufficient measurement samples. Also, because King measures the delay between two servers, say D_1 and D_2 , by subtracting the delay to D_1 from the delay to D_2 *via* D_1 , it is possible to end up with a very small or even negative delay value if the measurements were tainted by processing or queuing delays.

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 from both directions by using either server as the recursive server. For each direction, we make up to 50 attempts to measure the recursive delay to D_2 *via* D_1 , and up to 50 attempts to measure the 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 negative, or if the delay is greater than 2 seconds or smaller than 100 microseconds, it is discarded. These unrealistic delay values are likely caused by processing and queuing delays that affected the measurements. Also, if the obtained delay between D_1 and D_2 measured in each direction disagrees by more than 10%, we discard the measurement. 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 5000×5000 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 3997 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, 1600, 2400, and 3200. To reduce the sensitivity to a particular random sample, for each sub-sample size, we consider five random sample. Results presented in this thesis are averages over the five samples.

2.2 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 [WJ02] and GT-ITM [ZCB96]. The Inet generator creates a topology that has power-law node degree distribution properties. 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 16000-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 5081 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 4160-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 4096 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.

2.3 Limitations of Measured Delay Data

Our analysis is based on a carefully collected set of measured Internet delay data. The data set, however, does have 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. For example, even though a university campus may have thousands of hosts, we most likely pick only one of its DNS servers to include in the measurement. 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.

Chapter 3

Internet Delay Space Analysis

In this chapter, we first identify a set of metrics that are known to significantly influence the performance of distributed systems. Then, we analyze measured Internet 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.

3.1 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. [RKK04]).

Our algorithm to determine the global clustering works as follows. Given N nodes in the measured input data, it first treats each node as a singleton cluster. The algorithm then 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 delay 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

Figure 3.1 : Nearest neighbor directed graph analysis technique.

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 3.1. 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. [CDHR03]). 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 [KR02]. These algorithms are used, for instance, in Tapestry [ZKJ01]

and Chord [SMK⁺01] to select overlay neighbors. In a growth-restricted metric space, if the number of nodes with a delay of at most r from some node i is $B_i(r)$, then $B_i(2r) \leq c \cdot B_i(r)$, where c is a constant. 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 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 [GGG⁺03, CDHR02, RSS02, CDHR03]. 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 $D(k)$ [CDHR03].

Triangle inequality violations - The triangle inequality states that given points x , y and z in a Euclidean space, 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 [SBK02, KR02, WR02]. Thus, it is important to characterize the frequency and severity of the violations in the Internet delay space.

3.2 Analysis Results

We now present an analysis of the measured delay data with respect to the metrics described above, and some basic properties like the delay distribution. For comparison, we also show the relevant properties of the delays produced by the Inet and GT-ITM models.

We begin with a comparison of the delay distribution. In Figure 3.2(a), we can observe that the delay distributions of the measured data set have characteristic peaks at roughly 45ms, 135ms, and 295ms. This suggests that the nodes form clusters in the

Figure 3.2 : Global clustering properties. (a) Delay distribution. (b) 2D coordinates scatter plot. (c) Clustering results.

data. Analysis of random data sub-samples indicates that the delay distribution is also independent of sample size. 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 visualize the locations of nodes, we first embed the data sets into a 5D Euclidean space using a dimensionality reduction procedure that is robust to missing data. Then, we do a principal component analysis on the 5D Euclidean coordinates to get the first 2 principal components. Several techniques exist to compute the 5D embedding [NZ02, DCKM04, ST03, CCRK03, LHC03, TC03]. Here, we use a slightly modified version of the Vivaldi [DCKM04] method that avoids the missing measurements. We use 32 neighbors per node in Vivaldi.

Figure 3.2(b) displays the scatter plots of the first two principal components of the 5D embedding for different data sets. The visual differences between the measured data and the topology models are striking. It is easy to see that there exists clustering structure in the measured data. In contrast, the nodes in the topology models are distributed more uniformly in the space, and their resulting delay distributions are approximately normal.

Figure 3.3 : 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).

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 3.2(c). Regardless of the sample size, the largest cluster's size increases sharply at cutoff values 155ms and 250ms. 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 120ms, nodes can be effectively classified into the three major clusters. They account for 45% (the North America cluster), 35% (the Europe cluster), and 9% (the Asia cluster) of the nodes, respectively. The remaining 11% are nodes that are scattered outside of the major clusters. 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 also does not resemble that of the measured data.

The global clustering analysis 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 data sets. We emphasize that these results charac-

Sample size	# Cluster heads	Percentage
800	185	23.1%
1600	363	22.7%
2400	547	22.8%
3200	712	22.3%
3997 (all data)	884	22.1%

Table 3.1 : Average fraction of nodes classified as cluster heads in measured data.

terize the properties among edge networks in the Internet; they *do not* characterize the properties among end hosts within local area networks. Figure 3.3(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 3.1. Table 3.1 shows that the proportion of nodes in the data that are classified as local cluster heads is quite stable across different sample sizes. This observation is also true when each major global cluster is considered separately. This property will be useful when we turn to the synthesis of delay spaces later in this thesis.

In contrast, as shown in Figure 3.3(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

Figure 3.4 : Growth metrics. (a) $B(2r)/B(r)$ metric (log-linear scale). (b) $D(k)/D(N)$ metric.

of -1.3970. Thus, the Inet model does not reflect the local clustering properties among edge networks in the measured delay data. For the GT-ITM topology, as shown in Figure 3.3(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.

Next we analyze spatial growth. Figure 3.4(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-linear scale) and have very 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 new findings can help fine tune distributed system algorithms that are sensitive to the ball growth rate. On

Figure 3.5 : Type 1 triangle inequality violations for Measured Data (white color is most severe).

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 3.4(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. 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 [CDHR03].

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 ij , we count the number of Type 1 violations it causes. To illustrate how the number of triangle inequality violations are distributed over the major clusters, we present a matrix in Figure 3.5 for the measured data. To produce this figure, we first reorganize the original data matrix by grouping nodes in the same clusters together. The top left corner has indices (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 for any edge 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. 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 3.6 : Triangle inequality violation ratio distributions. (a) Type 1 violations. (b) Type 2 violations.

Figure 3.5. These results imply that, if two nodes are within the same major cluster, then the chance of finding a shorter overlay path is far lower than when the nodes are in different clusters. Moreover, edges that are used to form better overlay paths are most likely found inside a cluster. Interestingly, observe that it is least likely 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 3.6(a) and Figure 3.6(b) the cumulative distributions of Type 1 and Type 2 violation ratios for different sample sizes. Observe that the distribution is very 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.

3.3 Summary

Our analysis confirms some existing knowledge about the Internet delay space, such as the characteristic delay distribution and continental clustering. In addition, the analysis provides a number of new insights:

- The in-degree distribution of the directed nearest neighbor graph of the measured data resembles an exponential distribution and is stable across sample sizes. The relative number of local clusters also appears stable across sample sizes. These findings can be used to model local clustering properties in the Internet delay space.
- The ball growth metrics reflect the continental clustering structure of the delay space and the observed growth rate is low. These properties can inform the design of distributed algorithms, for instance to find the nearest neighbor. The $D(k)$ function empirically derived from the data shows that it is not difficult to encounter a close-by neighbor by random probing. The function can also be used to compute the expected delay stretch in structured overlay networks.
- The potential benefit of overlay routing for a pair of nodes ij and the utility of the pair for overlay routing can be predicted by the clusters that i and j belong to. In particular, 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.
- Delay spaces derived from existing Internet topology models are dramatically different from the Internet delay space. Understanding these differences can help practitioners to design better evaluation methodologies, more correctly interpret their results, and avoid drawing incorrect conclusions.

Chapter 4

Internet Delay Space Modeling

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 a measured Internet delay space. This model adequately preserves the relevant properties of the measured data, and it has only $O(N)$ storage overhead. Later in this thesis, we will also present techniques to synthesize realistic delay data for a much larger delay space than can be measured in practice so as to enable realistic large-scale simulations.

4.1 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 has a scalable $O(N)$ representation.

Although several techniques exist to compute a Euclidean embedding robustly [NZ02, DCKM04, ST03, CCRK03, LHC03, TC03, SKW04, Sli04], and previous studies have shown that an Internet delay space can be overall well approximated by

Figure 4.1 : Delay distribution of the 5D Euclidean map.

a Euclidean embedding with as little as 5 dimensions, such an embedding tends to inflate the small values ($< 10\text{ms}$) in the delay space too much [LZSS06].

In order to create a model that also preserves small values, we first use the Vivaldi algorithm to create a 5D 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 10ms . Next, among these node pairs, select the node pair (m, n) whose Euclidean distance $\hat{d}_{mn}$ in the embedding is smallest. If $\hat{d}_{mn} > 30\text{ms}$, the procedure terminates. Otherwise, the coordinates of node 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 effect of this procedure is that small values in the measured delay space that are mildly distorted in the initial Vivaldi 5D 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 4.1 shows the delay distributions for (1) the measured data, (2) all the

Figure 3.5), identify the edges in each pair of clusters that cause violations above a certain severity threshold, characterize the distortion distribution for these edges when they are mapped into the 5D Euclidean model, then use this same distortion distribution to introduce distortions when delays are generated from the 5D 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 4.2.

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 5D 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, $P_{g,l}^{Type-1}$, and a histogram $H_{g,l}^{Type-1}$ to characterize the real delay distribution of those severe violation edges. Similarly, for Type 2 violations, we compute the fraction $P_{g,l}^{Type-2}$ and the histogram $H_{g,l}^{Type-2}$. We also compute the fraction of edges that incur severe Type 1 and Type 2 violations simultaneously, $P_{g,l}^{Type-1\&2}$. 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 $P_{g,l_{ij}}^{Type-1}$, $P_{g,l_{ij}}^{Type-2}$, $P_{g,l_{ij}}^{Type-1\&2}$, and using ρ as a random variable, decide whether the edge ij should be treated as a severe Type 1 violation (with probability $P_{g,l_{ij}}^{Type-1} + P_{g,l_{ij}}^{Type-1\&2} \cdot (\frac{P_{g,l_{ij}}^{Type-1}}{P_{g,l_{ij}}^{Type-1} + P_{g,l_{ij}}^{Type-2}} - 1)$), or a severe Type 2 violation (with probability $P_{g,l_{ij}}^{Type-2} + P_{g,l_{ij}}^{Type-1\&2} \cdot (\frac{P_{g,l_{ij}}^{Type-2}}{P_{g,l_{ij}}^{Type-1} + P_{g,l_{ij}}^{Type-2}} - 1)$), or to return the value l_{ij} without distortion. If the edge ij is treated as a severe Type 1 violation, then we

Figure 4.3 : Local distortion technique.

use the histogram $H_{g,l_{ij}}^{Type-1}$ 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 $H_{g,D_{ij}}^{Type-2}$ 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.

Technique 3: Local distortion - To preserve the local clustering properties, we introduce additional local distortion. The idea 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 4.3. From the nearest neighbor directed graph analysis on the measured data, we identify local cluster centers and note their sizes. Suppose a local cluster center node i has a cluster size of s_i in the original data. We identify the set of its s_i nearest neighbors, X_i , in the model 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 in the model. 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 . This additional information adds a constant storage overhead.

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 identifier as the center and perform the above steps.

4.2 Modeling Framework

Based on the basic techniques described above, the overall framework for modeling a measured Internet delay space is 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 and their sizes. **Step 2.** Compute a 5D Euclidean embedding of the measured data using a robust method. Then, adjust coordinates to preserve small values. **Step 3.** For each cluster-cluster group g and Euclidean model delay l , compute the global distortion statistics $P_{g,l}^{Type-1}$, $P_{g,l}^{Type-2}$, $P_{g,l}^{Type-1\&2}$, $H_{g,l}^{Type-1}$, $H_{g,l}^{Type-2}$ using a severe violation threshold R . For each local cluster center i , compute the local distortion statistics r_i and t_i . **Step 4.** At this point, the original measured data is no longer needed. To compute the model delay between node i and j , first compute the Euclidean model delay, then apply global distortion if necessary, and finally apply local distortion if necessary. Return final value. The total storage overhead of the model is $O(N)$ and calculating the delay of an edge at run time is simple and has constant cost.

Figure 4.4 : Model vs measured data. (a) Delay distribution. (b) Clustering cut-off. (c) In-degree distribution. (d) Median $B(2r)/B(r)$. (e) $D(k)/D(N)$. (f) Triangle inequality violation ratio distribution.

4.3 Evaluating the Model

We evaluate the effectiveness of our modeling framework by comparing the properties found in the measured data against properties in the resulting model. Figure 4.4 presents our results. Overall, we can see that the model preserves all the characteristics of the delay space that we had identified. As expected, there are some small discrepancies. As we will show in the rest of this thesis, these small discrepancies do not negatively impact the ability to synthesize realistic artificial delay spaces.

Chapter 5

Internet Delay Space Synthesis

In this chapter, we build upon our empirical understanding of the Internet delay space and our delay space modeling techniques and investigate additional techniques to enable artificial *synthesis* of a realistic delay space. The goal is to allow synthesis of delay spaces at scales that exceed our capability to measure Internet delays. Such a tool is valuable for distributed system design and evaluation.

5.1 Building Block Techniques

The new techniques introduced in this section exploit the scaling properties found in the measured Internet delay space to enable the synthesis of a larger delay space.

Technique 4: Euclidean map synthesis - Given a 5D Euclidean map of an Internet delay space, we seek to capture its locality and growth characteristics so that we can synthesize an artificial map based on these characteristics 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 according to the intensities using an inhomogeneous Poisson point process [IW04, Rei93]¹. 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

¹The 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.

Figure 5.1 : Computing intensities in Euclidean map synthesis technique.

method ignores the growth characteristics in the data. As a result, synthetic points can only appear in hyper-cubes where points were originally found.

To incorporate growth characteristics, the idea is to introduce uncertainties in the locations of each point and compute intensities that predict growth. The idea is best explained with a simple example illustrated in Figure 5.1. 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. We will iteratively compute, using the first half of the points, the i^{th} intensity component C_i , which is a vector of intensities for the bins, to predict the 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. $R_i = S_i \setminus S_{i-1}$. Since this is the first component, R_1 is simply S_1 .

The *intensity* I_1 generated by the Gaussians 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 growth in the second half and we are done. However, in the example, R_1 is only covering 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 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 $S_2 \setminus S_1$, 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 C_i . Finally, we repeat the procedure to use the second half to predict the growth in the first half and use the average intensity of each bin as the final intensity. In practice, we divide the 5D space into 100 bins in each dimension and vary the Gaussian variance or width from one-tenth to ten times the bin width.

Technique 5: Local cluster size assignment - In order to preserve realistic local clustering property, the synthesizer draws the cluster sizes from the exponential distribution (as computed in Section 3.2) that approximates the local cluster size

Figure 5.2 : Average local density vs local cluster (star) size for different data sample sizes.

distribution of the measured data. What remains unclear is how to assign different cluster sizes to the synthesized cluster centers. Should the cluster sizes 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 5.2 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.

5.2 Delay Space Synthesizer DS^2

Based on the techniques described above and in Section 4.1, we have implemented a delay space synthesizer called DS^2 (see [DS2] for further information). At a high level, DS^2 works as follows (Steps 1-3 are identical to those in Section 4.2): **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. Then, adjust coordinates to preserve small values. **Step 3.** For each cluster-cluster group g and Euclidean delay l , compute the global distortion statistics $P_{g,l}^{Type-1}$, $P_{g,l}^{Type-2}$, $P_{g,l}^{Type-1\&2}$, $H_{g,l}^{Type-1}$, $H_{g,l}^{Type-2}$ 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. 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 , first compute the Euclidean delay. Apply global distortion, if necessary, according to the statistics from the measured data, and finally apply local distortion if necessary. Return final value.

Note that a lower bound can easily 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.

Figure 5.3 : 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.

5.3 Evaluating the Synthesized Delay 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 3997 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 random sample out of the 50x synthetic delay space.

The results in Figure 5.3 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 3997 node measured data very well. As expected, there are a few differences. However, these differences are small and we will show in Chapter 7 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.

5.4 Assumptions

DS^2 is designed based on a set of assumptions that are empirically derived from delays among edge networks in the Internet. It is not designed to synthesize delays within a local area network. Such a capability can be incorporated into DS^2 as future work.

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 [LGP⁺05, LZSS06] 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.

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

Chapter 6

Design and Implementation of DS^2

In this chapter, we describe the design and implementation details of the DS^2 software.

6.1 Design of DS^2

The architecture of DS^2 is shown in Figure 6.1. 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.

Specifically, the DS^2 static analysis tool implements step 1 to step 5 in Section 5.2. It is used to analyze the measured delay matrix, extract necessary statistics and then record the synthetic delay space in an intermediate file. While the DS^2 runtime distortion tool implements step 6 in Section 5.2, which is used to calculate the synthetic end-to-end delay based on the generated synthetic delay space. Given the user's delay request, it only needs to do simple runtime computation.

Figure 6.1 : Architecture of DS^2 software.

6.2 Implementation of DS^2

In this section, the implementation details of the DS^2 software are described. We also demonstrate how to encapsulate DS^2 into p2psim, a popular peer-to-peer simulator, in Section A.4.

6.2.1 The DS^2 Static Analysis Tool

The DS^2 static analysis tool performs static and one-time analysis on the input measured delay matrix. It produces a synthetic delay space model based on the analysis results. Note that the input matrix must be a $N \times N$ full matrix. The missing measurement in the input matrix must be represented by -1 . This tool is composed of the following major components:

- **Global clustering:** Many existing clustering algorithms are available. In DS^2 , the single linkage algorithm is employed to do clustering. This algorithm works as follows: given N nodes in the measured input data, it first treats each node as a singleton cluster. The algorithm then iteratively finds two closest clusters to merge. The distance between two clusters is defined as the unweighted average distance between

nodes in the two clusters. A cutoff delay value determines when to stop the merging process. By varying the cutoff value, we find that there are three major clusters in our measured delay matrix. So in our implementation, we assume that there are three major clusters in the underlying delay space and the rest of nodes form the fourth cluster. This implementation can be easily modified to support adaptive number of clusters.

- Local clustering: The nearest neighbor analysis technique described in Section 3.2 is used to determine all local cluster centers and their corresponding in-degrees. We assume the in-degree distribution conforms to an exponential distribution, so in our implementation we apply linear regression to fit the in-degrees and the logarithm of the probability of in-degrees to a straight line. Then the resulting best fit line is used to guide the local distortion statistics extraction component.
- Euclidean embedding: This component is used to embed the delay matrix into low dimensional Euclidean space in order to get a compact representation. Two representative embedding algorithms are included in DS^2 : Vivaldi and GNP. In the Vivaldi implementation, each node is assigned a number of random probing neighbors. When choosing random neighbors, we avoid those missing links. Users can replace the default random neighbor selection with their own neighbors assignment algorithm easily. In the GNP implementation, we pick a number of random nodes as landmarks. Similarly users can replace the default random landmarks selection algorithm with their customized landmarks selection algorithm easily.
- Euclidean map synthesis: This component is used to synthesize a larger Euclidean map from the Euclidean embedding of the input measured delay matrix. As described in Section 5.1, we first divide the Euclidean space into a large number of equal size bins. Here we need make a trade-off between preserving overall structure of the Euclidean embedding and adding uncertainty to capture the growth characteristics. We hope to introduce some uncertainty to the synthesis procedure while preserving overall structure. In our implementation, 5-D Euclidean space is used and each di-

mension is divided into 300 equal size bins. Then a series of Gaussian distributions with different variances are used to distribute the intensity across the whole space. Theoretically we need to keep track of the accumulated intensities of all 300^5 bins, which is infeasible. In our implementation, we actually use a sparse matrix to only keep records of the bins that have intensity larger than 0.00001 and ignore other bins in order to make the synthesis procedure feasible and efficient.

- Global distortion statistics extraction: This component is used to extract global distortion statistics from the measured delay matrix. The extracted distortion statistics will be used to guide global distortion in the DS^2 runtime distortion tool in order to preserve the triangle inequality violation property.
- Local distortion statistics extraction: This component is used to extract local distortion statistics, which will then be used to guide local distortion in the DS^2 runtime distortion tool in order to preserve the local clustering property.

In summary, the DS^2 static analysis tool can produce a realistic and compact delay space model based on the analytical findings from the measured delay matrix. Please see Section A.2 for the usage of this tool.

6.2.2 The DS^2 Runtime Distortion Tool

The DS^2 runtime distortion tool is used to generate synthetic end-to-end delay at runtime. This tool first loads the delay space model generated by the DS^2 static analysis tool, then to compute the delay between node i and node j , it will first calculate the Euclidean distance between node i and j and then apply appropriate distortions to the Euclidean distance. Two kinds of distortion can be applied here:

- Global distortion: As described in section 5.1, we rely on random numbers to decide whether and how we should distort each requested delay. In order to guarantee that the same distortion is applied to same delay each time when it is requested, we use the product of two node-IDs as the seed to initialize the random number generator.
- Local distortion: This step is straightforward. Appropriate local distortion is ap-

plied according to the extracted local distortion statistics.

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. The straightforward way of using DS^2 in simulations is as follows: **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. We will show how to use the DS^2 runtime distortion tool in the p2psim [p2p] in Section A.4.

6.3 Performance of DS^2

In this section, the performance of DS^2 is studied. The execution performance of the DS^2 static analysis tool is first presented, followed by the performance of the DS^2 runtime distortion module. Finally, we evaluate the performance of integrating DS^2 with p2psim.

All experiments in this section are done on a Dell Dimension 9100 desktop with a Pentium 4 3.0 GHz CPU and 2GB of memory, which runs Redhat Linux.

6.3.1 Performance of the DS^2 Static Analysis Tool

The DS^2 static analysis tool is applied to our 3997×3997 measured delay matrix to synthesize a synthetic delay space with a scaling factor of 10. Default values are used for all other parameters. The profiling results of all components in the DS^2 static analysis tool are shown in Table 6.1.

As can be seen from Table 6.1, “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 SF and the size of input matrix as N, then the computation complexity of the component “Extract local dis-

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

Table 6.1 : Execution time for major components in DS^2 static analysis tool.

tortion statistics” is $O((N \times SF)^2)$. But fortunately, this analysis component can be done offline and only need to be done once. The computation complexities of “Calculate triangle inequality violations” and “Global clustering” are both $O(N^3)$. In addition, they only depend on the size of the input delay matrix and have nothing to do with the scaling factor. Again these two analysis components can be done offline and only need to be done once.

The resulting synthetic delay space model is a 40MB ASCII text file, which can be easily loaded into any modern computer’s main 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 $40k \times 40k \times 4 = 6.4$ GB of memory space. While with the help of DS^2 , 40 MB of memory is enough to simulate a 40k-node network realistically.

Models	Average Access Time (microsecond)	Memory (MB)
DS^2 : 4k nodes	5.6	30
Delay Matrix: 4k nodes	0.24	64

Table 6.2 : Performance of standalone DS^2 runtime distortion tool.

6.3.2 Performance of the DS^2 Runtime Distortion Tool

In this section, we study the performance of the DS^2 runtime distortion tool. The DS^2 static analysis tool and the C++ version of the DS^2 runtime distortion tool are used to run the experiments.

6.3.2.1 Performance of Standalone 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 9 minutes 20 seconds for it to finish calculation. So DS^2 can handle each delay request in 5.6 microsecond on average. For comparison, another C program simply loads a $4k \times 4k$ delay matrix into the main memory and then issues 10^8 random memory access on the delay matrix. It costs 24 seconds for all the requests to be completed, which means it takes 0.24 microsecond to perform one memory access on average. The comparison results are summarized in Table 6.2. Although DS^2 is 23 times slower than direct memory access, such delay calculations are only a small part of a simulation. In addition, DS^2 features constant simulation overhead, i.e., the runtime simulation overhead does not increase with the size of the simulated network. While the approach of using full delay matrix does not scale.

Table 6.3 compares the performance of using DS^2 and delay matrix to simulate a 40k-node network. As you can see, DS^2 only requires 40 MB of memory to sim-

Models	Average Access Time (microsecond)	Memory (MB)
DS^2 : 40k nodes	5.6	40
Delay Matrix: 40k nodes	105	6400

Table 6.3 : Performance of DS^2 runtime distortion tool.

ulate a 40k-node network and the runtime simulation overhead remains the same as simulating a 4k-node network. While if a $40k \times 40k$ delay matrix is used, it requires 6.4 GB of memory and it cannot be completely loaded into memory so pages have to be swapped in and out, which is very inefficient. This is why the simulation overhead of direct matrix approach increases 438 times to 105 microsecond per request on average.

6.3.2.2 Performance of DS^2 in P2psim

Now we experiment to study the performance of DS^2 when it is incorporated into p2psim. We simulate the Vivaldi protocol in p2psim for a 100-second period. The simulation is first driven using our 3997×3997 measured delay matrix, and it takes 20 minutes 10 seconds. Then we run the simulation again using DS^2 to simulate a 3997-node network and it takes 31 minutes 24 seconds. Therefore p2psim simulation using DS^2 is roughly 50% slower than the one directly using delay matrix that can be fully loaded into memory. The extra simulation time in the latter experiment is introduced by DS^2 runtime distortion overhead.

Because it is not possible to simulate the Vivaldi protocol with 40k nodes in p2psim, we only present the estimated simulation time of 40k-node network. During the 3997-node network simulation, Vivaldi issues around 10^8 delay requests. So if Vivaldi is simulated on a 40k-node network, it will roughly issue $10 \times 10^8 = 10^9$ delay requests. The simulation time of Vivaldi using DS^2 will increase 10 times to 314 minutes, i.e., 5 hours 14 minutes. But if we use a $40k \times 40k$ delay matrix instead to

drive the simulation and we assume that the only difference between the simulation using delay matrix and the simulation using DS^2 model is how they handle delay request, then we can estimate that the extra simulation time of using delay matrix is $(105 - 5.6) \times 10^9$ microseconds, i.e., 27 hours and 36 minutes. Therefore, using delay matrix to simulate a 40k-node network requires much more simulation time.

In order to improve the simulation efficiency, a caching system can be implemented in DS^2 to cache the recently calculated delays. For those protocols that have good locality, the caching mechanism can certainly improve their simulation efficiency. For example, in the Vivaldi protocol, if each node has 32 random neighbors, then each node will request the delays to its fixed 32 neighbors periodically. If a cache bigger than $3997 \times 32 \times 4$ bytes is implemented in DS^2 , then the Vivaldi simulation using DS^2 will have very similar execution time to the one using 3997×3997 delay matrix.

Chapter 7

Applications

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

7.1 Server Selection

Increasingly, Internet services are distributed across multiple servers all over the world. The performance and cost of such Internet services depend on the server selection mechanisms they employ. Server selection redirects clients to an appropriate server, based on factors such as the location of the client, network conditions, and server load. A number of server selection systems [WSS05, FLM06, DCKM04] have been proposed and studied. In this section, the performance of Meridian [WSS05], Vivaldi [DCKM04] 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 3997-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 2. 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

Figure 7.1 : Performance comparison of three server selection algorithms.

servers. For each data set, we run five experiments and the cumulative distributions of server selection penalties are presented in Figure 7.1.

First of all, the synthesized 2k and 4k DS^2 data sets yield virtually identical results as the 2k and 3997-node measured data, even though they are synthesized from only 1k and 2k measured data samples, respectively. Second, using the Inet delay model significantly underestimates the performance of Vivaldi. In fact, 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 as a delay 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 results achieved with the measured data or DS^2 data. For example, on the 3997-node measured data, 40.4% of the servers chosen by Meridian are within 1ms of the closest servers, while this number is 17.2% and 50.4% on 4k Inet and 4k GT-ITM data, respectively. 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.

7.2 Structured Overlay Networks

Structured overlay networks like Chord, Kademlia and Pastry use proximity neighbor selection (PNS) to choose overlay neighbors [SMK⁺01, MM02, CDHR03]. PNS has been shown to effectively reduce routing stretch, query delay and network load, and to increase overlay robustness to failures and even to certain security attacks [SNDW06].

Here, we show the importance of using a good delay space to evaluate the ef-

fectiveness of PNS. To eliminate the influence of a particular PNS implementation, we assume in our simulations that the overlay chooses neighbors using perfect PNS, i.e., the closest node is chosen among all overlay nodes that satisfy the structural constraints imposed by a routing table entry. Unless otherwise stated, the results in this section have been evaluated on a 4000 node overlay network using FreePstry [Fre], where the delay space used was either based on measured data, DS^2 , Inet, or GT-ITM.

We firstly evaluate the following metrics: *Overlay Indegree* of a node, which is the number of overlay nodes that have the node in their routing tables. *Hop Length Distribution* of overlay route, which determines the latency and network load of overlay lookups. *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. This metric is important for dynamic caching and for the efficiency of multicast distribution trees.

Figure 7.2 : Overlay properties comparison: In-degree, hop length and route convergence.

Figure 7.2 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. The Inet model yields different results on indegree, because

Figure 7.3 : Performance comparison of proactive replication.

the power-law connectivity makes the closest leaf node of the high degree hubs in the topology the nearest neighbor for a large number of nodes. 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. 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.

Next, we show how mis-predictions of the somewhat abstract overlay metrics shown above can have an impact on application performance metrics whose relevance is more immediately apparent.

Effectiveness of PNS on Eclipse Attacks - In recent work on defenses against Eclipse attacks [CDG⁺02] on structured overlay networks, Singh et al. [SNDW06] argue that PNS alone is a weak defense. While earlier work has shown that PNS is effective against Eclipse attacks based on simulations with a GT-ITM delay model [HK03], Singh et al. demonstrate that the defense breaks down when using measured delay data as a basis for simulations. Moreover, they show that the effectiveness of the defense diminishes with increasing size of the overlay network. We have repeated their simulations using DS^2 data and confirmed that the results match the simula-

tions with measured data. Moreover, we are able to show that the effectiveness of PNS against Eclipse attacks continues to diminish as we increase the network size beyond the scale of measured delay data. This is another example of how the usage of inadequate delay models can lead to wrong conclusions. It also shows that DS^2 yields correct results, and that it is important to be able to synthesize delay models larger than measured data sets, in order to expose important trends.

Performance of Proactive Replication - The benefits of proactive replication in structured overlays to reduce overlay lookup hops and latency has been explored by Beehive [RS04]. We experimented with a simple prototype that does proactive replication based on the number of desired replicas of an object. The placement of replicas is biased towards nodes that share a longer prefix with the object identifier, in order to be able to intercept a large fraction of the overlay routes towards the home node of the object. We first evaluate the query lookup latency as a function of the total number of replicas for an object. Our system consists of a 4000 node network (3997 nodes for measured data) and a total of 10,000 objects.

Figure 7.3(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 7.2(b). With the last hops in realistic models dominating the total overlay path, employing even a very few replicas results in reducing the average latency significantly. 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. In this scenario, we assume that the objects were replicated on all nodes that matched at least one digit with the object identifier. Given a network of 4000 nodes with Pastry node IDs as digits in base 16, we observe that this corresponds approximately to the expected value of $(1/16) \times 4000 = 250$ replicas per object. Figure 7.3(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.

Again, we see that the delay model used in simulations of distributed systems has a significant impact on the results obtained for important application performance metrics. Inadequate delay models can lead to false conclusions about the effectiveness and performance of systems.

Chapter 8

Related Work

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 [Wax88]. 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* [ZCB96] and Tier by Doar [Doa96], emerged. Faloutsos *et al* [FFF99] studied real Internet topology traces and discovered the power-law node degree distribution of the Internet. Li *et al* [LAWD04] 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 [Ski], AMP [Act], PingER [Pin], and Surveyor [Sur] 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 Skitter work probes on the order of 1 million destinations, which is the largest among these projects. 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 thesis have been observed in previous work. For example, triangle inequality violations and routing inefficiencies have been observed in [SCH⁺99] and [NZ02]. 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 thesis 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 [AS96]. 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 completeness and scalability of the delay space representation. Many approaches for computing such an embedding have been studied [NZ02, DCKM04, ST03, CCRK03, LHC03, TC03, ST04, PCW⁺03]. We have not considered the impact of using different computation methods or using different embedding objective functions. This represents another area for future work.

Chapter 9

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 [DS2] for further information on DS^2 .

Appendix A

Manual of DS^2

We implement the DS^2 static analysis tool in C++ and the DS^2 runtime distortion tool in both C++ and Java. All the code can be downloaded from [DS2].

A.1 Code Structure

After unzipping the downloaded code, you will find all the source code in `ds2/src/` directory, where there are three sub-directories:

`ds2/src/synthesizer/`: The code in this directory is used to produce a synthetic, realistic and compact delay space model based on the input measured delay space. Please read Section A.2 for details.

`ds2/src/ds2_runtime_classes/`: This directory contains facility to generate end-to-end synthetic delay at runtime based on the delay space model generated by the code in `ds2/src/synthesizer/` directory. Both C++ and Java versions of the code are provided. Please read Section A.3 for details.

`ds2/src/p2psim/`: This directory includes example code to demonstrate the integration of DS^2 and p2psim. Please read Section A.4 for details.

A.2 Usage of DS^2 Static Analysis Tool

The code in `ds2/src/synthesizer/` directory is used to produce a synthetic, realistic and compact delay space model based on the input measured delay space. A Makefile is provided to compile the code and it will generate an executable file named “ds2”. The input to “ds2” must be a $N \times N$ measured delay matrix, the unit is in millisecond. The missing data in the delay matrix must be represented using -1. The “ds2” will

then generate a file named “DS2Model”, which is the synthetic delay space model based on the input measured delay matrix. The options that “ds2” supports are as bellow:

- [-a Euclidean embedding algorithm: 0 for GNP (default) and 1 for Vivaldi.]
- [-m Name of the measured delay matrix (mat_real as default).]
- [-n Size of measured delay matrix. You must provide the size of input matrix.]
- [-s Scaling factor for synthesizer, it must be larger than 1 and default value is 10.]
- [-r Type 1 triangle inequality violation ratio threshold, it is in (0, 100) and default value is 85.]
- [-i Type 2 triangle inequality violation ratio threshold, it is in (100, Infinity) and default value is 250.]
- [-t Cutoff value for global clustering algorithm, default value is 120ms.]
- [-l Number of neighbors, default value is 32. In GNP algorithm, it stands for the number of landmarks; while in Vivaldi, it stands for the number of random neighbors.]
- [-h Help information (this message).]

A.3 Usage of DS^2 Runtime Distortion Tool

The “ds2” described in Section A.2 generates a compact delay space model file, DS2Model. In order to use the delay space model in simulations, two runtime distortion classes are provided in the ds2/src/ds2_runtime_classes/ directory. One is written in C++ and the other is written in Java.

Here we take the C++ version runtime distortion class as an example to show how to use it. The easiest way to use DS^2 in your simulations is to include the ds2_runtime.h header and instantiate a ds2_runtime object. Before using the ds2_runtime object to generate end-to-end delays, the simulator must first call the loadModel(char *model_file_name) method, which will load and parse the delay space model file and store all the necessary statistics in the object. Then the simulator can call getSyntheticDistance(int index_i, int index_j) method to calculate the end-to-end

delay between node `index_i` and node `index_j`.

A.4 Integration DS^2 with P2psim

P2psim is a popular peer-to-peer protocol simulator. DS^2 can be easily integrated into p2psim. What we need do is to write another topology class that uses the DS^2 runtime distortion class. In the `ds2/src/p2psim/` directory, a topology class named “`e2eds2graph.C`” is included, which uses the DS^2 runtime distortion class to generate end-to-end delays.

In order to use this new class in simulations, we also need to modify the “`Makefile.in`” and the “`topologyfactory.C`” files in p2psim code package. The modified “`Makefile.in`” and “`topologyfactory.C`” are actually also included in the `ds2/src/p2psim/` directory. In order to use the “`e2eds2graph.C`” class in p2psim, just copy all the files in the `ds2/src/p2psim/` directory to the “`topologies`” directory in the p2psim package, then you will be able to use DS^2 to drive your p2psim simulations.

Bibliography

- [Act] Active measurement project, NLANR. <http://watt.nlanr.net>.
- [AS96] A. Acharya and J. Saltz. A Study of Internet Round-Trip Delay. Technical Report CS-TR-3736, University of Maryland, College Park, 1996.
- [CCRK03] 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.
- [CDG⁺02] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D.S. Wallach. Security for structured peer-to-peer overlay networks. In *Proceedings of OSDI*, Boston, MA, December 2002.
- [CDHR02] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, May 2002.
- [CDHR03] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Research, June 2003.
- [DCKM04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceeding of ACM SIGCOMM*, August 2004.
- [Doa96] M. Doar. A better model for generating test networks. In *Proceeding of IEEE GLOBECOM*, November 1996.

- [DS2] DS^2 . <http://www.cs.rice.edu/~eugeneng/research/ds2/>.
- [FFF99] C. Faloutsos, M. Faloutsos, and P. Faloutsos. On Power-law Relationships of the Internet Topology. In *Proceedings of ACM Sigcomm*, August 1999.
- [FLM06] Michael J. Freedman, Karthik Lakshminarayanan, and David Mazieres. Oasis: Anycast for any service. In *Proceedings of ACM NSDI*, May 2006.
- [Fre] Freepastry. <http://freepastry.rice.edu/>.
- [GGG⁺03] Krishna P. Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, August 2003.
- [GSG02] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the SIGCOMM IMW*, Marseille, France, November 2002.
- [HK03] K. Hildrum and J. Kubiawicz. Asymptotically efficient approaches to fault tolerance in peer-to-peer networks. In *Proc. 17th International Symposium on Distributed Computing*, Sorrento, Italy, Oct 2003.
- [KR02] David R. Karger and Matthias Ruhl. Finding nearest neighbors in growth restricted metrics. In *Proceedings of ACM Symposium on Theory of Computing*, 2002.
- [LAWD04] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the internet's router-level topology. In *Proceeding of ACM SIGCOMM*, August 2004.
- [LGP⁺05] Eng Keong Lua, Timothy Griffin, Marcelo Pias, Han Zheng, and Jon Crowcroft. On the accuracy of embeddings for internet coordinate systems. In *Proceedings of IMC*, Berkeley, CA, October 2005.

- [LHC03] H. Lim, J. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Proceedings of IMC*, Miami, FL, October 2003.
- [IW04] Jesper Møller and Rasmus Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. Chapman and Hall/CRC, 2004.
- [LZSS06] Sanghwan Lee, Zhi-Li Zhang, Sambit Sahu, and Debanjan Saha. On suitability of Euclidean embedding of Internet hosts. In *Proc. SIGMETRICS 2006*, June 2006.
- [MIP⁺06] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: an information plane for distributed services. In *Proc. OSDI 2006*, November 2006.
- [MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. IPTPS 2002*, March 2002.
- [NZ02] T. S. E. Ng and H. Zhang. Predicting Internet networking distance with coordinates-based approaches. In *Proceedings of IEEE INFOCOM*, June 2002.
- [p2p] p2psim. <http://www.pdos.lcs.mit.edu/p2psim/>.
- [PCW⁺03] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Proceedings of IPTPS*, 2003.
- [Pin] PingER. <http://www.slac.stanford.edu/comp/net/wanmon/tutorial.html>.
- [Rei93] Rolf-Dieter Reiss. *A Course on Point Processes*. Springer Series in Statistics. Springer, 1993.

- [RKK04] S. Ranjan, R. Karrer, and E. Knightly. Wide area redirection of dynamic content by internet data centers. In *Proceedings of IEEE INFOCOM*, Hong Kong, China, 2004.
- [Rou] Route views. <http://www.routeviews.org/>.
- [RS04] V. Ramasubramanian and E.G Sirer. Beehive: $O(1)$ lookup performance for power-law query distributions in peer-to-peer overlays. In *Proc. NSDI '04*, San Francisco, California, March 2004.
- [RSS02] S. Ratnasamy, S. Shenker, and I. Stoica. Routing algorithms for DHTs: Some open questions. In *Proceedings of IPTPS*, Cambridge, MA, March 2002.
- [SBK02] B. Bhattacharjee S. Banerjee and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, August 2002.
- [SCH⁺99] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-end Effects of Internet Path Selection. In *Proceedings of ACM Sigcomm*, August 1999.
- [Ski] Skitter. <http://www.caida.org/tools/measurement/skitter/>.
- [SKW04] A. Slivkins, J. Kleinberg, and T. Wexler. Triangulation and Embedding using Small Sets of Beacons. In *Proceedings of FOCS*, 2004.
- [Sli04] A. Slivkins. Distributed Approaches to Triangulation and Embedding. In *Proceedings 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.
- [SMK⁺01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

- [SNDW06] A. Singh, T. W. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *Proc. INFOCOM 2006*, Barcelona, Spain, April 2006.
- [ST03] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in Euclidean space. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, March 2003.
- [ST04] Y. Shavitt and T. Tankel. On the curvature of the Internet and its usage for overlay construction and distance estimation. In *Proceedings of IEEE INFOCOM*, April 2004.
- [Sur] Surveyor. <http://www.advanced.org/csg-ippm/>.
- [TC03] L. Tang and M. Crovella. Virtual landmarks for the internet. In *Proceedings of IMC*, Miami, FL, October 2003.
- [Wax88] B. Waxman. Routing of multipoint connections. *IEEE J. Select. Areas Commun.*, December 1988.
- [WJ02] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical Report UM-CSE-TR-456-02, University of Michigan, 2002.
- [WR02] Marcel Waldvogel and Roberto Rinaldi. Efficient Topology-Aware Overlay Network. In *First Workshop on Hot Topics in networks (HotNets-I)*, October 2002.
- [WSS05] Bernard Wong, Aleksandrs Slivkins, and Emin Gun Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proceedings of ACM SIGCOMM*, August 2005.
- [ZCB96] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE INFOCOM*, March 1996.

- [ZKJ01] B. Zhao, J. Kubiawicz, and A. Joseph. Tapestry: An infrastructure for wide-area fault-tolerant location and routing. *U.C. Berkeley Technical Report UCB//CSD-01-1141*, 2001.