# Highly Adaptive Lookup Systems for P2P Computing[1]

EWA KUSMIEREK, DAVID H.C. DU and JAMES C. BEYER
Digital Technology Center and Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
USA
{

*Abstract:* - The performance of file sharing peer-to-peer systems depends to a large degree on the speed of lookup operation. A number of proposed solutions rely on distributed hashing techniques. Traditionally nodes are assigned fixed length identifiers which does not allow the hash table to expand or shrink with the increase or decrease in the node count. With the fixed length identifiers, the performance deteriorates when the number of nodes reaches a high value. On the other hand, the overhead of maintaining per-node state (i.e., information of all adjacent nodes) can be unnecessarily large if the number of nodes is small. We propose a way to combine the *distributed* and *dynamic* nature of the system in a way that allows large and unpredictable changes in both the number and the distribution of nodes while providing scalability and good performance. Our approach is based on dynamic distributed hashing techniques; node identifier length varies with the number of nodes in the system. Hence, the system can adapt to the changing conditions and maintain good performance. We also describe the operations of the proposed adaptive system and verify its performance through simulations.

*Keywords:* - Peer-to-Peer, Dynamic hash table, Distributed hash table

## 1 Introduction

Peer-to-Peer (P2P) networks have recently gained a lot of interest and are a target of numerous research. They are instrumental in deployment of a number of large-scale distributed applications such as file sharing, distributed computing and collaborative environments. The requirements for P2P networks include good performance, self-organization, good scalability and robustness. There are additional requirements depending on the focused application. One of the major criteria for end-system multicast is the performance penalty compared to network layer multicast [10]. The file-sharing overlay performance evaluation is based on the path length measured by the number of nodes traversed to reach a node possessing the requested file. Path length determines the speed of lookup operation performed to retrieve the file. The cost in both overlay types is related to the per-node state [19].

The speed of a lookup operation in file sharing overlays depends on the method for determining the location of a requested object and on the routing used to reach that location. In Gnutella [1] flooding is used to search for the object. Napster [2] uses a central server that stores location of the available objects. Content addressable overlays [16, 18, 17, 20, 6] determine objects location by applying a hash function to the object's key, e.g. file name. They model the overlay network as a Distributed Hash Table (DHT). A typical design maps both nodes and objects to a common address space. The association between a node and an object is based on the numerical relation between node's and object's hash identifiers (addresses).

DHT-based systems are designed to handle changes in the number of nodes by applying the idea of consistent hashing [9]. The dynamic aspect of the system refers, however, not only to the number of nodes, but also to the *size of the hash table*. Having a fixed length identifier assigned to each node, limits the number of nodes in the system because the table cannot expand. In Chord [18], for example, the number of nodes must be small enough to make the probability of two nodes hashing to the same address negligible.

Most systems assume that the nodes are distributed *evenly* in the hash table. This may not always be the case. An uneven node distribution limits the number of nodes in the system even more since it increases the probability of collision. In systems such as CAN, where collisions do not take place, an uneven node distribution degrades the performance of a lookup process. The length of a path leading through a densely populated region is longer. The problem has been recognized mostly due to the potential load imbalance that it can cause [15, 3]. The mechanisms developed to address uneven node distribution include the idea of virtual peers in Chord and assigning each peer several zones. Such a solution increases the per-node state considerably for all nodes in the overlay.

The focus of our study is to propose a way to combine the *distributed* and *dynamic* nature of the system in a way that allows large and unpredictable changes in both the *number* and the *distribution* of nodes.We apply dynamic hash-

[1]This work is partially supported by NSF Grant EIA-0224424

ing techniques that allow the length of the identifier to vary. As the number of nodes increases, their identifiers become longer allowing the hash table to expand. The contraction takes place when the number of nodes decreases. The system benefits from having such adaptivity not only by being able to accommodate a wide range of the number of nodes without collisions but also by experiencing much slower performance degradation as the number of nodes increases.

We address a number of issues that arise when such a solution is applied in a distributed system. These include deciding when the identifier length should be changed, what the range of the change should be and who makes the decision. We describe modifications that have to be made to the per-node state, as well as to the routing process. Nodes have to be able to route message not only without knowing the total number of nodes in the system, but also without knowing the exact length of nodes' identifiers. We evaluate how the additional dynamic characteristics of the system affects its performance. We address also the issue of uneven node distribution by allowing the length of the identifiers to vary throughout the table.

The remainder of the paper is organized as follows. We give a short summary of two DHT-based systems and dynamic hashing in Section 2. We describe how dynamic hashing can be applied to CAN in Section 3. Section 4 contains performance evaluation. In Sections 5 and 6 we present some simulation results and offer some conclusions.

# 2 Related Work

Although we are addressing the general dynamic nature of P2P overlay networks, we briefly summarize CAN in this section and use this system to illustrate our concept. We also provide some background information on dynamic hashing.

## 2.1 Distributed Lookup Systems

The address space in CAN is represented by a virtual $d$-dimensional Cartesian space. A node is mapped randomly onto a point and assigned a zone in the vicinity. An object's address is obtained by applying a hash function to the object's key. The hash function yields $d$ coordinates in the virtual space. A node whose zone contains object's address stores the object.

Each node is aware only of the nodes whose zones are adjacent to its own. Thus, the size of the routing table is bounded by $\mathcal{O}(d)$. Routing is performed along a straight line connecting a node and the destination. The number of hops traversed to reach the destination is bounded by $\mathcal{O}(d\sqrt[d]{n})$, where $n$ is the number of nodes.

The virtual space is always partitioned into a number of zones equal to the number of nodes. When a new node joins the system, it "collides" with one of the existing peer. Thus, the number of zones has to be increased by splitting the "collision" zone in half. When a node leaves the system, the empty bucket is absorbed by one of the neighbors. Clearly, there is no upper bound on the number of nodes. However, as the number of nodes increases and their zone volumes become smaller, the performance deteriorates. Traveling a short distance in the virtual space may require traversing a large number of nodes.

The path length depends not only on the number of nodes but also on the number of dimensions, which determines the degree of connectivity among nodes. It is recommended to use a high number of dimensions if the number of nodes is also high. However, it is impossible to choose an optimal dimensionality if the number of nodes varies over time.

The possibility of uneven node distribution is addressed by adding a new node to the largest zone in the neighborhood of the randomly selected point. If we imagine the height of a zone in a 2-d system to be inversely proportional to its volume, we can say that the request rolls toward the low-elevation large-volume zone. However, the request can easily get trapped in a local minimum. Hence, this solution can smooth the node distribution only locally.

## 2.2 Distributed Dynamic Hashing

A number of techniques have been proposed for implementing dynamic hashing [12, 5]. We briefly summarize two approaches: directory-based [7] and directory-less [13]. We assume that each node (bucket) in a hash table can hold one object.

In Extendible Hashing [7] a distinction is made between the entries in a table (directory) and the buckets. The scheme uses pointers to provide a level of indirection that allows fewer buckets than directory entries to exist. If $d$ bits are used for addressing, a bucket with more than one pointer can be addressed with a number of bits smaller than $d$. When a collision occurs in such a bucket, a new bucket is created and the pointers are "split" between these two buckets. A collision in a bucket with one pointer is handled by doubling the size of the directory, i.e., by adding one more bit to the directory address space.

In Linear Hashing [13] the entries in the table are the buckets. The table size is increased by splitting one entry at a time when a collision occurs. Splitting an entry is realized by appending a new entry to the table. The new entry has the same address as the split one with a '1' prepended to it. The identifier of an old entry is also extended by prepending 0 to it. The entries are split in a round-robin fashion. Hence, the entry that is split is not necessarily the one involved in the collision.

Two features make the directory scheme more suitable for our purpose. It allows entries with various lengths of the identifiers to co-exist in the system, and the collision are resolved by splitting the bucket involved in the collision with-

out a need for a temporary solution to accommodate the new item.

Distributed versions of both dynamic hashing schemes have been proposed in [14, 4, 8]. Since the centralized structure does not exist in the distributed system, the table is split among multiple nodes. A two-level structure is used to locate the objects. An object is hashed into an entry containing a pointer to the actual bucket. In EH* [8] the directory is collapsed into several cache tables to eliminate multiple pointers pointing to the same entry. Each server is responsible for storing one cache entry and each server maintains a replica of the directory containing pointers to buckets. Since it is expensive to keep all replicas consistent, the information can be inaccurate and addressing errors can occur. Another approach is to have each node maintain only a small part of the directory and keep it up-to-date. Such approach has been adopted in some DHT-based P2P systems.

# 3 Dynamic Dimensionality

In the rest of the paper, for the convenience of presentation we focus on applying dynamic distributed hashing to CAN to address the dynamic aspect of the system. We examine the "addressing" scheme in terms of the length of the node identifiers and point out how it can be enhanced. Then we discuss how this enhancement affects the operations of the system and its performance.

## 3.1 Dynamic Hashing in CAN

A node in CAN is assigned to a single zone in a $d$-dimensional space. Hence, its identifier consists of $d$ coordinates. More precisely, there is a range for each coordinate specifying a width of the zone along a given dimension. This range is represented by a binary string. A string consisting of a single bit 0 corresponds to the maximum width along a given dimension. When a zone is split between two nodes, the node that gets the lower half appends a 0 to the binary string for a dimension along which the split takes place, the other node appends a 1.

The current scheme allows adding nodes indefinitely. The number of entries in the hash table is unlimited, but the volume of the corresponding space is fixed. As the number of zones increases and their volumes become smaller, the performance worsens. Traveling a short distance in the virtual space may require a large number of hops. In order to control the path length and to improve the lookup time, we extended dynamic hashing onto the *number of dimensions*. Our approach permits dynamic changes of the volume by varying the dimensionality of the system.

When the partition of space becomes too fine, we increase the volume of the space by adding another dimension. The identifier of each node is extended by adding another binary string. An increase in dimensionality results in the higher node connectivity by increasing the number of neighbors. Recall that routing table size is bounded by $\mathcal{O}(d)$. Consequently, the path increases slower with the increase in the number of nodes.

## 3.2 Node Density

We now proceed to answer the following two questions: when a new dimension should be added and who makes that decision. The number of nodes in the system is a good indicator on how fine the partitioning of space is. Therefore, a threshold on the number of nodes is used to control the dimensionality. Let $n_0$ denote the threshold. Then, when the number of nodes reaches $n_0^d$, the dimensionality is increased from $d$ to $d+1$.

Since the system is fully distributed a decision on dimensionality adjustment has to be made in a distributed manner. Each node computes an estimate of the total number of nodes by calculating first a *local node density*, i.e., the number of nodes per unit of volume. The calculation is based on the knowledge of the system a node already has: knowledge of its neighborhood. Then the total number of nodes is equal to the local node density multiplied by the volume of the whole space.

Note that if the distribution of nodes is uniform, all nodes will have similar estimates and the system will quickly converge to higher dimensionality. Otherwise, an increase in the number of dimensions will be only local, affecting a small volume of space. Therefore, the number of dimensions may vary throughout the space. Dimensionality may be higher in the more dense areas. This property is very useful in improving the performance of the system with uneven node distribution. It allows us to increase the connectivity only locally without imposing an unnecessary overhead in the whole space.

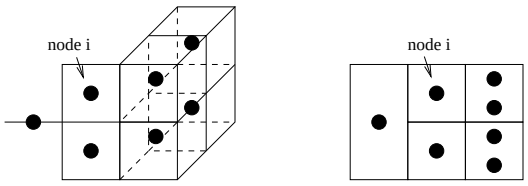



Figure 1: Calculating density in multiple dimensionality case

Since nodes of various dimensionalities may exist in the system at the same time, the first step in assessing node density is the equalization of the number of dimensions throughout the neighborhood. If node $i$ has $d_i$ dimensions, it projects any zone in its neighborhood with higher dimensionality into $d_i$ dimensions, and extends any zone with lower number of dimensions into $d_i$-dimensional space. A zone is extended by assuming that it covers the whole range in each of the additional dimensions. Figure 1 presents an example.

An even distribution of nodes throughout the space is optimal for the path length. Hence, we want to maintain this property by keeping the number of nodes along each dimension roughly the same. When a new dimension is added, the subsequent zone splits are performed only along that new dimension in order to decrease the difference between number of nodes along "old" and "new" dimensions. The threshold $n_0$ can be interpreted as a threshold on the number of nodes along a single dimension.

### 3.3 Routing

Multiple dimensionalities of the space affect a number of system operations. The detailed description of this effect can be found in [11]. Here we describe only the changes to routing procedure. Routing follows a straight line through the Euclidean space and the next hop is selected based on distance between a node and the destination. A node chooses the neighbor which is the closest to the destination. as the next hop. Having variable dimensionality throughout the space poses the question how the distance should be calculated, i.e., what dimensionality should be assumed.

One approach to the problem is to use the maximum number of dimensions $D_{max}$ by extending the zones with dimensionality lower than $D_{max}$. We show that we can reduce the computational cost of selecting the next hop by using *local* dimensionality, i.e., the dimensionality of a neighbor.

Assume that the neighbor node's dimensionality $d$ is lower than $D_{max}$. We use $d$ coordinates of point $c$, $(c_1, c_2, ..., c_d)$, to calculate the distance and compare it with the distance based on $D_{max}$ coordinates. Let $a$ be the point in the node's zone which is closest to $c$ in $d$-dimensional space. The distance between $a$ and $c$ is then: $\sqrt{(a_1 - c_1)^2 + ... + (a_d - c_d)^2}$. When dimensionality is extended to $D_{max}$, we use all coordinates of point $c$: $(c_1, c_2, ..., c_d, ..., c_{D_{max}})$. Now the point $(a_1, a_2, ..., a_d, c_{d+1}, ..., c_{D_{max}})$ belongs to the zone and it is the closest point to $c$ in that zone. Hence, the distance is the same in both dimensionalities.

## 4 Performance Comparison

In order to evaluate the performance of the dynamic system, we compare it against the performance of the static system with the same number of nodes. We assume that a central mechanism is used to keep track of the number of nodes and control dimensionality. Hence, the results presented here are based on the asymptotic behavior of the dynamic system.

The path length is bounded in the following way. We first find a bound on the path length that can be traveled in each of the dimensions and then add these component. The path length bound in the static system under the assumption of even distribution is given by:

$$P_s(n,d) = \sqrt{\sum_{i=1}^{d} (\sqrt[d]{n})^2} = \sqrt{d}\sqrt[d]{n} \qquad (1)$$

where $n$ is the number of nodes and $d$ is the number of dimensions.

The path length bound for a dynamic system is given by:

$$P_d(n,n_0) = \sqrt{(d(n)-1)n_0^2 + \left(\frac{n}{n_0^{d(n)-1}}\right)^2} \qquad (2)$$

where $n_0$ is the threshold on the number of nodes and $d(n)$ is the corresponding number of dimensions:

$$d(n) = \lceil \log_{n_0} n \rceil \qquad (3)$$

Figure 2 presents the comparison of the path length bounds for static and dynamic system with threshold $n_0 = 10$. The number of dimensions $d = 3$ in the static system is assumed optimal for a 1000 nodes. The dynamic system uses smaller number of dimensions for $n < 1000$. Therefore, path length is potentially longer than path length in the static system but the size of the routing table maintained by each node is smaller.

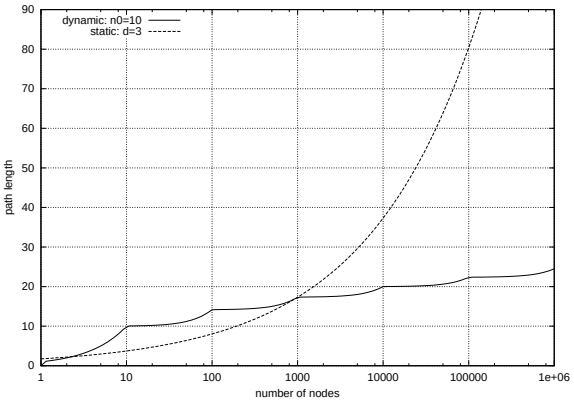
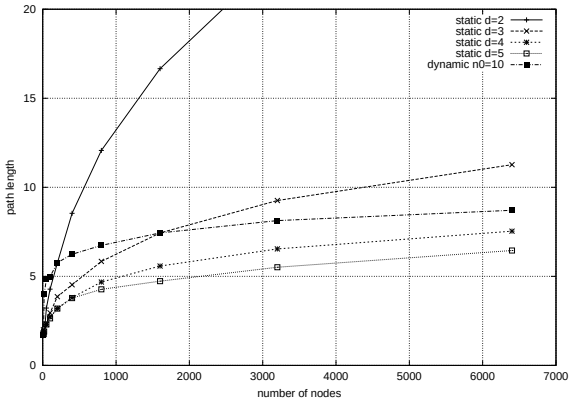



Figure 2: Performance comparison for $n_0 = 10$

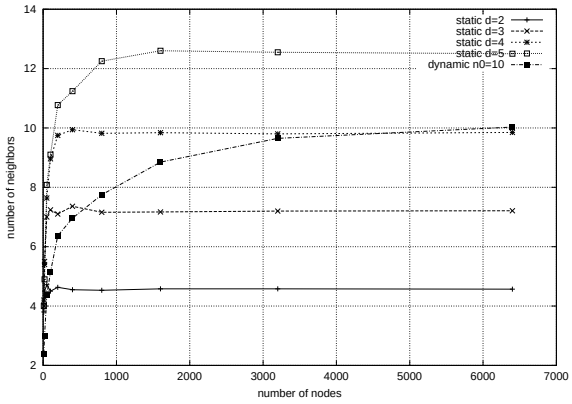
Since the selected threshold $n_0$ is equal to $\sqrt[d]{1000}$, both system have the same path length bound for $n = 1000$. Beyond that point the path length bound for a dynamic system is smaller since the dimensionality keeps increasing. The table size in the dynamic system increases with the number of nodes but only linearly.

### 4.1 Routing Table Size

There are two main differences between the static and the dynamic system with respect to the routing table size. In the static system the routing table size does not depend on the number of nodes and is determined by dimensionality. In




(a) Average path length




(b) Average number of neighbors

Figure 3: Comparison between static and dynamic system

the dynamic system the number of neighbors changes with the number of nodes and is affected by the dimensionality of the node's neighbors. Consider a d-dimensional node, which has neighbors with higher dimensionality. The number of neighbors of d-dimensional node is higher than indicated by its dimensionality. On the other hand, a lower-dimensionality neighbor generally does not have the opposite effect on the number of neighbors.The dynamic system has nodes with various dimensionalities present in the system since the change of dimensionality occurs gradually. We conclude that the average routing size table is higher than indicated by the average dimensionality.

# 5 Simulation Results

We conducted a number of simulations to verify the performance of the dynamic system. We examine how the performance changes with the number of nodes and how it compares to the performance of the static system.

## 5.1 Path Length and Routing Table Size

In the first set of tests, each node in the dynamic system evaluates the density based on the volume of its own zone and uses $n_0 = 10$ as the threshold. Within the given range of the node count ($\leq 6400$), the dynamic system starts with 1 dimension and reaches dimensionality 5. Therefore, we have included static systems with dimensionality 2, 3, 4 and 5 in the test. 1-dimensional system performance deteriorates very quickly and thus was excluded. Figure 3(a) presents the comparison between the average path lengths obtained in both types of systems. The average path length is calculated over a set of paths obtained by selecting a random destination for each node in the system.

We observe that as the number of nodes increases, the path length in the dynamic system outperforms static systems with consecutive dimensionalities. First, the path length becomes shorter than in the 2-dimensional (2-d) system, then it overtakes the 3-d system. The difference between the path length in the dynamic system and in the 4-d system decreases past the intersection point with the 3-d path length and we expect the dynamic system to overtake the 4-d system for a higher number of nodes. When the number of nodes reaches 6400 the dynamic system has mostly nodes with dimensionality 4 but the average path length is longer than that in the 4-d system. This is the price we pay for the dynamic system's ability to adapt. When the threshold $n_0$ is equal to 10, we consider dimensionality 4 to be optimal for the number of nodes $10^3 \leq n < 10^4$. Hence, the performance of the dynamic system with respect to the path length is slightly below optimal.

Figure 3(b) presents the comparison of the average per-node state in the same settings. The average number of neighbors increases with the number of nodes for both systems. It increases also with dimensionality in the dynamic system. For the largest examined number of nodes (6400), it is close to the number of neighbors in the 4-d system, and the majority of nodes in the dynamic system have dimensionality 4.

# 6 Conclusions and Future Work

We have presented how CAN can benefit from the idea of dynamic dimensionality. By allowing the dimensionality to vary with the number of nodes and throughout the space, the system can adapt to dynamical changes in the set of nodes. The size of the routing table maintained by each node is increased compared to the system with static dimensionality

but the performance measured by the path length is considerably improved. The technique used to maintain dynamic dimensionality is derived from dynamic hashing. It adapts the length of identifiers of both data items and nodes, allowing to control the access time.

To support the claim of broad applicability of dynamic hashing to general distributed lookup systems, we briefly describe how Chord's performance can be improved using dynamic hashing. Assume that $m$-bit identifiers are used. Once the collision is encountered, the length of identifiers is doubles by appending another $m$ bits. In the geometric interpretation, we create another ring and attach it to the the existing node, say $i$, participating in the collision. This node belongs now to two rings. The routing is done using $m$ most significant bits of the identifier. If item's successor, node $i$, does not have the item, it passes the request onto the second ring for a similar routing procedure, this time using a second group of $m$ bits. Note that the secondary ring is 2-dimensional since node $i$ is potentially responsible for a set of data items. These data items are now partitioned among nodes on the secondary ring. The identifiers of nodes on the secondary ring have the first $m$ bits such that node $i$ is their successor. The second group of $m$ bits can have an arbitrary value. New nodes, whose successor is node $i$, are added to the secondary ring. The procedure of extending the identifier can be continued by adding more rings at different levels (dimensions). The details of maintaining the connection between rings as well as performance improvement that can be obtained are a subject of future work.

## Acknowledgment

The authors would like to thank Sylvia Ratnasamy for making the implementation of CAN available for simulations.

*References:*

[1] Gnutella http://gnutella.wego.com.

[2] Napster http://www.napster.com/.

[3] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, Mar. 2002.

[4] R. Devine. Design and implementation of DDH: A distributed dynamic hashing algorithm. In *4th International Conference on Foundation of Data Organizations and Algorithms*, 1993.

[5] R. Enbody and D. H.-C. Du. Dynamic hashing schemes. *ACM Computing Surveys*, 20(2):85–114, 1988.

[6] J. K. et al. OceanStore: An architecture for global-scale persistent storage. In *9th international Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[7] R. Fagin, J. Nievergelt, N. Pippenger, and H. Strong. Extendible hashing - a fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

[8] V. Hilford, F. Bastani, and B. Cukic. EH*-extendible hashing in a distributed environment. In *Proceedings of the 21st Computer Software and Applications Conference*, 1997.

[9] D. Karger, E. Lehman, T. Leighton, M. Levibe, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocol for relieving hot spots on the world wide web. In *29th Annual ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[10] D. Kostic and A. Vahdat. Latency versus cost optimizations in hierarchical overlay networks. Technical Report CS-2001-04, Duke University, November 2001.

[11] E. Kusmierek, D. Du, and J. Beyer. Highly adaptive look-up system for peer-to-peer computing. Technical Report 004-09, University of Minnesota, 2004.

[12] P.-A. Larson. Dynamic hashing. *BIT*, 18:184–201, 1978.

[13] W. Litwin. Linear hashing: A new tool for file and table addressing. In *VLDB*, pages 212–223, 1980.

[14] W. Litwin, M. Neimat, and D. Schneider. LH* - linear hashing distributed files. In *ACM SIGMOD*, 1993.

[15] A. Rao, K. Lakshminarayanan, S. Surana, and R. K. I. Stoica. Load balancing in structured p2p systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, Mar. 2002.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[17] 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, 2001.

[18] 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.

[19] J. Xu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. In *INFOCOM*, 2003.

[20] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, Apr. 2001.