Distributed Querying of Internet Distance Information

Rodrigo Fonseca
University of California, Berkeley
rfonseca@cs.berkeley.edu

Puneet Sharma Sujata Banerjee Sung-Ju Lee Sujoy Basu
HP Labs, Palo Alto, USA
{puneet.sharma,sujata.banerjee,sungju.lee, sujoy.basu}@hp.com

***Abstract*— Estimation of network proximity among nodes is an important building block in several applications like service selection and composition, multicast tree formation, and overlay construction. Recently, scalable techniques have been proposed to estimate inter-node latencies, including network coordinate systems like GNP and Vivaldi. However, existing mechanisms for querying such information do not scale well to a very large number of nodes, when one wants to accurately find a set of nodes globally closest to a given node. In this paper we are concerned with distributing the position data among a set of infrastructure nodes, and propose ways of partitioning and querying this data. The trade-offs between accuracy and overhead in this distributed infrastructure are explored. We evaluate our solution through simulations with real and synthetic network measurement data.**

I. Introduction and Motivation

As Internet services further proliferate, current knowledge of network properties becomes more crucial in maintaining efficient operation of the service infrastructure, and the desired performance of applications. These network path properties include end-to-end latency, bandwidth, hops, and error rates between two end points. 'Network health' can change dynamically as the load patterns change or failures occur. Knowledge of such parameters can be crucial to many uses. A distributed service infrastructure can use such network health information to make resource allocation decisions. Applications use such information to tune performance parameters. End clients use information about current network performance to choose a server that maximizes their user experience, for example in a networked multi-player game scenario.

Scalable measurement and querying of dynamic network properties is a challenging task and centralized solutions will certainly fail beyond a certain network size. Below we discuss the options of centralization, replication and partitioning of a network position information repository. Further, measuring every parameter on every path with the lowest measurement granularity is practically impossible[1], and some estimation/inferencing/prediction techniques will be needed. The goal is to provide the most accurate measurement attainable with the lowest measurement overhead as possible.

In this paper we focus only on latency information: how to build an infrastructure to allow querying of network distance to be performed in a scalable and distributed way. This is an important building block for applications such as server selection, multicast tree formation, and overlay construction. We believe that the architecture we explore here could be extended to other network information as well. The specific scenario we examine consists of a very large set of server and client nodes, and a smaller number of measurement infrastructure nodes. We generically call the client and server nodes 'service nodes' throughout the paper. What the infrastructure provides to the server and client nodes is a way of finding other service nodes that satisfy some distance constraints. Specifically, we look into queries for the closest service node to another given node.

Recently, scalable techniques have been developed that allow the estimation of latency and/or ordering of nodes according to latency, *e.g.* [1]–[4], and we use these as methods to obtain what we call *position information* about network nodes. The idea is that the position information of a node can be obtained with relatively few measurements, and given two node's information it is possible to estimate their distance without directly measuring it.

Given a database with such position information about nodes, one can answer queries like 'what is the closest node to node A', 'what are the k closest nodes to A', or 'give me k nodes closer to A than x'. A centralized database with information about all nodes has the advantage of complete information: answers returned will be globally best with no extra cost. However, there are some disadvantages to a centralized system, specially as the number of service nodes increases. First, the centralized system has a single point of failure, and may hinder the availability of the service. Second, if this database is to receive updated information periodically from all of the service nodes, there is clearly a limit to the traffic that such updates can impose: in practice this limits either the frequency of the individual updates or the number of nodes in the system, given the same amount of bandwidth. Lastly, a centralized database cannot be located at the same distance from all nodes that might issue queries, and some of these nodes may end up paying a higher latency to perform the queries. Replicating the database will increase the availability and potentially decrease the average latency for querying, while still providing the same answers as the centralized system, but it further aggravates the network traffic problem for updates.

We propose partitioning the position information database

[1]In some cases, measurement access is not available due to policy decisions or for technical reasons. In other cases, measurement may impair performance.

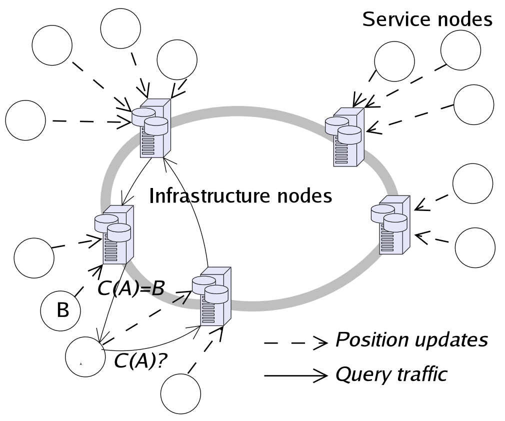

Fig. 1. Architecture for the network position information database

across a set of infrastructure nodes. Our goal is to partition the information such that the quality of the answers is close to that of the centralized system, the update and query traffic is load-balanced among infrastructure nodes, and the query latency is similar to that of the replicated case. Figure 1 shows the distributed architecture for querying the network distance information.

In the next section, we present a brief description of the related work. This is followed by a description of our proposed approach and architecture in Section III. An evaluation of the different schemes is presented in Section IV, followed by concluding remarks.

II. Related Work

There has been quite a bit of work in acquiring and representing network positions recently [1]–[3].

These approaches assign nodes coordinates in a geometric space, such that the distance, in this space, is a good estimate of the true network latency. The advantage in comparison to actually measuring the distance is that only a small fraction of the $O(n^2)$ measurements between the service nodes are needed. GNP embeds the nodes in a cartesian space, and finds that not many dimensions are needed to represent the distances to a good approximation. In [5], the authors use Principal Component Analysis to reduce the dimensionality of the resulting space, without a significant loss in accuracy. Another position represenation, landmark order vector has been presented in [4]. The landmark order vector presents position as a vector of landmark *id*s, ordered by increasing latency.

Our work is orthogonal to these approaches, as we are concerned with partitioning and querying the position information in a scalable way.

In [6], two spatial-database approaches are compared for supporting multi-dimensional range queries in P2P systems. The first approach uses space-filling curves to map multi-dimensional data to a single dimension. The later is then partitioned by ranges among the available nodes. The second approach uses kd-trees to partition the multi-dimensional space into hypercuboids, each of which is assigned to a node. The fundamental difference with our work is our focus on network proximity. Our multi-dimensional data points represent network position, and so the interesting query for us is the distance or relative order of data points. For them, the multi-dimensional data represents metadata of content stored in a P2P network, and so multi-dimensional range queries on the data points is interesting. So even though their first approach of using space-filling curve is similar to one of our approaches, the way in which queries are done is different. They map each multi-dimensional query into one-dimensional queries, which are individually routed. We simply route to the network position of the querying node, after mapping it to the DHT geometry.

PIER [7] is a distributed query engine that sacrifices ACID semantics to perform database queries over a DHT. A study of how proximity queries or range queries will perform on PIER is not available at this point.

In Sword [8], the authors present a framework for storing information about service nodes in a DHT, and to answer multi-attribute queries to locate suitable nodes. When querying for information, a node selects one particular infra-structure node that sends sub-queries to different infrastructure nodes. The replies are aggregated on the original node, and then processed to yield the combined results.

We envision a system in which processing of the query is more distributed, with as much work as possible being done by the infrastructure nodes that have the information.

III. Architecture

The distributed architecture for querying the network distance information, as shown in Figure 1, has three basic components: position measurement and representation, information partitioning and distribution, and information querying. In this paper we focus on information partitioning and querying, and compare schemes based on use three well-known representations of node network position: *landmark distance vectors*, *landmark order vectors*, and *geometric coordinates*. These use measurements to a small set of reference nodes, called landmarks.[2] The landmark distance vector approach (DV) represents the position information as simply the set of distances to the landmarks. The landmark order vectors use the same information, but the position information is a vector of landmark *id*s, ordered by increasing latency. This is the representation used in the Binning scheme described in [4]. Finally, coordinate based methods, such as those used by GNP [1] and Vivaldi [2], embed the nodes in an Euclidean geometric space which allows latencies between two nodes to be predicted fairly accurately by computing the Euclidean distance in that space.

The partitioning scheme assigns each service node A, to one infrastructure node called its *root* – $R(A)$ – that is responsible

[2]For the geometric coordinates, it is not necessary to have the set of landmarks as a reference. Vivaldi [2], for example, uses periodic measurements among random pair of nodes.

to storing the information about the service node. We design the partitioning such that service nodes that are close to each other are mapped to the same, or nearby, infrastructure nodes. This partitioning is optimized for proximity queries of the type 'what is the closest node to node A', or 'the k closest nodes to A', or 'the k nodes closer to A than a given threshold'. These queries can restrict how many different infrastructure nodes are queried, or a maximum execution time. The general approach to answering the queries is to direct the queries relative to node A to $R(A)$. As an added benefit, if A is issuing a query about itself, such as 'the closest node to me', it is likely, according to our partitioning schemes, that $R(A)$ is closer to A than other infrastructure nodes. We investigate two different strategies to perform the partitioning, and how to do querying in each case, which we describe below.

Closest Partitioning This is the simplest heuristic for partitioning: each service node gets mapped to its closest infrastructure node in terms of latency, analogously to a Voronoi diagram in a geometric space. While intuitive, depending on the topology it is not necessarily true that every pair of closest nodes will be mapped to the same infrastructure node. For example, in Figure 1, node B is the closest to A, but both are closest to different infrastructure nodes. For querying the data, when there is a need to expand the search to other infrastructure nodes, we propose that the first contacted infrastructure node sort the other infrastructure nodes in order of proximity to the query, and embed this 'source route' in the query which is forwarded to these nodes recursively.

DHT-based Partitioning The second strategy is to use a Distributed Hash Table (DHT) [9]–[12] abstraction to aid in the partitioning and querying. A DHT has a number of participating nodes that store key, value pairs in a distributed way. Both the nodes and the keys for the data are given coordinates in the same underlying geometry [13], and each key is associated with the DHT node that is closest in the geometry. Routing to a given key is performed in the geometry by choosing neighbors that will decrease the 'distance' to the destination.

We place the infrastructure nodes and the data about the service nodes in the same DHT *id* space. This assumes there is a mapping between a node's network position information and the DHT's underlying geometry. The infrastructure nodes form a DHT with their *id*s derived from their true network position. Subsequently, the service nodes insert their data in this DHT using their mapped network positions as keys. The DHT implicitly defines a partitioning, as each key is the responsibility of a DHT node; we call $R(A)$ the infrastructure node responsible for the key of node A. A query for the closest node to a given node A is made using the mapped network position of the node A. This query is routed by the DHT to the root node of the query coordinates, the closest in the DHT's *id* space. If the mapping between network positions and the DHT's *id* space preserves the distance information, *i.e.*, nodes that are close in the network are likely to be close in the DHT as well, then the reached infrastructure node will have information about other nodes which are close to the queried node, and the query will likely be answered locally.

TABLE I
MAPPINGS POSITIONS TO THE 1D DHT.

Position Representation	Mapping
Distance Vectors	Hilbert Curves
Order Vectors	Recursive Partitioning
Coordinates	Hilbert Curves

The performance of this scheme depends directly on the underlying DHT geometry, on the representation of network position, and on the mapping between the two. We use previous work on representing network positions, and describe how we map the representations into a DHT. We only evaluate mappings using a one dimensional DHT similar to Chord [9] in this paper, and leave other DHT geometries for future work.

For the *distance vectors* and *geometric coordinates*, we use Hilbert space filling curves to map the higher dimensional vectors into one dimension. A Hilbert curve produces a one to one correspondence between points in a sequence and points in the original space. Points in the original space that are close to each other have a good chance of being close in the curve as well, even though that is not possible in all cases. The *landmark order vectors* require a different mapping strategy, as they are permutations of the landmark *id*s, and not actual coordinates. We use a mapping introduced in [4] consisting of a recursive partitioning of the space. Table I summarizes these mappings.

Practical Considerations In the closest partitioning, finding the closest infrastructure node to a service node is a smaller instance of the original problem. We assume all infrastructure nodes know about the other infrastructure node's positions. In this way, any infrastructure node can determine which one is the closest infrastructure node to a given querying node, and redirect the query accordingly. We argue that there are no scalability issues in this smaller instance because the number of infrastructure nodes is bound to be one or two orders of magnitude smaller than the number of service nodes.

In both schemes, service nodes need to know how to find at least one infrastructure node, and this can be done by using DNS, for example. This information can then be cached for direct future access. Also, in both schemes can use a soft-state approach to maintaining the information. The service nodes will periodically update their information to the responsible infrastructure node, and service node failures can be detected by timeouts.

IV. EVALUATION

Recalling the tradeoffs involved in the partitioning of the position database, we use the following metrics when evaluating the different schemes. The **Fraction of times the root is the closest infrastructure node** and **Latency to** $R(A)$ metrics indicate how quickly nodes get to the responsible infrastructure nodes. The globally best answer should be in the first infrastructure node queried as much as possible, so we measure the **Fraction of times** $R(A)$ **has the information to the closest node to** A**,** $C(A)$. If the best answer is not in

$R(A)$, we use the **Effectiveness at** $R(A)$ metric to indicate the latency of the closest node stored at the first hop (R(A)), over the latency of the globally closest node in the system. If we have to perform a search, we are also interested in the **Hops to find closest node** metric. Finally, to measure the imbalance in the load imposed to each infrastructure node, we also look at the **Standard Deviation in the cluster sizes**.

We evaluate the performance of the different schemes using two datasets obtained from wide-area Internet measurements, one based on ping measurements [14] using 175 Planetlab [15] nodes and the other based on the King dataset [16]. We also performed evaluation using a 10,000 node transit-stub topology. Due to space considerations we only include the results for the King dataset, but the other results can be found in [17], and show similar trends. The King dataset is a set of latency measurements between 1740 DNS servers, used originally for evaluation in [2]. The data is available at [16], and uses the methodology described by Gummadi in [18] for determining the latency between two hosts A and B, from a third node C, using recursive DNS queries.

Given the matrix of inter-node latencies from the dataset, the first step is defining the landmark nodes for the experiment. These landmark nodes will determine the landmark vectors, the landmark order vectors, and the basis from which to calculate the GNP coordinates. We selected 12 landmarks by enforcing a maximum minimum separation criterion. The set of landmark nodes and the distances from all other nodes to them define the different representations of the position information.

For each node we obtain 12-component *distance vectors*, and 12-component *order vectors*. These are respectively mapped to a one dimensional coordinate space using Hilbert curves and the recursive partitioning, as described in Section III. The landmark vectors are also used as input to the GNP software [1], which we use with the default settings to produce a 7 dimension geographic coordinate.[3] These coordinates are also mapped to one dimension using Hilbert curves. To serve as a baseline and a comparison on how unintelligent mappings should behave, we also perform a random mapping, by assigning each node a position drawn uniformly at random from the interval $(0, 1)$.

We simulated the querying of the database, with each service node querying the infrastructure for the closest node to it. For each experiment, a given number I of infrastructure nodes is selected at random from the set of all nodes,[4] and all our results for a given I are averaged over 5 different selections of infrastructure nodes.

For the DHT cases, the simulation proceeds using a simplified ring topology: the $(0, 1)$ interval is mapped to a circle much like what Chord does, the infrastructure nodes form the DHT, and the service nodes' information is mapped as described previously. When performing a query for the closest node to A, we start at the partition $R(A)$, and do an expanding ring search, looking at the two neighbors of $R(A)$ in the ring, and moving outwards at each step, until we find the global closest node.

For the case of the closest partitioning, the partitions are obtained by assigning each node to its closest infrastructure node. We use the same set of infrastructure nodes as in the DHT simulations for comparison. As there is no implicit relationship among the different partitions, the search, when the globally closest node is not found in the partition of the responsible infrastructure node, is performed by following a linear sequence of infrastructure nodes, ordered according to the distance to the queried node.

A. Results

In this section we compare the results of the Closest Heuristic (named 'Closest' in the graphs) and the random partitioning ('Random'), with 4 different mappings into the 1 dimensional DHT: Recursive Partitioning for order vectors ('OV Recursive'), Hilbert mapping of Distance Vectors, both under the original and a logarithmic scaling ('DV Hilber Lin', and 'DV Hilbert Log', respectively), and finally a Hilbert mapping of GNP obtained coordinates, 'GNP Hilbert'.

We ran simulation in which all true distances are know between the nodes, and the restriction is just the information that is available in each infrastructure node. In this way we are able to isolate the effects that the distribution has on the quality and cost of the results, independently of the effects of the particular technique to actually find the closest nodes.

Figure 2(a) shows the fraction of times a node and its closest node have the same responsible infrastructure node. This is very important, for when this happens the query can be answered without any additional hops. As expected, the 'Random' partitioning behaves approximately like the inverse of the number of infrastructure nodes. The behavior of all the DHT mappings is similar, within a range of 10% of each other. There results for Closest, on the other hand, are significantly better, stabilizing at around 60% to 70%.

If the globally closest node is not in the same infrastructure node, it is interesting to know how well can one do if, instead of searching other infrastructure nodes for the best global answer, one settles for the best answer in the first infrastructure node. This is depicted in figure 2(b). In the figure, we can see the latency of the best answer in the first infrastructure node, divided by the best global answer. We notice that Closest stabilizes quickly even with many infrastructure nodes (smaller partitions), signalling that the partitioning is quite effective in preserving the distance relationships.

Next we look at the IP latency between node A and the responsible infrastructure node $R(A)$. This latency influences the querying process, since all queries will incur this latency cost. Figure 2(c) shows the fraction of times $R(A)$ is the closest infrastructure node, and figure 3(a) has the actual latency between A and $R(A)$ (not necessarily the closest).

[3]This has been found in [1], for example, to be suitable to capture the structure of Internet latencies well.

[4]As we select infrastructure nodes randomly from the set of all nodes in the measurements, we expect that these will follow the same distribution in terms of network position as the nodes themselves: as a result, there will be more infrastructure nodes where there are more nodes.

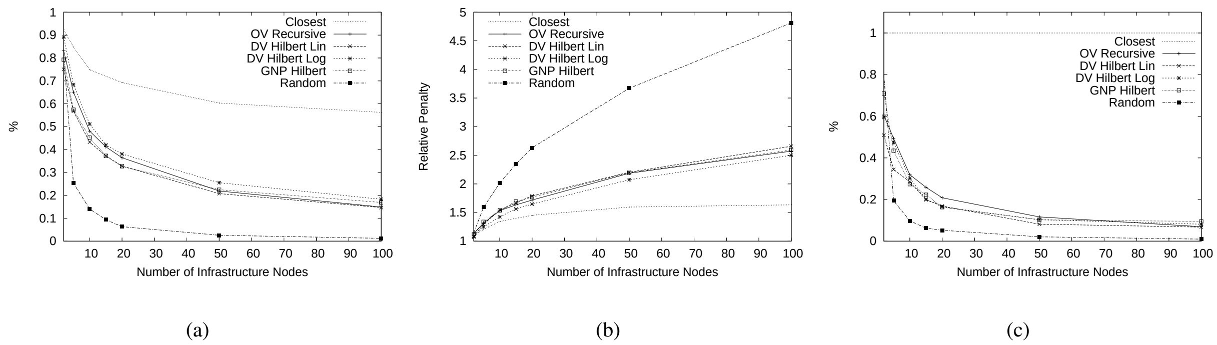

Fig. 2. (a) Fraction of times the global closest node is in the same infrastructure node ($R(A) = R(C(A))$). (b) Latency penalty if selecting the closest at $R(A)$. (c) Fraction of times $R(A)$ is the closest infrastructure node to A.

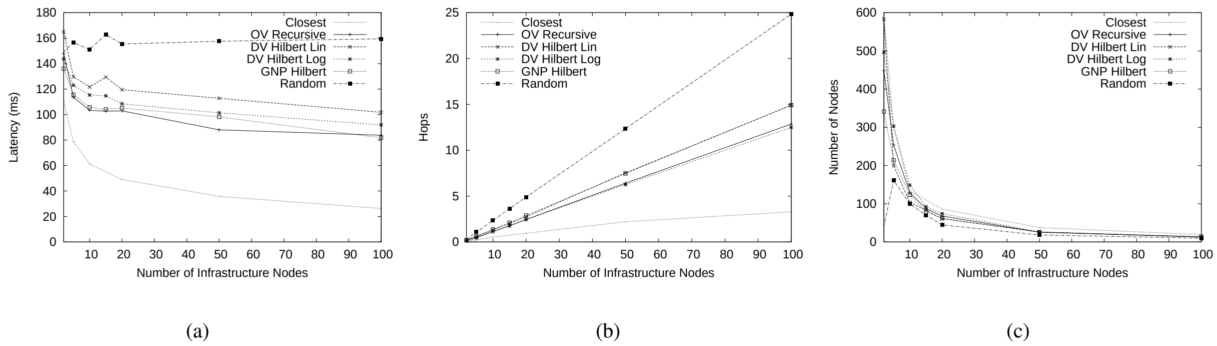

Fig. 3. (a) Latency between A and $R(A)$. (b) Number of infrastructure hops for finding the globally closest node. (c) Cluster load balancing: standard deviation in the number of nodes per infrastructure node.

Figure 3(b) concerns the search performance, and is important to set guidelines on how far one should go to get an answer reasonably close to the globally best answer. It shows the number of hops in the infrastructure to find the globally closest node that answers the query. The performance for closest is again very good, and even with 100 infrastructure nodes, the closest is within less than 5 hops.

Finally, figure 2(c) shows the standard deviation of the number of service nodes assigned to each infrastructure node, an indication of load imbalance. We notice tradeoff between the performance of the scheme according to other metrics and the variation in cluster size. Random has the smallest variance, as expected, while closest has the largest variance.

V. Conclusions and Future Work

From our initial results, we observe that partitioning of a position information database can have the benefits of distributing the update and querying load, while producing answers that are close to the best answer with complete information an reducing the querying latency by up to 7 times. Coupled with the existing distributed techniques to obtain the position information, we can have a scalable, distributed solution that can be used a building block for many applications. The closest partitioning is quite attractive in all of our performance and cost metrics, except for a disadvantage in terms of load balancing. We attribute part of the gap in performance of the different DHT mappings to the reduction in dimensionality to 1 dimension, which introduces some error in the estimation of distances. Also, the search employed by the closest partitioning, sorting the infrastructure nodes by proximity to the query, requires more information about the infrastructure than the search employed by the DHT. In all fairness the DHT solutions could be more scalable for this reason that the closest strategy, but some facts encourage us to implement the closest partitioning: its simplicity, the gap in the performance compared to the other strategies, and the fact that it's scale will be orders of magnitude less than the number of service nodes, making it a quite practical technique. The DHT-based partitioning can leverage all of the DHT techniques to maintain the infrastructure connectivity and the partitioning in the face of nodes failing and being added to the infrastructure. We did not take into account these further costs involved in maintaining the DHT.

Future work involves implementing and deploying a prototype system, in PlanetLab for example, to test the performance and scalability of the schemes in a real environment. We also want to investigate other DHT geometries that can represent higher dimensional data in a better way, to see if we can have the good performance of the closest partitioning.

References

[1] T. S. E. Ng and H. Zhang, “Predicting internet network distance with coordinates-based approaches,” in *Proceedings of the INFOCOM 2002*, June 2002.

[2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: A decentralized network coordinate system,” in *Proceedings of Sigcomm 04*, August 2004.

[3] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, “Lighthouses for scalable distributed location,” in *Proceedings of the IPTPS 2003*, Berkeley, CA, February 2003.

[4] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, “Topologically-aware overlay construction and server selection,” in *Proceedings of the IEEE INFOCOM 2002*, June 2002.

[5] L. Tang and M. Crovella, “Virtual landmarks for the internet,” in *Proceedins of the Internet Measurement Conference*, October 2003.

[6] P. Ganesan, B. Yang, and H. Garcia-Molina, “One torus to rule them all: Multidimensional queries in p2p systems,” in *Proceedings of the Seventh International Workshop on the Web and Databases, WebDB 2004*, 2004, pp. 19–24.

[7] R. Heubsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica, “Querying the internet with pier,” in *Proceedings of the 29th VLDB Conference*, 2003.

[8] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, “Scalable wide-area resource discovery,” UC Berkeley, Tech. Rep. UCB//CSD-04-1334, July 2004.

[9] 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 the ACM SIGCOMM 2001*, San Diego, CA, August 2001, pp. 149–160.

[10] A. I. T. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer sysstems,” in *Proceedings of IFIP/ACM Middleware*, Heidelberg, Germany, November 2001.

[11] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: A global-scale overlay for rapid service deployment,” *IEEE JSAC*, 2003, to appear.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, , and S. Shenker, “A scalable content-addressable network,” in *Proceedings of the ACM SIGCOMM 2001*, August 2001.

[13] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, “The impact of dht routing geometry on resilience and proximity,” in *Proceedings of the ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

[14] J. Stribling, “Planetlab all pairs ping data,” http://www.pdos.lcs.mit.edu/ strib/projects.html.

[15] PlanetLab, http://www.planet-lab.org.

[16] T. M. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling, “King dataset,” August 2004, http://www.pdos.lcs.mit.edu/p2psim/kingdata/.

[17] R. Fonseca, P. Sharma, S. Banerjee, S.-J. Lee, and S. Basu, “Distributed querying of internet distance information,” HP Labs, Palo Alto, Tech. Rep. HPL-2005-43, 2005.

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble, “King: Estimating latency between arbitrary internet end hosts,” in *Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002)*, Marseille, France, November 2002.