Long Range Contacts in Overlay Networks*

Filipe Araújo
U. Lisboa
filipus@di.fc.ul.pt

Luís Rodrigues
U. Lisboa
ler@di.fc.ul.pt

June 2, 2005

Abstract

In this paper we present and evaluate a novel mechanism, called *Hop Level*, that creates and maintains long range contacts (LRCs) in overlay networks. The Hop Level mechanism owns the following characteristics: *i*) lazy creation of the LRCs, *ii*) support for unbalanced node distribution, *iii*) support for multidimensional spaces and *iv*) near-optimal path lenght/node degree trade-off. These characteristics make Hop Level specially suited for overlay networks that support range data queries (as opposed to distributed hash tables that only support exact queries) with one or more dimensions. Furthermore, and unlike previous similar work, Hop Level can handle churn very well, because it postpones creation of the LRCs until it is necessary. In this way, nodes that have short lives do not overload the network with their state update requests.

1 Introduction

Distributed hash tables (DHTs) have recently emerged as an important component for distributed systems. A DHT is a dictionary that outputs values in exchange of keys. A common aspect to the most well-known DHTs [9, 16, 14, 17, 6, 7, 13] is that they operate in the application layer as an overlay network. To overcome the limitations inherent to DHTs, some researches have proposed a shift to a more powerful paradigm: the distributed storage systems [4, 5, 8] (DSSs). Unlike DHTs that only perform exact queries, DSSs allow efficient range queries. As a consequence, when compared to a DHT, the design of a DSS is more complex. First, in a DSS, we cannot assume that data is uniformly distributed in space. Second, we cannot assume that entrance and departure patterns of data items will favor balancing. On the contrary, DHTs were based

*Selected sections of this report will be published in the Proceedings of the Proceedings of the Euro-Par 2005, Lisboa, Portugal, August 2005. This work was partially supported by LaSIGE and by the FCT project INDIQoS POSI/CHS/41473/2001 via POSI and FEDER funds.

on the assumption that consistent hashing would result in a perfect balance of node identifications and data items.

Often, in overlay networks, it is possible to distinguish between two different types of contacts: "nearby" contacts, forming a kind of connected lattice between nodes that have close virtual identifications, and "long range contacts" (LRCs) between nodes that have "distant" virtual node identifications. While the former type of contacts may be important in certain overlays, to ensure connectedness and routing convergence, short path lengths actually depend on the latter type of contacts. In fact, it is the capability to "jump" over many close nodes in a single hop that makes it possible to achieve short path lengths. Therefore, in this paper, we present the Hop Level mechanism, which creates and maintains LRCs in overlay networks. The Hop Level mechanism can be used in many different overlay networks to reduce path lengths, including DHTs and DSSs. Nevertheless, it is particularly well suited to DSSs, because it can cope with unbalanced distribution of nodes and it supports single as well as multi-dimensional data. We believe that this is one of the most innovative aspects of Hop Level, because most overlay networks are tied to unidimensional address spaces, where nodes must be numerically or alphabetically ordered (e.g., SkipNets [6]).

Since node degree and diameter of a network cannot be arbitrarily and simultaneously reduced, the trade-off between these two metrics is often used as a fundamental efficiency measure of an overlay network. For an $O(1)$ node degree, the expected diameter can be at best $O(\log n)$, while for an $O(\log n)$ node degree, the expected diameter cannot be shorter than $O(\log n / \log \log n)$ [7], for an n-node network. Given this limitation, path lengths of Hop Level achieve a nearly optimal trade-off with node degree. Furthermore, unlike existing overlay networks that implement DHTs and DSSs, when a node using the Hop Level mechanism enters the network, it postpones the creation of the LRcs to simplify the entrance. Later, it progressively creates the LRCs as they are needed to route real messages. In fact, lazy creation of the LRCs is one of the most significant aspects of Hop Level, as this reduces control traffic with only a minor effect on routing performance. In this way, behavior of Hop Level under churn is very good.

The remainder of the paper is organized as follows: Section 2 states the problem we are solving. Section 3 overviews previous work. Our long range contact mechanism is described and evaluated, respectively in Sections 4 and 5. Section 6 concludes the paper.

2 Problem Statement

Throughout this paper we will consider that routing convergence is ensured by nearby contacts already existing in the overlay network, e.g., as in [10] or [9]. Although these are examples of two-dimensional networks (of which we tested the Delaunay triangulation of [10]), there is however no restriction to the number of dimensions of the overlay network. A crucial point here is that distribution

of nodes does not need to follow any specific pattern.

Hence, we will consider the following conditions: *i*) nodes are organized into a multidimensional underlying overlay network having only nearby contacts; and *ii*) identification of nodes is arbitrary (as a result, distribution of nodes in space may be unbalanced). The goal of condition *ii* is to maintain locality, by preventing arbitrary conversion of node addresses from one space of identifications to another, e.g., by an hash function. There are many practical examples where this restriction holds. In a DSS, nodes may receive their identification according to the data items that they store. In [4], the overlay structure directly reflects the contents of the data, which is organized in a sequential order. In this way, it is possible to make range queries efficiently. On the contrary, hashing data to obtain some balance in a different identification space would defeat this goal. Another example where the condition *ii* holds occurs in systems where identification of a node bears some relation with its physical location, like in [3] or when using landmark ordering [13].

Furthermore, we will consider the use of a routing scheme where *i*) the preprocessing algorithm can only collect information of $O(1)$ nearby peers and $O(\log n)$ distant peers per node and *ii*) the routing algorithm will select, among the forwarding node's contacts (either short or long range), the one which is closest to destination in terms of Euclidean distances[1]. Given these conditions, our goal is to design a mechanism that creates and maintains a set of LRCs at each node such that routing convergence is guaranteed with $O(\log n)$ expected path lengths *despite* non-uniform node distribution. Moreover, each node should store $O(\log n)$ LRCs and this number must not depend on the size of the virtual identification space, but only on the nodes effectively existing in the system. Balancing the workload among the peers in the DSS is not a goal of this paper; such issue is orthogonal to our work and is already tackled in previous work, like [4].

Before presenting the Hop Level mechanism we will overview previous research in the topic of overlay networks to capture the relevant features that should be owned by efficient sets of LRCs.

3 Related Work

There is a huge body of work related with overlay networks and, in particular, with DHTs. In some DHTs it is possible to do an explicit separation between nearby and long range contacts (e.g., in DHTs based on a ring). However, there are also many other systems where this separation is only implicit or non-existent. In constrat to the previous cases, CAN [13] exhibits no LRCs but only short range contacts, thus having longer path lengths. To overcome this limitation, Xu and Zhang [17] proposed a mechanism called "expressways for

[1]There is no loss of generality in assuming Euclidean distances, as other metrics could also be used if more appropriate to the structure of the lattice, e.g., Manhattan distance or unidimensional virtual identification distance.

CAN" that augments basic CAN with LRCs. This work and others, like [9] and [11] are very similar in spirit to the Hop Level mechanism.

All the DHTs referred before assume a balanced distribution of nodes in space. Unlike these, LAND [1] copes with unbalanced distribution of nodes, but it does not meet the conditions stated in Section 2, because it hashes identifiers of objects. SkipNet [6] was also designed from scratch to cope with the unbalanced use of identification space. In fact, SkipNet is more appropriate to support a DSS, because it supports range queries. However, the identification space of a SkipNet is unidimensional and generalization to higher-dimensional spaces does not seem trivial. Unlike SkipNet, [4, 5, 8] have explicit support for complex load balancing mechanisms without impairing efficient range queries. Of these, only Mercury [5] supports multidimensional range queries. However, Mercury requires a different data structure (a ring of nodes) for each queriable attribute (including a copy of the data). When compared to these systems, support of multidimensional range queries is inherent to the Hop Level mechanism and does not need to be mapped to multiple unidimensional queries.

4 Hop Level LRCs Mechanism

We now describe our proposal to build LRCs in unbalanced overlays. Using our *Hop Level* mechanism, LRCs are established automatically whenever a message goes through b consecutive hops. Consider, for instance, that some node F is forwarding a message m to node N_1 originated at node S and destined to node D. If node F realizes that N_1 will be the b-th hop of m it triggers the creation of a LRC from S to N_1, denoted by $S \xrightarrow{1} N_1$. To do this F sends a control message to S. The process is repeated from N_1 onwards: if after b hops, message m reaches N_2, N_1 will create a LRC to N_2, $N_1 \xrightarrow{1} N_2$, and so on. Let us call these LRCs, *level-1 LRCs*. If the message path is very long, it may happen that a sequence of b *level-1 LRCs* occurs, for instance: $S \xrightarrow{1} N_1$, $N_1 \xrightarrow{1} N_2$, ..., $N_{b-1} \xrightarrow{1} N_b$. In this case, a new LRC from S directly to N_b should be created. This new LRC, $S \xrightarrow{2} N_b$, is one level above of the previous ones. This mechanism should be applied recursively for all levels. Hence, a LRC of level-l jumps over b^l hops.

To bound the number of LRCs per node, we limit the number of LRCs that exist in each level. This allows the number of LRCs to grow with the size of the network. The shape of this growth is evaluated in Section 5.

4.1 Algorithm

Our implementation of Hop Level algorithm requires a minimum of three variables per level l to be carried in each message m: the number of hops, $nh_m[l]$, the node that may receive a new LRC of that level, $s_m[l]$, and whether this node has space for an additional LRC, $a_m[l]$. Whenever level counter $nh_m[l-1]$ reaches the limit b, a new LRC, starting at $s_m[l]$ should be created. To conserve

Algorithm 1 Hop Level algorithm

{Executed at node F when forwarding m to node N}
{Control information carried in message m:}
{max_m — highest valid level; p_m — level of LRC used to reach F;}
{$\forall k \in [0, max_m] : nh_m[k], s_m[k], a_m[k]$ — resp., number of hops, first node and whether there are available slots in the first node for level-k;}

1: $l \leftarrow$ level of LRC from F to N ($F \xrightarrow{l} N$)
2: **if** $p_m = \perp$ **or** $p_m < l$ **then**
3: $max_m \leftarrow l + 1$; $lim \leftarrow max_m$
4: **else**
5: $lim \leftarrow p_m$
6: **end if**
7: **for all** $k \in \{l, \ldots, lim - 1\}$ **do**
8: $s_m[k+1] \leftarrow F$; $a_m[k+1] \leftarrow a_F[k+1]$; $nh_m[k] \leftarrow 0$
9: **end for**
10: $nh_m[l] \leftarrow nh_m[l] + 1$
11: **while** $nh_m[l] \geq b$ **do**
12: $nh_m[l] = 0$
13: **if** $a_m[l+1] > 0$ **then**
14: instruct $s_m[l+1]$ to create LRC $s_m[l+1] \xrightarrow{l+1} N$
15: **end if**
16: $l \leftarrow l + 1$;
17: **if** $max_m == l$ **then**
18: $max_m \leftarrow max_m + 1$; $nh_m[max_m - 1] \leftarrow 0$
19: $s_m[max_m] \leftarrow s_m[max_m - 1]$; $a_m[max_m] \leftarrow a_m[max_m - 1]$
20: **end if**
21: $nh_m[l] \leftarrow nh_m[l] + 1$
22: **end while**

space we do not discuss signaling cost here, but it is possible to leave some of this temporary information at the nodes to shorten messages.

When a forwarding node uses a LRC of level-l to send a message, it must check the LRC used by the previous hop node, say level-p. If $l > p$, neither one of the LRCs that preceded this hop can be used to create new LRCs (e.g., if a level-3 LRC is being taken after a previous level-2 LRC). Now, consider that message m is going to be sent along its b-th consecutive hop of level-l to node N. In this case, forwarding node F sends a control message to the node that initiated the sequence of level-l, prompting it to create a LRC of level-$(l+1)$ to node N. Then, node F sets the number of hops of level-l to 0 and increments the number of hops of level$(l + 1)$ by 1. Should this substituting hop become the b-th hop of level-$(l + 1)$, the same process is repeated for level-$(l + 1)$, and so on, until a level with fewer than b hops is reached.

To implement this algorithm, messages must carry the level p_m of the LRC used by the previous hop to reach F, and an indication of the highest level of the array that contains valid information, max_m. Each node F, when forwarding the message m to N, executes Algorithm 1. $a_F[k]$ is a boolean variable that indicates whether F has slots available at level k to store additional LRCs. If F is the source of the message, $F = S$, it is necessary to set previous level $p_m \leftarrow \perp$. In this case, the execution of the algorithm will initialize $max_m \leftarrow l + 1$, $s_m[max] \leftarrow S$, $a_m[max] \leftarrow a_S[max]$ and $nh_m[max - 1] \leftarrow 0$.

To maintain the LRCs evenly distributed in face of membership changes, we periodically delete the least recently used LRC of some randomly selected levels. In our experiments, path lengths did exhibit low sensitivity to variations of the

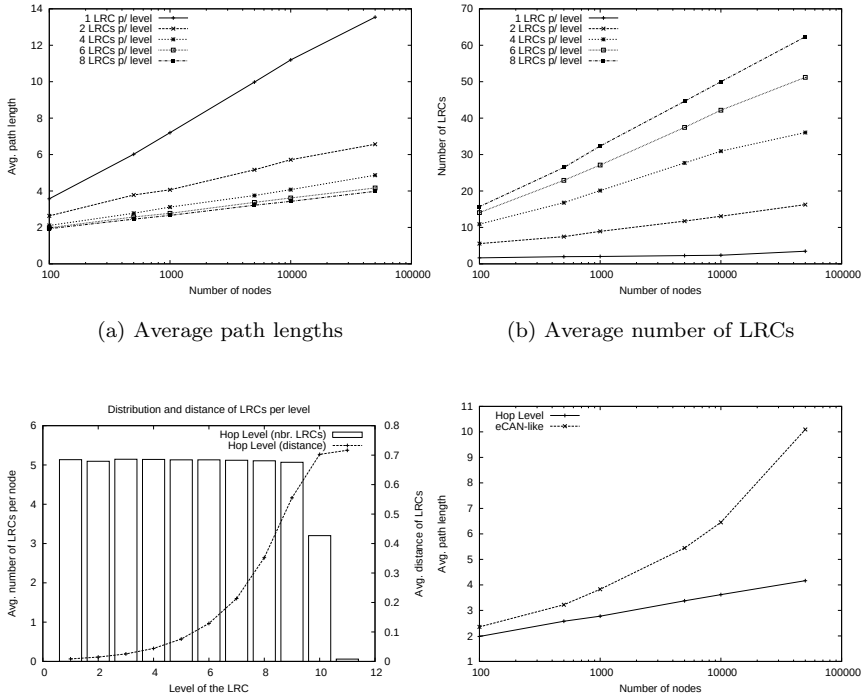
deletion period. Nodes should also purge hanging LRCs that point to neighbors that left. To do this, nodes can send periodic beacons to their neighbors. Alternatively, we can trade this beacon traffic by latency, by using, again, a lazy approach. In this latter solution, nodes only detect that a LRC is hanging when they try to use it. For the highest churn rates we tested in Section 5, when using a lazy apprach, 13.3% of the messages tried to follow hanging LRCs. This figure goes down to 1.2% for the lowest churn rate.

5 Evaluation

Experiment Settings In this section we experimentally evaluate Hop Level with $b = 2$. Most experiments, including the comparison with eCAN-like mechanism (to be presented ahead) use a Delaunay triangulation as the underlying lattice [10]. However, for benchmarking purposes we have also used a mapping of a two-dimensional space into a unidimensional ring. In our experiments we evaluate the following aspects: *i*) the behavior of Hop Level, when different limits for LRCs by level are used; this includes knowing the distribution of the LRCs by the levels; *ii*) the behavior of Hop Level when compared to the eCAN-like mechanism, both in balanced and extremely unbalanced scenarios; *iii*) the behavior of Hop Level in a ring; *iv*) the cost of the bootstrap mechanism of Hop Level and, finally; *v*) the behavior of Hop Level in dynamic settings, including settings with strong membership variation, i.e., under churn.

In the tests, arbitrary pairs of nodes exchange a large number of messages in networks with sizes ranging from 100 to $50,000$ nodes. To route the messages we have used the greedy routing algorithm, because it has good performance and it works both in the underlying lattice and with LRCs, without requiring any extensions. Furthermore, it agrees to the conditions of Section 2. Hence, next hop is always the neighbor (connected by a short or long range contact) closest to destination. To let Hop Level LRC scheme converge, and depending on the network size, we routed up to $1,000,000,000$ different messages and only used the final 3000 paths in the evaluation of path lengths. Nevertheless, we also show that our mechanism achieves good routing performance much earlier than that. To test unbalanced distributions of nodes we used a truncated Gaussian bivariate distribution with standard deviations of 0.01 in a $[0, 1] \times [0, 1]$ square.

Number of LRCs per Level The first aspect that we evaluate is the performance achieved by different configurations of the Hop Level mechanism. The goal is to determine the limit for the number of LRCs per level that ensures the most reasonable compromise between path lengths and node degrees. Figures 1(a) and 1(b) respectively show the average path lengths (in number of hops) and the average number of LRC used by each node for different network sizes and for different configurations of the Hop Level mechanism: with 1, 2, 4, 6 and 8 LRCs per level. We can see that all configurations achieve an approximately logarithmic/logarithmic trade-off (a logarithmic growth is represented by a straight line). We believe that this is quite an interesting aspect, because



(a) Average path lengths

(b) Average number of LRCs

(c) Distribution of LRCs

(d) Hop Level vs. eCAN-like

Figure 1: Path length and number of LRCs

it minimizes the need for manual configuration of parameters. In the rest of our experiments we set the limit to 6 LRCs per level. Figure 1(c) shows how many LRCs exist on the entire network and the average length of those LRCs for each hop level. To do this evaluation, we have used a $50,000$ node network with a balanced distribution of nodes, because a balanced distribution allows to reason in terms of distance. From the growth of the number of levels it is possible to determine the growth of the number of LRCs per node. Given that the distance growth from one level to the next is approximately exponential this figure points to the conclusion that the number of levels is approximately logarithmic.

Comparison with "eCAN-like" and Hop Level in a Ring To offer some comparative measurement, we ran our scheme against a benchmark mechanism called "eCAN-like". This benchmark results from an adaptation of the eCAN [17] logarithmic/logarithmic node degree/path length mechanism (whose applications most closely resemble those of our own algorithm). Although we made some simplifications to the original eCAN, we believe that our implemen-

tation of expressways mimics the eCAN LRC mechanism with enough accuracy to allow a fair comparison. The idea in eCAN-like is to make a first level division of the entire space in four big squares. Each node keeps LRC to the two neighboring squares. Then, the four big squares are further divided in other four smaller squares. This time, nodes inside squares have a total number of four LRC (above, below, right and left). This process is repeated for as many levels as wanted. In our context, we fixed the number of levels to 8, in a total of 30 LRCs. The actual LRC will be the node responsible for the central point of each neighboring square. Comparison of Hop Level against the eCAN-like is depicted in Figure 1(d), for unbalanced networks. The number of LRCs is not depicted because it is constant in eCAN-like. Bad behavior of the eCAN-like mechanism is easily explainable: density of LRCs is no longer enough near the center and routing to nearby nodes will tend to become linear with the number of hops in the lattice, instead of logarithmic. On the contrary, node distribution has a very little impact on Hop Level.

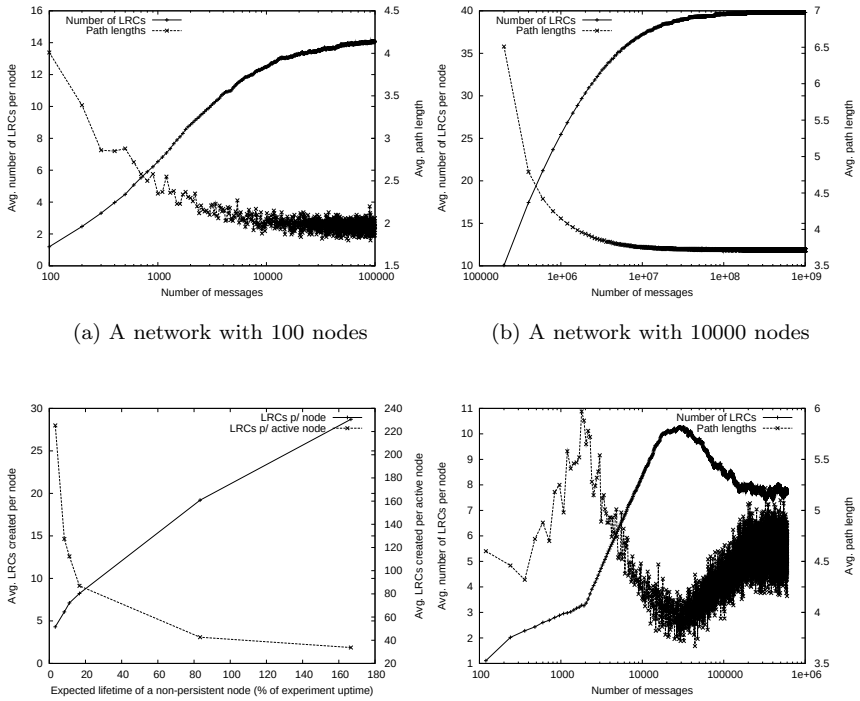
Due to lack of space, we do not show results of mapping a two-dimensional space into a ring (the same could be done for any number of dimensions). As expected, path lengths in a ring are also logarithmic, but paths are shorter in a multidimensional space due to the higher connectivity of nodes. This result is interesting not only for benchmarking, but also because it shows that hashing nodes into a ring can have its costs in performance (not to mention a possible loss of locality information).

Network Convergence Figures 2(a) and 2(b) depict for two network sizes the growth in the number of LRCs of the entire network and the reduction in the path lengths. In both cases, we can observe that for all network sizes under test, a short number of messages suffices to let the network reach a state similar to a steady state. For all network sizes we tested, path lengths within 3 times the optimal can be achieved before 5 messages have been generated by each node.

Dynamic Settings In this section we will use settings similar to the ones described in Araneola [12], which are based on real measurements [2, 15]. Hence, we assume that around 7% of the nodes are permanent. The remaining 93% of the nodes are non-permanent and can enter or leave the network at any instant and repeatedly do so. When a node enters the network it becomes active, when it leaves it goes to sleep state. When network starts, non-permanent nodes are neither active nor sleeping, but in a fourth state that we can call as *out*. This means that the network starts with 7% of the permanent nodes. Then, a bootstrap process starts, bringing 50 new nodes from *out* to active or sleep states with equal probabilities at each time step[2]. After each time step, any non-permanent node that is either active or sleeping can switch from one state to the other with a given fixed probability [3] — this simulates the churn (note that

[2] A time step is counted after 50 messages.

[3] Hence, an exponential distribution can model joins and leaves.



(a) A network with 100 nodes

(b) A network with 10000 nodes

(c) Number of LRCs

(d) Dynamic 1000 node network

Figure 2: Dynamic performance

nodes reenter the network in a fresh state, i.e., without any LRCs originating or pointing to it). A node can never return to the *out* state. The main parameter to vary in this experiment is the rate at which nodes enter and leave the network or, in other words, the average lifetime of non-persistent nodes. The probability of switching state after a time step is varied from 0.00005 to 0.0025. In the Hop Level mechanism, churn is associated with two types of costs: the signaling cost of changing network topology and the cost of worse routing performance.

Figure 2(c) shows the number of LRCs created in the network under churn (signaling cost). From the perspective of active non-persistent nodes, the shorter the lifetime, the fewer LRCs such a node will create. This corresponds to the line deemed "LRCs p/ node". On the other hand, the load for the network and for the persistent nodes increases with churn. This is represented in the line deemed "LRCs p/ active node", which shows the total number of LRCs created in the network, divided by the average number of active nodes. We can see that even with very small lifetimes, the growth in the number of LRCs created per active node is moderate. Churn also degrades routing performance. This is illustrated in Figure 2(d) for a non-persistent node's lifetime of 10% of

experiment up time. The pattern depicted in this graphic is similar for other average lifetimes. Some time after the number of nodes stabilizes, the number of LRCs per (new entering) node starts to decay until it stabilizes to a value that depends on the churn rate.

6 Conclusions

In this paper we presented the Hop Level mechanism that manages Long Range Contacts (LRCs) in overlay networks. Experimental results showed that performance of Hop Level is nearly optimal and independent of node distribution in space. Furthermore, Hop Level resists churn very well without compromising performance in fresh networks. For these reasons, we believe that the Hop Level mechanism is applicable to a broad class of overlay networks, including multidimensional range queries in Distributed Storage Systems.

References

[1] I. Abraham, D. Malkhi, and O. Dobzinski. Land: stretch $(1 + \epsilon)$ locality-aware networks for dhts. In *fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 550–559. Society for Industrial and Applied Mathematics, 2004.

[2] K. Almeroth and M. Ammar. Collecting and modeling the join/leave behavior of multicast group members in the mbone. In *High Performance Distributed Computing (HPDC '96)*, pages 209–216, Syracuse, NY, USA, august 1996.

[3] F. Araújo and L. Rodrigues. Geopeer: A location-aware peer-to-peer system. In *The 3rd IEEE International Conference on Network Computing and Applications (NCA '04)*, pages 39–46, Cambridge, MA, USA, august 2004.

[4] J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load balancing and locality in range-queriable data structures. In *Twenty-Third Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2004)*, St. Johns, Newfoundland, Canada, July 2004.

[5] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353–366, 2004.

[6] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA., March 2003.

[7] F. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table, 2003.

[8] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43. ACM Press, 2004.

[9] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[10] J. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with Delaunay triangulation overlays. Technical Report CS-2001-26, University of Virginia, Department of Computer Science, 5 2001.

[11] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *4th Usenix Symposium on Internet Technologies and Systems*, 2003. http://www.usenix.org/events/usits03/.

[12] R. Melamed and I. Keidar. Araneola: A scalable multicast system for dynamic environments. In *The 3rd IEEE International Conference on Network Computing and Applications (NCA '04)*, pages 5–14, Cambridge, MA, USA, august 2004.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[15] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, january 2002.

[16] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *ACM SIGCOMM*, San Diego, August 2001.

[17] Z. Xu and Z. Zhang. Building low-maintenance expressways for p2p systems. Technical Report HPL-2002-41, HP, 2002.