A Lightweight Approach to Network Positioning

Bernard Wong, Emin Gün Sirer
Dept. of Computer Science, Cornell University, Ithaca, NY 14853

Abstract— **This paper describes a peer-to-peer overlay network for performing location-aware node and path selection in large-scale distributed systems. Our system, Meridian, provides a simple, lightweight and scalable framework for keeping track of location-information for participating nodes. The framework is based on local, relative coordinate systems in multi-resolution rings, direct measurement with scalable node-to-node handoff, and gossip protocols for dissemination. Large scale simulations and an implementation deployed on PlanetLab show that the framework can locate the closest node to given target with less than a 5ms median error, and the simplicity of the approach lends itself to a compact implementation.**

I. Introduction

Selecting nodes based on their position in the network is a commonly encountered problem in many distributed systems. Recent work has focused on network embeddings, which map high-dimensional network measurements into an address in a smaller Euclidian space. For instance, recent work in network positioning [26, 11, 24, 36, 34, 28, 10, 27] maps a large vector of node-to-node latency measurements into a single point in a d-dimensional space.

The driving application for much of this work is to find the closest peer to a reference node in a given, possibly very large, set of peers. Finding the closest peer is a basic operation in content distribution networks (CDNs) [20], large-scale multiplayer games [23], and some peer-to-peer overlays [19, 21, 7, 6]. Efficiently discovering the closest peer can significantly reduce latency, bandwidth, and network load; for instance, a geographically distributed peer-to-peer web crawler can reduce crawl time and minimize network load by delegating the crawl to the peer closest to each target web server. The state-of-the-art approach to locating the closest server in a scalable manner is to explicitly compute a network embedding for all server nodes and the target location, and then to route through a CAN-like [29] substrate to the node closest to the target. This approach incurs errors in the embedding stage and requires a heavyweight overlay in the routing stage that, to date, has not been implemented or deployed.

This paper introduces a lightweight and scalable framework for finding the closest node to a given target point in a distributed system. Called Meridian, this system is based on a loosely-structured overlay network, uses direct measurements instead of a network embedding, and builds many local coordinate systems instead of an absolute coordinate space[1]. It performs network positioning simultaneously with query routing to efficiently find the closest node in a set of peers without explicitly performing an embedding into a global coordinate system. It is robust in the presence of churn, supports hosts behind firewalls, and incurs errors comparable to the published error rates for network embedding schemes.

Meridian is composed a routing scheme based on multiresolution rings, an algorithm for maximizing the geographic diversity in each ring, and a lightweight and scalable gossip protocol for membership updates. Each Meridian node keeps track of a fixed number of peers and organizes them into concentric rings of exponentially increasing radii. A query is matched against the relevant nodes in these rings, and optionally forwarded to a subset of the node's peers. A scalable gossip protocol is used to notify other nodes of membership in the system.

We evaluate Meridian through both large-scale simulations based on actual inter-node latency data collected from the Internet, and a full implementation deployed on PlanetLab [2]. Our simulations are based on node-to-node round-trip latency measurements for 4000 nodes and 16 million node pairs on the Internet using the King [17] measurement technique. We use this large-scale data set to evaluate the Meridian approach and show that Meridian is fast, scalable and accurate. We measure the accuracy of our system using a relatively smaller deployment on 166 PlanetLab nodes. The Meridian approach lends itself to a relatively simple implementation. The whole implementation is only 2500 lines, with most of the line count stemming from support for firewalled hosts, and is structured as a self-contained module that can be linked into applications.

The next section describes the architecture and operation of the Meridian overlay. The system is simple, loosely-structured, and entails modest resources for maintenance. Section III shows how this system dynamically routes queries to the the closest server to a given node. Section IV evaluates the proposed system with measurements from a large-scale network study. Although less general than GNP combined with CAN, we show that

[1] We use the term "location" to refer to a node's position in the Internet as defined by its roundtrip latency to other nodes. While Meridian does not assume that there is a well-defined location for all nodes, our illustrations depict a single point in a two-dimensional space for clarity.

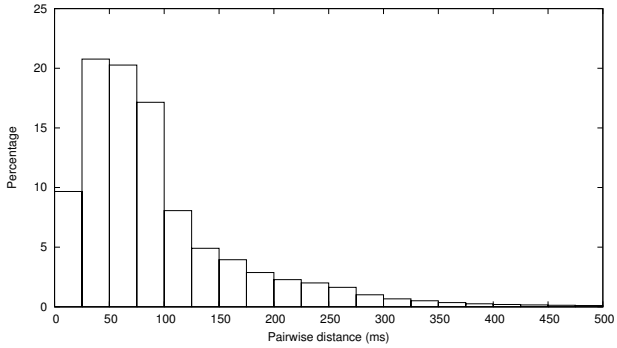

Fig. 1. Pair-wise latency histogram at 25ms intervals for 4000 geographically diverse DNS servers.

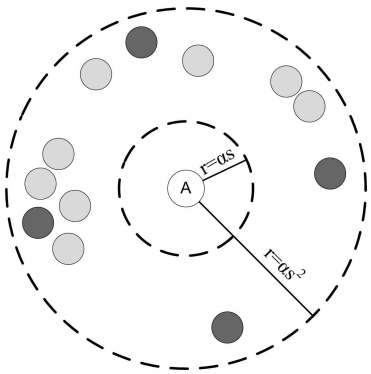

Fig. 2. Each Meridian node keeps track of a fixed number of other nodes and organizes these nodes into concentric, non-overlapping rings of exponentially increasing radii. Within a given ring, a set of nodes that span a large amount of space (dark) are more desirable than a more limited subset (light).

Meridian incurs errors less than those incurred by GNP in the network embedding phase.

II. FRAMEWORK

The basic Meridian framework is based around three mechanisms: a loose routing system based on multi-resolution rings on each node, an adaptive ring membership replacement scheme that maximizes the usefulness of the nodes populating each ring, and a gossip protocol for node discovery and dissemination.

A. Multi-Resolution Rings

Each Meridian node keeps track of a small, fixed number of other nodes in the system, and organizes this list of peers into concentric, non-overlapping rings. The ith ring has inner radius $r_i = \alpha s^{i-1}$ and outer radius $R_i = \alpha s^i$, for $i > 0$, where α is a constant, and $r_0 = 0$ for the innermost ring. Each node keeps track of a finite number of rings; all rings $i > i^*$ for a system-wide constant i^* are collapsed into a single, outermost ring that spans the range $[\alpha s^{i^*}, \infty]$.

Meridian nodes measure the distance d_j to a peer j, and place that peer in the corresponding ring i such that $r_i < d_j \leq R_i$. This sorting of neighbors into concentric rings is performed independently at each node and requires no fixed landmarks or distributed coordination. There is an upper limit of k on nodes kept in each ring, where peers are dropped from overpopulated rings; consequently, Meridian's space requirement per node is $O(1)$.

The rationale for exponentially growing ring radii is derived in part from the observed distribution of inter-node distances on the Internet and in part from the types of location-related queries that the Meridian framework is expected to handle. Figure 1 shows a histogram of 16 million pair-wise node distance measurements from DNS servers on the Internet. The number of nodes that are at a given distance from a node drops exponentially with distance. An exponentially increasing radius makes the total number of rings per node manageably small and i^* clamps the total number of rings at a constant. The ring structure also recognizes the decreasing importance of fine grain distance partitioning for far away nodes.

B. Ring Membership Management

The number of nodes per ring, k, represents an inherent tradeoff between accuracy and overhead. A large k increases a node's information about its peers and helps it make better choices when routing queries. On the other hand, a large k also entails more state, more memory and more bandwidth at each node. Within a given ring, node choice has a significant effect on the performance of the system. For instance, if the nodes within a given ring are clustered together, their utility is very small, despite their additional cost. A key principle, then, is to promote geographic diversity within each ring.

Meridian achieves geographic diversity by periodically reassessing ring membership decisions and replacing ring members with alternatives that provide greater diversity. Within each ring, a Meridian node not only keeps track of the k primary ring members, but also a constant number l of secondary candidates. Periodically, all $k + l$ ring members measure their distances to all other members of the same ring. A node i will thus obtain its distance d_j^i to another node j, for all $0 \leq i, j \leq k + l$. These measurements are combined together to yield coordinates for each node in a Lipschitz embedding [3]. In a Lipschitz embedding, the coordinates of node i simply consist of the tuple $< d_1^i, d_2^i, ..., d_{k+l}^i >$, where $d_i^i = 0$. A Lipschitz embedding is trivial to construct and, unlike a GNP coordinate, is precise – embedding error is zero since all distances are preserved and there is no mapping from high-dimensional data to a lower number of dimensions. Having computed the Lipschitz coordinates for all of its members in a ring, a Meridian node can then determine the subset of k nodes

that provide the most geographic diversity. We quantify geographic diversity through the hypervolume of the k-polytope formed by the selected nodes. For small k, it is possible to determine the maximal hypervolume polytope with k vertices by considering all possible polytopes; instead, we take a simple, greedy approach that works well in practice for large k. A node starts out with the $k + l$ polytope, and iteratively drops the vertex (and corresponding dimension) whose absence leads to the smallest reduction in hypervolume until k vertices remain. The remaining vertices are designated the new primary members for that ring, while the remaining l nodes become secondaries. This computation can be performed in linear time using standard computational geometry tools [8]. The ring membership management occurs in the background and its latency is not critical to the correct operation of Meridian.

C. Gossip Based Node Discovery

The use of a gossip protocol to perform node discovery allows Meridian nodes to be loosely connected, highly robust and allows membership change propagation to be inexpensive. Our gossip protocol is based on an anti-entropy push protocol [14], but relaxed to remove the need for comparison of ring sets between two gossiping nodes. This relaxation is possible as our goal is not for each node to discover every node in the system, but simply for each node to discover a diverse set of other nodes.

Our gossip protocol works as follows:

1. Each node A randomly picks a node B from each of its rings.
2. A sends a gossip packet to B containing a randomly chosen node from each of A's rings.
3. On receiving the packet, node B determines its latency to A and to each of the nodes contained in the gossip packet from A through direct probes.
4. After sending a gossip packet to a node in each of its rings, node A waits until the start of the next gossip period and then begins again from step 1.

In step 3, node B sends probes to A and to the nodes in the gossip packet from A regardless of whether B has already discovered these nodes. This is to ensure that stale latency information can be replaced, as latency between nodes on the Internet changes dynamically. The newly discovered nodes are placed on B's rings as secondary members, subject to FIFO replacement.

The period between gossip cycles is initially set to a small value in order for new nodes to quickly propagate their arrival to the existing nodes. The new nodes gradually increase their gossip period to the same length as the existing nodes. The choice of a gossip period depends on the expected rate of latency change between nodes and expected churn in the system.

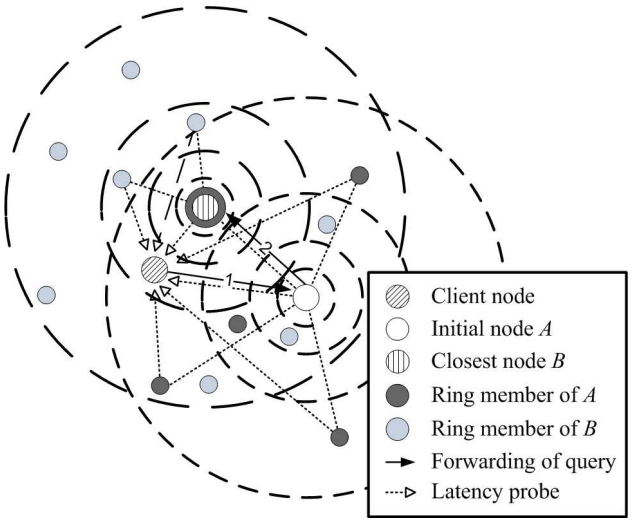

Fig. 3. A client sends a find closest peer request to an arbitrary Meridian node A, which determines its latency d to the client and probes its ring members between $\frac{d}{2}$ and $\frac{3d}{2}$ to determine their distances to the client. The request is forwarded to the closest node thus discovered, and the process continues recursively until no closer node is detected.

III. ROUTING

Meridian locates the closest peer by performing a multi-hop search where the node at each hop exponentially reduces the distance to the target. This is similar to searching in structured peer-to-peer networks such as Chord [35], Pastry [31] and Tapestry [38], where each hop brings the query exponentially closer to the destination, though in Meridian the routing is performed using physical latencies instead of numerical distances in a virtual identifier space. Meridian uses an acceptance threshold β, which serves a purpose similar to the routing base in structured peer-to-peer systems; namely, it determines the reduction in distance at each hop.

When a Meridian node receives a client request to find the closest peer, it determines the latency d between itself and the client. Once the latency is determined, it locates its corresponding ring j and simultaneously queries all nodes in that ring, as well as all nodes in the adjacent rings $j - 1$ and $j + 1$ whose distances to the origin are within $\frac{d}{2}$ to $\frac{3d}{2}$. These nodes measure their distance to the target and report the result back to the source. Nodes that take more than $2d$ to provide an answer are ignored, as they cannot be closer to the target than the source.

The route acceptance threshold is met if one or more of the queried peers is closer than β times the distance to the client, and the client request is forwarded to the closest peer. If no peers meet the acceptance threshold, then routing stops and the closest peer currently known is chosen. Figure 3 illustrates the process.

Meridian is agnostic to the choice of a route acceptance threshold β, where $0 \leq \beta < 1$. A smaller β value reduces

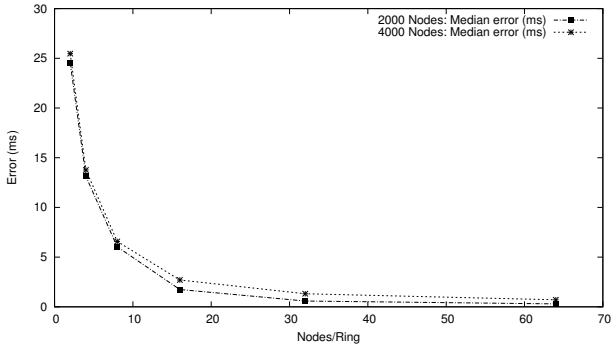

Fig. 4. The error drops sharply and flattens out at thirty-two nodes per ring for both curves.

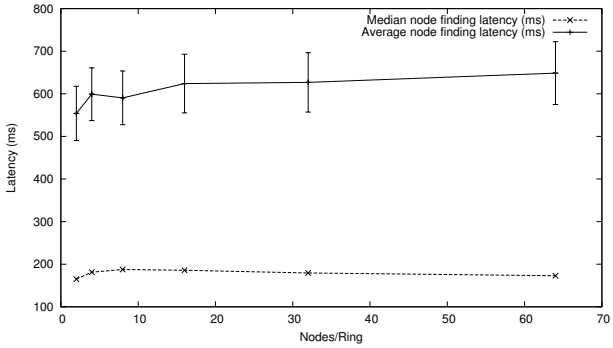

Fig. 5. Increasing the number of nodes per ring introduces a small corresponding increase in the average node finding latency, where the median latency is relatively unaffected.

the hop count, as fewer peers can satisfy the requirement, but introduces additional error as the route may be prematurely stopped before converging to the closest peer. A larger β may reduce error at the expense of increased hop count.

IV. EVALUATION

We first evaluate Meridian through large scale simulations based on inter-node latency data collected from 4000 Internet hosts. We collected pair-wise round trip latency measurements between 4000 DNS servers, spanning 16 million node pairs, using the King technique [17]. The study was replicated 9 times from 9 different PlanetLab [2] nodes across North America, with the median value of the 10 runs taken for the round-trip time of each pair of nodes. The system was configured with $k = 32$ nodes per ring, $l = k$, $i^* = 9$ rings per node, size of the innermost ring $s = 2$ms, $\alpha = 1$ms, $\beta = \frac{1}{2}$, and probe packet size of 100 bytes for 25000 queries. The 95% confidence interval is shown for each of the mean values that are presented. All references to latency in this section are in terms of round trip time.

The accuracy, as well as bandwidth consumption, of Meridian depends critically on k, the number of primary nodes per ring. Figure 4 plots the error, defined as the distance between the closest node determined by Meridian and the closest node determined through global information, as a function of the number of nodes per ring. It shows that even modest choices for k yield high accuracy.

High accuracy must also be coupled with low latency for interactive applications that have a short lifetime per query and low tolerance for initial setup time. Figure 5 plots the latency, measured as the sum of the maximum latency probe at each hop plus the hop latency, versus the number of nodes per ring. It shows that the majority of queries can be completed in less than 200ms at a system size of 4000 nodes.

Bandwidth consumed per query is one of the most significant limiting factors limiting the scalability of the system, as it determines the aggregate network load posed by Meridian. Figure 6 shows that the total bandwidth consumed per query query grows sub-linearly with system size, with 4000 nodes requiring only 10.5 KB per query.

V. IMPLEMENTATION

We have implemented and deployed Meridian on PlanetLab. The implementation is small, compact and straightforward; it consists of approximately 2500 lines of C++ code. Most of the complexity stems from support for firewalled hosts.

Hosts behind firewalls and NATs are very common on the Internet, and a system must support them if it expects large-scale deployment over uncontrolled, heterogenous hosts. Meridian supports such hosts by pairing each firewalled host with a fully accessible peer, and connecting the pair via a persistent TCP connection. Messages bound for the firewalled host are routed through its fully accessible peer – a ping, which would ordinarily be sent as a direct UDP packet or a TCP connect request, is sent to the proxy node instead, which forwards it to the destination, which then performs the ping to the originating node and reports the result. A node whose proxy fails is considered to have failed, and must join the network from scratch to acquire a new proxy. Since a firewalled host cannot directly or indirectly ping another firewalled host, firewalled hosts are excluded from ring membership on other firewalled hosts, but included on fully-accessible nodes. Since routing through another host increases the latency of a query, Meridian adds a constant γ to packet timeouts to compensate (for a total of $2d + \gamma$); in our implementation, $\gamma = 2d$.

We deployed the Meridian implementation over 166 PlanetLab nodes. We benchmark the system with 800 target web servers drawn randomly from the Yahoo web directory, and examine the latency to the target from the node selected by Meridian versus the optimal obtained by querying every node. Overall, median error in Meridian is

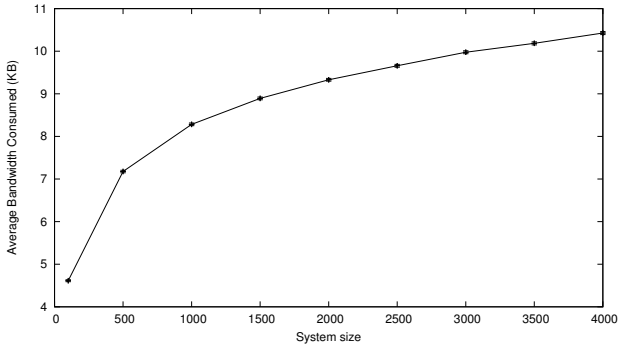

Fig. 6. Bandwidth requirements per query increase sub-linearly with system size.

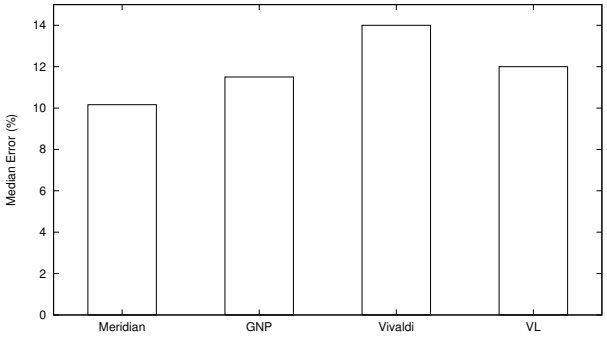

Fig. 7. Meridian's positioning error is comparable to the embedding error in network embedding systems. Sources: GNP Figure 12 of [26], Vivaldi Figure 5 of [11], and Virtual Landmarks Figure 9 of [36].

only 1.435ms, well within the simulation results, and corresponds to a 10.2% deviation from the optimal. To put these numbers in perspective, Figure 7 illustrates the percentage error of Meridian compared with the embedding error of several virtual coordinate systems. We assume that these systems incur no error in the routing phase, either through full-knowledge of the network ($O(N)$ state) or through a CAN-like substrate. Although the systems are not directly comparable due to different data sources, the graph shows the percentage error of the different systems are within the same order of magnitude of each other.

VI. RELATED WORK

The closest server selection problem has been the focus of much prior work. Dynamic server selection [4] proposes to probe all of the servers and pick the one with the lowest latency; a simple approach that scales linearly with the number of servers. A detailed simulation study [18] examines schemes, such as anycast, BGP-polling, and triangulation, to locate the nearest node; these schemes require infrastructural changes. IDMaps [16] can estimate the approximate distance between two IP addresses based on strategically placed tracer nodes, which require infrastructural changes to deploy, without direct measurement. Beaconing [22] leverages the triangle inequality to find the closest node to a given target; the beacon nodes require $O(n)$ state. Binning [30] proposes to store distances to well-known landmarks in DNS, and use them for geographic query resolution.

Recent work on network positioning can be categorized roughly into the landmark based systems such as GNP [26], ICS [24], Virtual Landmarks [36], Lighthouse [28], PIC [10] and NPS [27], and the simulation based systems such as Vivaldi [11] and BBS [34]. Both types can accurately embed nodes into a Euclidean coordinate space, which allows the distance between any two nodes to be determined without direct measurement. These coordinates are general, but need to be coupled with a geographically organized P2P substrate such as CAN [29] for scalable, position-based node discovery. There are inherent embedding errors in each of these systems, and the use of a substrate, such as CAN, significantly increases the complexity of the overall system.

Proximity based neighbor selection [7, 6] performs a proximity search using the node entries in the route table of a structured P2P system. The time and space complexity of two similar techniques are discussed in [19] and [21], but these techniques have not been evaluated with a large scale data set or implemented.

VII. SUMMARY

This paper describes Meridian, a lightweight and scalable framework for network proximity problems on the Internet. Meridian is based on a loosely structured overlay network. It uses direct measurements instead of coordinates to perform location-aware query routing without incurring the complexity and overhead of an embedding into an absolute coordinate system.

Both large-scale simulations and an actual implementation indicate that the system is effective – it incurs error comparable to or less than systems based on absolute embedding, is decentralized, requires relatively modest state and bandwidth, and locates nodes quickly. The implementation is compact, and the system can be incorporated easily into CDNs, online games, and other distributed systems where finding the closest node to a target is an essential building block. Overall, Meridian gains its high accuracy and simplicity from its use of many local coordinate systems, its resilience from a loosely structured overlay, and its performance from its use of geographic routing among multi-resolution rings. We are currently examining whether the lightweight approach advocated in this paper can be applied to other positioning-related problems.

REFERENCES

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles*, Banff, Canada, October 2001.

[2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Proceedings of Networked System Design and Implementation 2004*, San Francisco, CA, March 2004.

[3] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[4] R. Carter and M. Crovella. Server Selection Using Dynamic Path Characterization in Wide-Area Networks. In *Proceedings of IEEE INFOCOM 1997*, Kobe, Japan, April 1997.

[5] R. Carter and M. Crovella. On the Network Impact of Dynamic Server Selection. *Computer Networks*, 31:2529–2558, 1999.

[6] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. In *Technical Report MSR-TR-2003-82*, Microsoft Research, 2002.

[7] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. In *Technical Report MSR-TR-2003-52*, Microsoft Research, 2003.

[8] U. G. Center. QHull. UIUC Geometry Center, QHull Computational Geometry Package, http://www.qhull.org, July 2004.

[9] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS 2000*, Santa Clara, CA, June 2000.

[10] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *Proceedings of ICDCS*, Tokyo, Japan, March 2004.

[11] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical, Distributed Network Coordinates. In *Proceedings of Hotnets 2003*, Cambridge, MA, November 2003.

[12] W. Cui, I. Stoica, and R. Katz. Backup Path Allocation Based On A Correlated Link Failure Probability Model In Overlay Networks. In *IEEE International Conference on Network Protocols*, Paris, France, November 2002.

[13] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles*, Banff, Canada, October 2001.

[14] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of Distributed Computing*, Vancouver, BC, August 1987.

[15] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A Novel Server Selection Technique for Improving the Response Time of a Replicated Service. In *Proceedings of IEEE INFOCOM 1998*, San Francisco, CA, March 1998.

[16] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transactions on Networking*, 9:525–540, October 2001.

[17] K. Gummadi, S. Saroiu, and S. Gribble. King estimating Latency between Arbitrary Internet End Hosts. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop 2002*, Marseille, France, November 2002.

[18] J. Guyton and M. Schwartz. Locating Nearby Copies of Replicated Internet Servers. In *Proceedings of ACM SIGCOMM 1995*, Boston, MA, September 2002.

[19] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao. Distributed Object Location in a Dynamic Network. In *Proceedings of 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, Winnipeg, Manitoba, Canada, August 2002.

[20] K. Johnson, J. Carr, M. Day, and M. Kaashoek. The measured performance of content distribution networks. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.

[21] D. Karger and M. Ruhl. Finding Nearest Neighbors in Growth-restricted Metrics. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, Montreal, Quebec, Canada, May 2002.

[22] C. Kommareddy, N. Shankar, and B. Bhattacharjee. Finding Close Friends on the Internet. In *Proceedings of ICNP 2001*, Riverside, CA, November 2001.

[23] R. Lawrence. Running Massively Multiplayer Games as a Business. In *Keynote: In Proceedings of Networked System Design and Implementation 2004*, San Francisco, CA, March 2004.

[24] H. Lim, J. Hou, and C. Choi. Constructing Internet Coordinate System Based on Delay Measurement. In *Proceedings of ACM SIGCOMM Internet Measurement Conference 2003*, Miami, Florida, October 2003.

[25] P. Maniatis, M. Roussopoulos, T. Giuli, D. Rosenthal, M. Baker, and Y. Muliadi. Preserving peer replicas by rate-limited sampled voting. In *Proceedings of the Nineteenth Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[26] T. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of IEEE INFOCOM 2002*, New York, NY, June 2002.

[27] T. Ng and H. Zhang. A Network Positioning System for the Internet. In *Proceedings of USENIX 2004*, Boston, MA, June 2004.

[28] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, Feburary 2003.

[29] S. Ratnasamy, P. Francis, M. Hadley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[30] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proceedings of IEEE INFOCOM 2002*, New York, NY, June 2002.

[31] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware 2001*, Heidelberg, Germany, November 2001.

[32] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles*, Banff, Canada, October 2001.

[33] S. Savage, A. Collins, and E. Hoffman. The End-to-End Effects of Internet Path Selection. In *Proceedings of ACM SIGCOMM 1999*, Cambridge, MA, September 1999.

[34] Y. Shavitt and T. Tankel. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, April 2003.

[35] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[36] L. Tang and M. Crovella. Virtual Landmarks for the Internet. In *Proceedings of ACM SIGCOMM Internet Measurement Conference 2003*, Miami, Florida, October 2003.

[37] H. Weatherspoon, T. Moscovitz, and J. Kubiatowicz. Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems. In *Proceedings of International Workshop on Reliable Peer-to-Peer Distributed Systems*, Osaka, Japan, October 2002.

[38] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In *Technical Report UCB/CSD-01-1141*, UC Berkeley, April 2001.