The Impact of DHT Routing Geometry on Resilience and Proximity

K. Gummadi[*], R. Gummadi[†], S. Gribble[‡], S. Ratnasamy[§], S. Shenker[¶], I. Stoica[‖] [*]

ABSTRACT

The various proposed DHT routing algorithms embody several different underlying routing *geometries*. These geometries include hypercubes, rings, tree-like structures, and butterfly networks. In this paper we focus on how these basic geometric approaches affect the resilience and proximity properties of DHTs. One factor that distinguishes these geometries is the degree of *flexibility* they provide in the selection of neighbors and routes. Flexibility is an important factor in achieving good static resilience and effective proximity neighbor and route selection. Our basic finding is that, despite our initial preference for more complex geometries, the ring geometry allows the greatest flexibility, and hence achieves the best resilience and proximity performance.

Categories and Subject Descriptors

C.2 [**Computer Systems Organisation**]: Computer Communication Networks

General Terms

Algorithms, Performance

Keywords

DHT, Routing Geometry, Flexibility

[*]University of Washington. gummadi@cs.washington.edu
[†]USC, Los Angeles. gummadi@usc.edu
[‡]University of Washington. gribble@cs.washington.edu
[§]Intel Research, Berkeley. sylvia@intel-research.net
[¶]ICSI, Berkeley. shenker@icsi.berkeley.edu
[‖]UC Berkeley. istoica@cs.berkeley.edu

[*]This work is supported in part by NSF grants CCR-0121341, ITR-0085670, IIS-0205635, ANI-0196514, ANI-0207399, ITR-0205519, ITR-0081698, ITR-0121555, ANI-00225660, ANI-0133811 and Stoica's Sloan Foundation Fellowship.

SIGCOMM'03, August 25–29, 2003, Karlsruhe, Germany.

1. INTRODUCTION

The unexpected and unprecedented explosion of peer-to-peer file-sharing, ignited by Napster and refueled by a succession of less legally vulnerable successors (*e.g.*, Gnutella, Kazaa), inspired the development of distributed hash tables (DHTs). DHTs offer a promising combination of extreme scalability (scaling typically as $\log n$) and useful semantics (supporting a hash-table-like `lookup` interface), and have been proposed as a substrate for many large-scale distributed applications (see, for example, [5, 12, 22]).

Our focus here is not on the uses of DHTs but on the underlying DHT routing algorithms. A wide variety of DHT routing algorithms have been proposed, and the list grows longer with each passing conference. However, the DHT routing literature is still very much in its infancy; as a result, most DHT routing papers describe new algorithms and very few provide more general insight *across* algorithms or usefully compare *between* algorithms.

In this paper we attempt to take a very small step in this direction by looking at the basic *geometry* underlying DHT routing algorithms and how it impacts their performance in two important areas: static resilience and proximity routing. While the current collection of DHT routing algorithms differs in many respects, perhaps the most fundamental distinction between them lies in their different routing geometries. For instance, CAN [19] (when the dimension is taken to be $\log n$) routes along a hypercube, Chord [25] routes along a ring, Viceroy [14] uses a butterfly network, and PRR's algorithm [16] uses a tree-like structure. These geometries have differing degrees of *flexibility* in choosing neighbors and next-hop paths. The importance of geometry, and the resulting flexibility, isn't clear when looking only at the typical metrics of concern – state (the number of neighbors) and efficiency (the average path length) – because many of these algorithms can achieve the same state/efficiency trade offs. However, geometry (and flexibility) becomes more relevant when looking at other performance issues.

One of the crucial questions facing DHTs is whether they can operate in an extremely transient environment where a significant fraction of the nodes are down at any one time. There are many facets to this issue, most notably the speed and overhead of recovery algorithms, but one relatively (but not completely; see [13, 22]) unexplored area is static resilience, which is how well the DHT can route even before the recovery algorithms have had a chance to work their magic. We discuss this issue in Section 3.

Another crucial question is how well DHTs can adapt to the underlying Internet topology. Much research has been devoted to incorporating *proximity* into DHT routing protocols, and there are two areas of concern. The first is the total latency of the DHT routing path, which should be no more than a small multiple of the underlying Internet latency. We discuss path latency in Section 4. The second is whether such paths converge; the various tasks of providing efficient caching, building parsimonious multicast trees, and finding the closest server out of many are all made easier if DHT routing paths have a *local convergence* property that we explore in Section 5.

Our paper examines the extent to which geometry impacts the performance in these two areas, and thus we begin our paper by discussing geometry in Section 2. However, before embarking on this discussion, which we hope provides some insight, we readily confess that our paper is a very initial stab at the problem and is undoubtedly incomplete. Our various sins include (1) only picking a few, not all, of the currently proposed DHT routing algorithms, (2) not considering factors, such as symmetry, that may affect the state management overhead and (3) only focusing on two performance issues (resilience and proximity), and not considering interactions between these various properties. Rectifying these omissions is the subject of future work.

2. GEOMETRIES AND ALGORITHMS

To provide context for the technical material to follow, we first discuss the philosophy that motivates this work. As noted before, there are a myriad of DHT designs in the literature, all extensively analyzed and energetically promoted. However, we envision that there will be only one or a few very large-scale DHT infrastructures. Thus, out of the cornucopia of routing choices we must select one (or a few) on which to build future systems. The question is: how do we make that choice?

One approach is to view these systems as complete coherent proposals and have a *bake-off* among them, comparing their performance in real-world situations. This *black-box* approach, which is taken in [1, 24] and elsewhere, compares the designs as turn-key systems. While this provides performance comparisons for a fixed set of designs and a given environment, it doesn't provide much guidance for designing new DHTs. A very different approach is to view each proposal as a set of somewhat independent design choices and to evaluate the wisdom of each design choice in the hope that this insight will eventually lead to a superior hybrid design. This paper adopts the latter approach. Thus, we do not intend our observations to be rankings of the current DHT proposals, but rather to be hints about how one might better design a future DHT routing algorithm.

2.1 Geometries

One can roughly divide DHT design issues into two categories:

Routing-level This category is confined to two well-defined issues: neighbor selection – how one picks the set of neighbors for a node – and route selection – how one chooses the next hop when routing a message. These choices determine what we call the *routing behavior* of a DHT.

System-level This category contains everything else. These are higher-level design decisions that apply across all routing-level choices. Examples of system-level issues are caching and replication and whether the actual delivery of messages is done iteratively or recursively.

Routing-level and system-level choices are not always completely independent, in that some routing-level design choices affect the feasibility or performance of a system-level design choice, but often they are largely orthogonal and should be treated as such.

Another useful distinction is between the routing *algorithm* and the routing *geometry*.

Algorithm This refers to the precise details of how neighbors and next-hops are chosen. Any change in these details is a change in the routing algorithm.

Geometry This is not a precise term, and we present no formal definition, but often the way in which neighbors and routes are chosen has a compelling geometric interpretation. Conversely, the choice of a routing geometry constrains the way in which neighbor and routing choices are made. Small changes to the routing algorithm do not change the underlying routing geometry.

Our work here focuses on the impact of routing geometry, rather than on the particular algorithmic details. As noted earlier, most of the current DHT routing proposals have very clear geometric interpretations. The question is which properties of these various DHT routing algorithms derive from their basic choice of geometry, and which can be altered by small algorithmic changes. Understanding the constraints and possibilities inherent in an underlying routing geometry will help guide us when designing future DHT routing algorithms.

Note that all of these geometries are capable of providing $O(\log n)$ path lengths with $O(\log n)$ neighbors (and Viceroy can improve that to $O(1)$ neighbors). Thus, we don't focus on how geometry effects space and efficiency. We instead note that these geometries place very different constraints on route and neighbor selection. The resulting varying degrees of *flexibility* each geometry provides has significant impact on the resilience and proximity properties of the system. We now discuss flexibility in somewhat greater depth.

2.2 Flexibility

Flexibility is nothing more than the algorithmic freedom left after the basic routing geometry has been chosen. This freedom is exercised in the selection of neighbors and routes. We discuss these issues in turn:

2.2.1 Neighbor Selection

DHTs have a routing table comprised of neighbors. In the original set of DHT proposals, some algorithms made this choice of neighbors purely deterministic (*i.e.,* given the set of identifiers in the system, the neighbor tables were completely determined), and others allowed some freedom to choose neighbors based on other criteria in addition to the identifiers; most notably, proximity (*i.e.,* latencies) have been used to select neighbors. However, the question is not whether the initial proposal included this feature, but whether the basic routing geometry precludes it. That is, if the routing geometry precludes choosing neighbors based on proximity, then that it is a true loss of flexibility; if the routing geometry allows such a choice, then the omission is merely a small algorithmic detail that can be changed.

Several DHT proposals make use of what we will call *sequential* neighbors. These are neighbors to which one can route and be sure of making progress towards *all* destinations. The leafset in Pastry and the successors in Chord are examples of sequential neighbors. Some geometries naturally support such sequential neighbors, and some don't. Since such neighbors play a crucial role in recovery algorithms, several DHT proposals whose basic geometries don't naturally support sequential neighbors have augmented their design to include them. The result is a somewhat *hybrid* geometry.

2.2.2 Route Selection

Given a set of neighbors, and a destination, the routing algorithm determines the choice of the next hop. However, flexibility is relevant here, for two reasons. First, when the determined next-hop is down, flexibility describes how many other options are there for the next-hop. If there are none, or only a few, then the routing algorithm is likely to fare poorly under high failure rates.

Second, analogous to choosing neighbors based on proximity, one might want to choose next hops based on proximity. To some extent, this reflects the same degree of freedom alluded to above (for picking other options under failure) but it arises in a different context.

2.3 Algorithms

We now discuss some basic routing geometries. We do so by reviewing several DHTs and describing how their routing algorithm can be interpreted geometrically. We make special note of the flexibility in neighbor and route selection provided by these different geometries. However, it is important to note that we consider such flexibility only when it can be exercised without significantly altering the state-efficiency tradeoff for that geometry. For instance, for Chord we could pick $O(n)$ neighbors and choose routes with $O(n)$ path lengths with a spectacular degree of flexibility but that would not represent a desirable design.

In the discussion that follows, we assume systems with n nodes and $\log n$ bit node identifiers. In practice, these algorithms typically use $\log N$ bit identifiers where $N \gg n$, so the identifier space is not wholly populated by nodes, but for clarity and conciseness we assume wholly populated identifier spaces in the following discussion. Likewise, our description treats identifiers as binary strings but in practice they will be to some base b. Neither of these assumptions will affect the correctness of our comments, but both will help facilitate the presentation.[1] Finally, we abuse terminology and frequently use the term node to refer to the node's identifier.

We consider the following DHTs: PRR [16], CAN [19, 18], Chord [25], Viceroy [14], Pastry [22] and Kademlia [15]. For reasons of space and time, there are many algorithms that we do not consider, among them being Tapestry [28] (though our discussion of PRR should mostly apply to Tapestry as well), a recent de Bruijn inspired algorithm [6], and a randomized algorithm due to Kleinberg [11].

2.3.1 Tree

The tree's hierarchical organization makes it a likely candidate for efficient routing, and in fact it is the geometry underlying PRR's algorithm, which is perhaps the first DHT routing algorithm. The basic routing algorithms in Tapestry and Pastry are both rather similar in spirit to this original algorithm although Pastry (as we describe later) also uses a ring-like geometry in addition to the tree. In a tree geometry, node identifiers constitute the leaf nodes in a binary tree of depth $\log n$; the "distance" between any two nodes is the height of their smallest common subtree. Each node holds $\log n$ neighbors, where the i^{th} neighbor is at distance i from the given node and routing works by greedy routing towards the destination. In other words, a node has neighbor nodes that match each prefix of its own identifier but differ in the next bit. Routing is achieved by successively "correcting" the highest order bit on which the forwarding node differs from the destination, effectively increasing the length of the longest prefix match by one at each hop.

We make the following observation on the flexibility a node has in choosing its neighbors: any given node has 2^{i-1} options in choosing a neighbor at distance i from itself, corresponding to the subtree of nodes that share the first $\log n - i$ bits with the given node but differ on the $\log n - i + 1$ bit. Thus, these "unconstrained" lower order bits give a great deal of freedom in choosing neighbors, with the amount of freedom increasing exponentially with increasing distance i. This yields a total of approximately $n^{(\log n)/2} (= \Pi_{i=1}^{\log n} 2^i)$ *possible* routing tables per node. Of course, a node will choose exactly one of these possible routing tables.

We now consider routing flexibility. Given a particular choice of routing table, how much flexibility does a node have in selecting the next hop to a given destination? We observe that in a tree, a node has only one neighbor that reduces the distance to the destination; *i.e.*, only one neighbor can increase the length of the longest prefix match. Thus, in contrast to the generous flexibility in choosing neighbors, a tree offers *no* flexibility in route selection.

2.3.2 Hypercube

The routing used in CAN resembles a hypercube geometry. CAN uses a d-torus that is partitioned amongst nodes such that every node "owns" a distinct zone within the space. As explained in [18], a CAN node's identifier is a binary string representing its position in the space. When $d = \log n$ dimensions the neighbor sets in CAN are exactly those of a $\log n$-dimensional hypercube.[2] Each node has $\log n$ neighbors; neighbor i differs from the given node on *only* the i^{th} bit. The "distance" between two nodes is the number of bits on which their identifiers differ and routing works by greedy forwarding to reduce this distance. Thus routing is effectively achieved by "correcting" bits on which forwarding node differs from the destination. The key difference between routing on the hypercube and the tree is that the hypercube allows bits to be corrected in any order while on the tree bits have to be corrected in strictly left-to-right order. The reason the hypercube can use out-of-order bit

[1] Our simulation results, presented later in the paper, do not assume dense identifier space. However, they all use base 2 to allow us to work with reasonable path lengths (typically 6-8 hops); clearly however, our results would be less relevant for DHTs with very short paths of 1-2 hops [7].

[2] This hypercube-like interpretation of CAN is also extendable to $d < \log n$; however, for the purposes of this paper we restrict ourselves to the case where CAN has $\log n$ dimensions and routes in $\log n$ hops so that it is more directly comparable to the other DHTs.

fixing is because, unlike on the tree, a node's neighbor only differs from itself on a single bit and hence previous corrections of lower order bits are maintained as higher order bits are corrected. Thus, the hypercube offers greater flexibility in route selection. Specifically, in routing from a source to destination that are at a distance of $\log n$, the first node has $\log n$ next hop choices, the second node has $\log n - 1$ choices, and so on yielding a total of approximately $(\log n)!$ routes between two nodes. Note that each of these paths is of the same length – they only differ on the *order* in which destination bits were fixed. The hypercube however pays a price for this flexibility in route selection. Because a node's neighbors differ from itself on exactly one bit, it has only one possible choice for each of its neighbors. Thus a node has no flexibility in selecting its neighbors. This is the opposite of what occurs with the tree, which has much neighbor selection flexibility but no route selection flexibility.

2.3.3 Butterfly

The Viceroy algorithm emulates the operation of a traditional butterfly network but adapts this structure to be self-organizing and robust in the face of node arrivals and departures. Viceroy improves on the state-efficiency tradeoff of previously proposed DHTs by routing in $O(\log n)$ hops with constant state at each node. The details of Viceroy are fairly involved, so we only provide a very sketchy overview. In a traditional butterfly, the nodes are organized in a series of $\log n$ "stages" where all the nodes at stage i are capable of (essentially) correcting the i^{th} bit in the identifier. To ensure correctness in the face of node dynamics, Viceroy imposes a global ordering on all the nodes in the system and requires each node to hold, as neighbors, its immediate successor and predecessor in this ordering. A node also holds, as neighbors, its immediate successor and predecessor from among the nodes in its own stage. Viceroy routing consists of three phases: the first uses $O(\log n)$ hops to move up to the first stage, the second uses another $O(\log n)$ hops to traverse down the stages until it reaches the vicinity of the destination at which point routing enters its third phase and uses the successor/predecessor neighbors to reach the destination in a $O(\log n)$ additional hops. We note that this final routing phase, which uses $O(\log n)$ hops, does not permit flexibility in either route or neighbor selection. Thus, while the butterfly achieves greater efficiency than the other DHTs we consider, it results in far less flexibility. We stress that this is not a "flaw" in the particular Viceroy design; rather, we conjecture that this limitation is fundamental to constant state algorithms.[3]

2.3.4 Ring

In a Ring geometry, nodes lie on a one-dimensional cyclic identifier space on which the "distance" from an identifier A to B is calculated as the clockwise numeric distance from A to B on the circle.[4] Chord embodies such a ring geometry. In Chord, a node with identifier (say) a maintains $\log n$ neighbors (called fingers) where the i^{th} neighbor is the node closest to $a + 2^i$ on the circle. Hence, a node can route to an arbitrary destination in $\log n$ hops because each hop cuts the distance to the destination by half. Although the original Chord proposal defines a specific set of neighbors for a given node identifier, this rigidity in neighbor selection is in no way fundamental to a ring geometry.[5] Specifically, routing on a ring can be achieved in $O(\log n)$ hops even if node a were to pick its i^{th} neighbor as any node in the range $[(a + 2^i), (a + 2^{i+1})]$ rather than the exact node closest to $a + 2^i$ on the circle as originally defined by Chord. This implies that in terms of the flexibility of neighbor selection, a ring geometry (like the tree) has 2^i possible options in picking its i^{th} neighbor for a total of approximately $n^{(\log n)/2}$ possible routing tables for each node. Having selected one of its possible routing tables, we examine the flexibility in route selection now available to the node in terms of which of its neighbors make progress towards a destination. For two nodes that are initially $O(n)$ distance apart, the first node has approximately $\log n$ of its neighbors that make progress towards the destination. After the first hop, the next node will have approximately $(\log n) - 1$ possible next hops and so on to yield a total of approximately $(\log n)!$ possible routes for a typical path. Note that all these paths respect the efficiency bound of $O(\log n)$ hops. This is because routing from the source to destination uses $\log n$ hops that span exponentially different distances – while greedy routing takes these hops in the decreasing order of their spans, any path that takes each of the different spans just once will reach the destination in $\log n$ hops irrespective of the order in which the spans are taken. Later, in section 4.1, we will define a rule that allows Chord to take each of these different spans just once without imposing any order.

While our discussion above is limited to paths that are bounded by $\log n$ hops, Chord also allows paths that are much longer than $\log n$. This is accomplished by taking multiple hops of smaller spans instead of a single hop of large span. For example, one could take two successive hops using $i - 1^{th}$ neighbors of span 2^{i-1} each instead of a single hop using i^{th} neighbor of span 2^i.

2.3.5 XOR

Kademlia [15] defines a novel routing metric – the distance between two nodes is the numeric value of the exclusive OR (XOR) of their identifiers. For lack of a more intuitive name, we use the term XOR geometry to refer to the geometry interpretation yielded by this XOR metric. A Kademlia node picks $\log n$ neighbors, where the i^{th} neighbor is any node within an XOR distance of $[2^i, 2^{i+1}]$ from itself. Examination of the above definition of neighbors reveals that Kademlia's routing table permits exactly the same routing entries as for tree geometries such as PRR. Moreover, by routing greedily on the XOR metric, Kademlia chooses exactly the same routes as PRR when the routing table is fully populated (*i.e.,* no failures); successive hops "fix" bits from left to right. Under failures, however, Kademlia behaves differently since, unlike on the tree, even if a node cannot fix the highest differing bit it can still make progress in the XOR distance to the destination, effectively fixing a lower order

[3]Moreover, there are likely ways to restore some flexibility by keeping more state, but making that state less "critical" to the operation of the algorithm (and hence not imposing the same recovery requirement as the critical state). This is the subject of future work, and we do not pursue it here.

[4]Different ring-based DHTs measure distance on the ring in slightly different ways; Chord uses the clockwise distance while Viceroy uses the minimum of the clockwise and anticlockwise distance. This difference only changes the constants in our flexibility bounds.

[5]This is a fairly well recognized fact in the DHT community and by the developers of Chord, and we do not claim to be the first to make this observation.

property	tree	hypercube	ring	butterfly	xor	hybrid
Neighbor Selection	$n^{\log n/2}$	1	$n^{\log n/2}$	1	$n^{\log n/2}$	$n^{\log n/2}$
Route Selection (optimal paths)	1	$c_1(\log n)$	$c_1(\log n)$	1	1	1
Route Selection (non-optimal paths)	-	-	$2c_2(\log n)$	-	$c_2(\log n)$	$c_2(\log n)$
Natural support for sequential neighbors?	no	no	yes	no	no	Default routing: no Fallback routing: yes

Table 1: The neighbor and route selection flexibility at any node in various routing geometries. c_1 and c_2 are small constants.

bit. Thus, multiple paths exist between a source and destination, but these paths are *not* of equal lengths. This stems from the fact that the distance of a node's neighbors with respect to the node have little bearing on their distance to the destination. Intuitively, even though Kademlia offers the flexibility of fixing lower order bits before higher ones, the lower order fixed bit need not be preserved by later routing hops that fix higher order bits.

2.3.6 Hybrid

So far we have only presented *pure* geometric interpretations. However, some routing algorithms employ dual modes, where each mode inspires a different geometric interpretation; we call these *hybrid* geometries. We use Pastry as our canonical example of a hybrid geometry because it combines the use of a tree geometry with that of a ring geometry. Node identifiers are regarded as both the leaves of a binary tree and as points on a one-dimensional circle. In Pastry, the "distance" between a given pair of nodes is thus computed in two different ways – the first is the tree distance between them, the second is the cyclic numeric distance between them. By default, Pastry uses the tree distance as its metric for routing and only falls back to using the ring geometry when the tree-based routing fails. Thus, its freedom of neighbor selection is the same as Tree geometries. The route selection flexibility is more subtle. The hybrid geometry allows one to take hops that do not make progress on the tree but do make progress on the ring; these paths however do not necessarily retain the $\log n$ bound on the number of hops.[6]

While we use Pastry as our canonical example, we point out that other DHTs can, and do, make use of a similar hybrid geometry, which we now discuss.

As mentioned earlier, sequential neighbors are those that make progress towards *all* destination identifiers. This requires a single global ordering on the distances between nodes, and thus the ring is the only geometry that naturally supports sequential neighbors. However, several of the designs can be augmented to include sequential neighbors, especially for recovery, essentially by defining a separate ordering (aside from the ordering used for normal routing). In particular, Viceroy and CAN (see [18]) have incorporated sequential neighbors. They are examples, like Pastry, of hybrid geometries. Our evaluation in this paper will explore the extent to which sequential neighbors, both naturally supported and artificially added, improve proximity and resilience.

Table 1 summarizes our discussion on the different routing geometries and their flexibility. Note that we enter slightly different constants for the flexibility in route selection for the different geometries. Specifically, we claim that the Ring and Hypercube have twice the flexibility in route selection compared to the Hybrid and XOR geometries.[7] Our simulation results in later sections validate this difference.

3. STATIC RESILIENCE

One of the reasons DHTs are seen as an excellent platform for large scale distributed systems is that they are resilient in the presence of node failures. This resilience has three different aspects, only one of which we explore here:

Data replication: When nodes fail, the data (or pointers) they are holding go with them. Measures must be taken so that this doesn't result in the loss of data from the system as a whole. Several (complementary) approaches have been proposed, most notably data replication [3, 4]. In our discussion, we assume that the degree of replication is adequate to prevent data loss, so the remaining question is whether one can *route* in the presence of failures.

Routing recovery: When failures occur, they deplete the routing tables in the remaining nodes. Recovery algorithms are used to repopulate the routing tables with live nodes, so that routing can continue unabated.

Static resilience: However, the recovery algorithms take some time to restore the routing tables, so one should still ask how well DHTs can route *before* routing state is restored. We call this *static resilience* because we keep the routing table static, except for deleting failed nodes, and ask how well routing performs. Hence, static resilience measures the extent to which DHTs can route around trouble even without the aid of recovery mechanisms that fix trouble. Thus, static resilience gives a measure of how quickly the recovery algorithm has to work; DHTs with low static resilience require much faster recovery algorithms to be similarly robust.

[6] One might argue that even on the tree, similar multiple paths exist if we were to allow "sideways" hops that maintain, but do not increase, the length of the longest prefix matched. However this would require some rule to prevent looping which would require an ordering of nodes that share the same prefix. This is precisely what Pastry achieves in defining a second distance metric.

[7] To see why this is true, consider a node on the ring routing to a destination at a distance between $[2^i, 2^{i+1}]$ from itself; for this node all of its first i neighbors make progress to the destination. Likewise, on the Hypercube, if a node differs from the destination on k bits then its k corresponding neighbors all make progress. In the Hybrid and XOR however, given a destination at a distance of between $[2^i, 2^{i+1}]$, a node's i^{th} neighbor will only make progress if its i^{th} bit differs from the destination's which is only true half the time.

While several papers consider resilience in the presence of active recovery algorithms, only a few examine static resilience. References [18, 22] address this issue in the contexts of CAN and Pastry respectively, while some recent work in reference [13] examines how certain graph theoretic properties of a overlay structure affect its static resilience.

Routing Geometry	Average Hopcount	Median Hopcount	90 Percentile Hopcount
XOR	7.7	8	10
Ring	7.4	7	10
Tree	7.7	8	10
Butterfly	21.4	21	28
Hypercube	7.7	8	10
Hybrid	7.7	8	10

Table 2: Comparing the hopcounts for different DHTs over a 65,536 node network with no failures.

To test for static resilience, we use a 65,536 node network. We allow different DHTs to populate their routing tables, ensuring that all the geometries (with the exception of the Butterfly[8]) store the same amount of state (number of the routing table entries) at any node. As shown in Table 2, the performance of the geometries (with the exception of the Butterfly) are very similar when there are no node failures. We now let some fixed fraction of uniformly chosen nodes fail and remove the entries corresponding to the failed nodes from the routing tables. We then try to route from each live node to every other live node and ask how often the routing can succeed. In particular, we look at two metrics:

- **% paths failed**: this describes how often routing was not able to connect two live nodes.
- **% increase in path length**: this describes the increase in path length, compared to the path length when there are no failed nodes.

In what follows, we consider the algorithms based on the following routing geometries: XOR, Ring, Tree, Butterfly, Hypercube, and Hybrid.

3.1 Performance Results

In this section, we discuss three questions:

Question #1: *How does the static resilience of various geometries compare?* The left graph in Figure 1 shows the results for % of failed paths as the % of failed nodes is varied. The results are very consistent with the degree of route selection flexibility in each geometry (see Table 1). The Tree and Butterfly have no route selection flexibility, and their resilience is quite poor; when 30% of the nodes have failed, almost 90% of their paths have failed. To the other extreme, the Ring and Hypercube geometries have the most flexibility in route selection, and their resilience is significantly better; when the same 30% of nodes are failed, under 7% of the routes have failed.[9] Intermediate between these two cases are the Hybrid and XOR geometries, which have about half the number of alternate paths (and hence, half the routing flexibility) that the Ring and Hypercube have. Their resilience is correspondingly inferior to the Ring and Hypercube, but far superior to the Tree and Butterfly; when the same 30% of nodes are failed, about 20% of routes have failed.

The right graph in Figure 1 shows the results for % increase in the average path lengths (or *path stretch*) as the % of failed nodes is varied. The path stretch of the Hypercube is minimal, consistent with our observation that all its alternative paths are of equal length. Ring suffers intermediate path stretch as some of its alternate paths are longer than the rest. All the other geometries incur significant path stretch because they have only longer alternate paths. The path stretch decreases as a large fraction of the nodes fail because very few routes succeed at high node failure rates, and those that succeed are very short.

Question #2: *How does the addition of sequential neighbors affect the static resilience of various geometries?* The previous results did not include sequential neighbors. In Figure 2, we consider what happens when we add 16 sequential neighbors to the various algorithms. We eliminate the XOR geometry because it doesn't support sequential neighbors, and the Tree is not included because it is represented by Hybrid. The most obvious result here is that sequential neighbors greatly increase resilience to path failures; no path failures are seen in any geometry even when 30% of the nodes have been failed. This suggests that DHTs, when equipped with sequential neighbors, can route successfully even under high node failure rates. The Ring performs significantly better than the Hypercube and others whose sequential neighbors are artificially grafted on. However, the increase in resilience to path failures comes at the cost of path stretch. All the algorithms suffer significantly greater path stretch (now that the Hypercube has sequential neighbors, not all paths are of equal lengths).

Question #3: *Are sequential neighbors better than regular neighbors for ensuring static resilience?* While the previous results were specific to sequential neighbors, one could conjecture that increased resiliency could equally well be achieved just by increasing the total number of neighbors without insisting that they be sequential. Thus, we now ask whether sequential neighbors are especially useful in increasing resilience. We do this by considering the Ring geometry and compare cases where the total number of sequential and regular neighbors are the same, but the number of sequential neighbors are different. The results are shown in Figure 3. The left graph in the figure indicates that at high node failure rates, sequential neighbors are better than regular neighbors at increasing resilience to path failures but, the right graph indicates that they can lead to significantly longer paths (note that the Y-axis of the right graph shows path hop-counts and not path stretch). Hence, the use of sequential neighbors might be the preferred option if one were only concerned with routing success and not other metrics such as the total path latency.

To summarize, our results confirm that the static resilience of a geometry is largely determined by the amount of routing flexibility it offers. Thus, the Ring which has the greatest routing flexibility has the highest resilience, while Tree and Butterfly which have the least flexibility in routing have the least resilience. Further, the addition of sequential neighbors can make DHTs significantly more resistant to path fail-

[8]Unlike other DHTs, the amount of state stored by a Butterfly node is always a constant and cannot be controlled.

[9]Ring performs better than Hypercube as even though they have the same number of $\log n$ length paths, Ring has many alternate paths that are longer than $\log n$, while Hypercube has none.

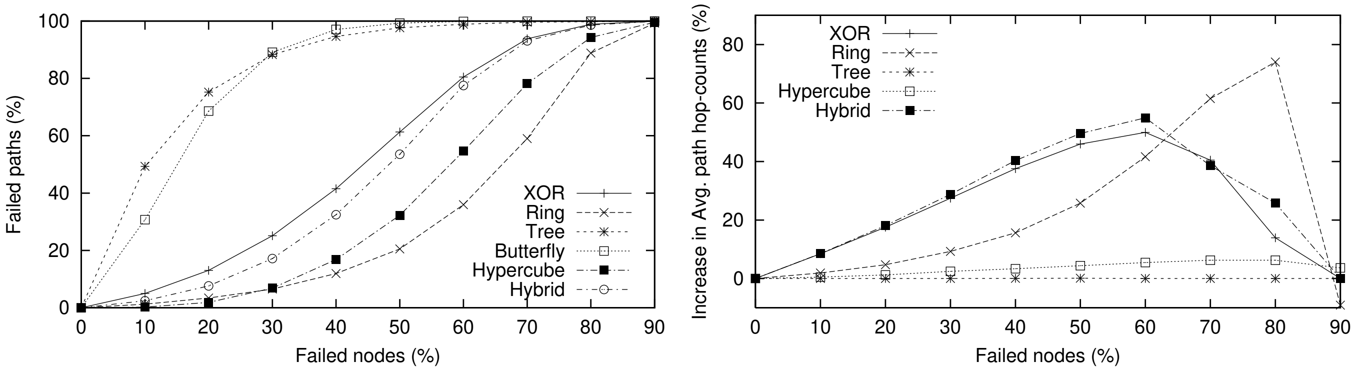

Figure 1: Left: Percentage of failed paths for varying percentages of node failures across different routing geometries. Right: Percent increase in average path hop-counts of successful paths for varying percentages of node failures across different routing geometries. The Butterfly is left off of this graph because so few routes are usable, and those that are sometimes take *shorter* paths than the original ones, resulting in a negative path stretch.

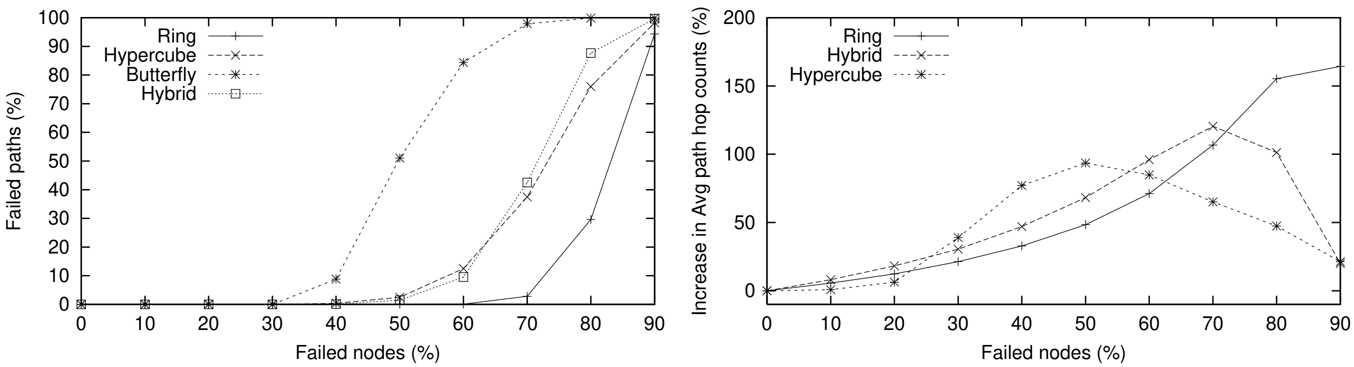

Figure 2: Left: Percentage of failed paths for varying percentages of node failures across different routing geometries. Right: Percent increase in average path hop-counts of successful paths for varying percentages of node failures across different routing geometries. Butterfly is left off of this graph because its path increase is so much higher than the others, reaching 700%, that it would distort the y-axis. All algorithms use 16 sequential neighbors.

ures, though path stretch can get much worse. Finally, for the Ring topologies, replacing additional sequential neighbors with regular neighbors yields a similar, but smaller, increased resistance to path failures but a much smaller path latency.

4. PATH LATENCY

DHTs are designed to provide efficient routing as measured in terms of hopcount (the number of overlay hops between the source node and the destination node). While hopcount is an important metric for measuring the processing and bandwidth requirements at the peers, it does not adequately address the issue of end-to-end latency because each overlay hop could potentially involve significant delays (intercontinental links, satellite links, etc.). As a result, there has been much recent effort to reduce end-to-end latencies in DHT routing algorithms by considering the relative *proximity* of overlay nodes (*i.e.*, the IP latency between them) [2, 8, 9, 10, 16, 21, 27]. The proposed methods fall into three broad categories, two of which we consider here.

Proximity Neighbor Selection (PNS): The neighbors in the routing table are chosen based on their proximity.

Proximity Route Selection (PRS): Once the routing table is chosen, the choice of next-hop when routing to a particular destination depends on the proximity of the neighbors.

Proximity identifier Selection (PIS): As explored in [21], one can pick the node identifiers based on their geographic location. Since this makes load balancing hard, and increases the likelihood of correlated failures, we don't discuss this method here.

We thus consider the two proximity methods PNS and PRS. The section begins with a short description of these methods, and the rest of the section is devoted to their analysis. While our evaluation is based on recursive (as opposed to iterative [3]) routing, we believe that our key conclusions regarding the relative performance of PNS and PRS hold true for iterative routing too. Confirming this belief is the subject of future work.

Evaluating proximity methods requires the topology of an underlying network along with its link latencies. Testing proximity methods on only one or two topologies doesn't ensure that the results will generalize. So, after defining PNS and PRS, we discuss whether we can more generally under-

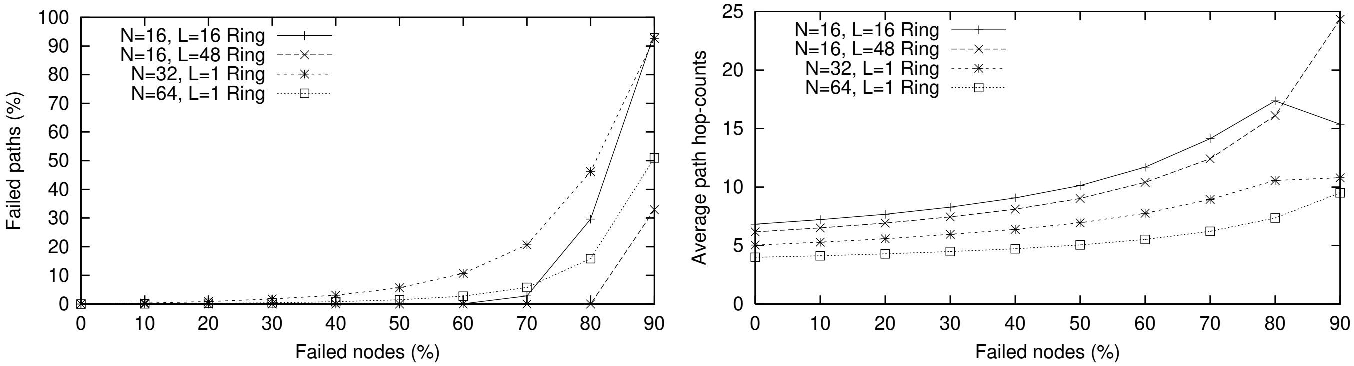

Figure 3: Left: Percentage of failed paths for varying percentages of node failures for a Ring geometry for varying numbers of neighbors (N) and sequential neighbors (L). Right: Average path hop-counts of the successful paths for varying percentages of node failures.

stand the role of topology and link latencies in proximity methods.

After these preliminaries, we finally address the question of how geometry affects path latency in DHT routing algorithms. Our analysis style here is different from that of the previous section on static resilience; there, we were comparing the detailed differences in geometries. Here, we will find that the geometries themselves make less difference than whether or not they can support PNS and/or PRS. Thus, this section is really a comparison of those two proximity methods. As discussed in Section 2, some geometries are capable of PNS, some of PRS, and others of both. In this section, we wish to evaluate the relative performance of PNS and PRS to see whether restrictions on adopting them is a significant hindrance.

4.1 PNS and PRS

In DHT algorithms that have flexibility in choosing neighbors, typically these neighbors have to be chosen from some subset of the identifier space. The ideal PNS algorithm would be to select the closest neighbors (as measured by latency) in these subsets. For example, the subset for the i^{th} neighbor of a node a in a Tree geometry is the identifier space of the sub-tree at depth i containing the node, while in a Ring geometry it is the identifier space $[(a+2^i),(a+2^{i+1})]$. However, identifying the closest nodes is hard in practice, as the sizes of the subsets grow exponentially with i. So, various heuristics have been proposed in [2, 8] to approximate the performance of ideal PNS. Here, we define one such heuristic, dubbed PNS(K) that uses random sampling. PNS(K) samples K consecutive nodes starting from the first element in the relevant subset and picks the closest one. We don't dwell here on how one should pick K, but in general a node can make a reasonable choice of K after inspecting its latency distribution (see below). From now on, we use the term PNS to refer to ideal PNS.

The PRS algorithms have to deal with a more complicated tradeoff between the number of hops and the latency. Any neighbor closer to the destination in the identifier space is a *valid* next hop, and without proximity, the next hop is chosen in a greedy fashion to decrease the number of hops. While there are a number of heuristics that trade hops for latency, we focus on three heuristics that we found effective for each of the Ring, XOR and Hypercube geometries.

The heuristic for Ring takes advantage of the multiple paths with equal number of hops to a destination (see Table 1) and chooses the next hop from a subset of neighbors, called the candidate set, which do not (usually) increase the routing path hops. To select the candidate set, the distance to the destination is expressed in binary notation, and neighbor i is chosen to the set if there is a 1 in the i^{th} position. The closest member of the candidate set is picked as the next hop. When coupled with PNS(K), the algorithm disallows the closest $\log k$ neighbors from the candidate set (unless, of course, the destination lies within the closest $\log k$ neighbors).[10] However, this heuristic cannot be applied to XOR as its geometry does not have the luxury of multiple paths with the same number of hops to the destination. Our PRS heuristic for XOR takes a non-greedy next hop only when its latency is smaller than the latency of the greedy next hop choice by more than the average latency in the network. This primarily helps to avoid very long hops. In Hypercube, all the alternate paths have the same number of hops, so our PRS heuristic is very simple. From the valid next hops, we pick the one with smallest latency.

We now investigate the role of topology in determining the effectiveness of these proximity methods.

4.2 Role of Topology and Latency

One of the aspects that makes it hard to understand proximity methods is that their performance depends so critically on the underlying topology and its latency characteristics. While there is a large literature describing possible approaches to topology modeling – starting with the initial random graphs of Waxman, to the structural generators in GT-ITM (transit-stub and tiers), to the more recent set of power-law degree-based generators – there is little known about how to assign latencies in such a topology. While previous studies [2, 8, 9, 21] evaluate proximity methods using one or more of these topology generators along with some rather *ad hoc* choice of latency assignments, they don't analyze how their choice of latencies affects the performance of the proximity methods. Thus, research into proximity methods is now in a position where we neither know how to describe the real-world latencies nor understand their effect on our proposed algorithms.

[10] Thus, this version of PRS adds nothing when combined with the ideal PNS algorithm, which effectively has infinite K.

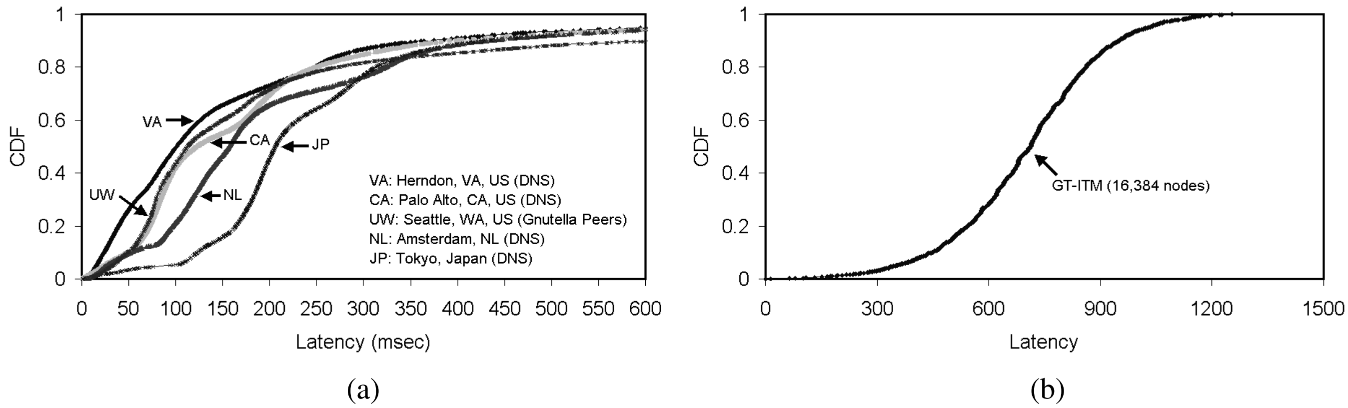

Figure 4: The CDF of latency distributions for (a) the Internet as seen from different geographic locations and (b) a 16,384 node GT-Itm topology as seen from a typical node.

We propose one possible way out of this bind. We conjecture that the effect of topology and latencies, for a large class of networks including the current Internet graph, can be reasonably well approximated by looking only at the latency distribution as seen from a "typical" node. That is, we conjecture that when choosing neighbors or next-hops, it is a reasonable approximation to consider the set of possibilities as coming from an independent drawing from the given latency distribution.[11]

If this conjecture holds, then there are two immediate benefits. First, one can empirically measure the latency distribution of the Internet from various suitably located hosts, so that one need not guess at latency assignments in an Internet topology model. Second, given this measured distribution one can *compute* (not merely simulate) an approximation to the expected performance of a proximity method. To evaluate our conjecture, we simulate the performance of the various proximity methods over a latency annotated network topology and compare them with their performance computed using only the latency distribution seen by a random node in the topology. In doing so, we make a further approximation that the latency distribution is uniform across all nodes. As we will see below, the results from this very simple and rough approximation agree rather well with our simulation results (see Figure 5).

To illustrate the real-world latencies, we used data from the Skitter project [17] and a P2P measurement project [23] to plot the latencies to a large number of end hosts spread across the Internet as seen from various geographical locations in Figure 4(a) (similar measurements can be seen in [26] and elsewhere). The end hosts measured in the Skitter project cover a large fraction of routable IP prefixes, while those measured in the P2P project are Gnutella hosts. A striking feature common to all these latency graphs is that the curves rise sharply in a certain latency range, indicating a heavy concentration of nodes within the latency range. We note that these latency graphs differ significantly from the assumptions required in [10, 16] to prove their bounds. In Figure 4(b), we show the latency distribution from a typical node in our 16,384 node GT-ITM topology that we used for our simulation results presented later. An important difference between the observed and GT-ITM latencies is that a non-negligible fraction of the observed real-world latencies are very large. For the GT-ITM case, there are no paths that are more than double the median latency, whereas in all the observed distributions atleast 10% of the paths have latencies double that of their median.

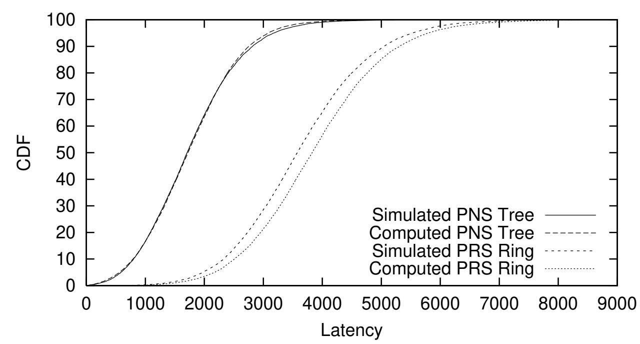

Figure 5: The CDF of path latency distributions for PNS Tree and PRS Ring derived in two ways: *simulated* using a GT-ITM topology and *computed* only using its latency distribution.

Figure 5 compares the computed and simulated results for two algorithms, PRS Ring and PNS Tree, on our GT-ITM topology with 16,384 nodes. Links between two transit nodes are assigned a latency of 100, while those between a transit node and a stub node are assigned 20. The latencies of links between two stub nodes are set to 5. Note that the computations agree well with the simulations confirming our conjecture earlier that computations over the latency distribution seen by a typical node provides a reasonable approximation to the actual simulations. The results for PRS are not as close as those of PNS, and this is likely due to the fact that PRS algorithms typically lead to variations in the hop count. However, the differences between simulated and computed results are far less than the differences between the GT-ITM and observed latency distributions. Thus, whatever precision we have lost by using computations rather than simulations are more than made up for by the increase in accuracy in modeling reality.

In what follows, we will consider only XOR, Ring, Hypercube, and Tree. The Hybrid algorithm and the Tree algorithm are essentially identical when there are no node failures.[12] The Butterfly does not admit either PNS or PRS.

[11] We acknowledge that this approximation suffers if the latency distribution varies substantially and qualitatively from point-to-point. However, the approximation need only be good enough to capture the relative merits of different approaches, and is not intended to provide quantitatively accurate descriptions of a method's performance.

[12] One minor point: while the Tree does not admit PRS, the

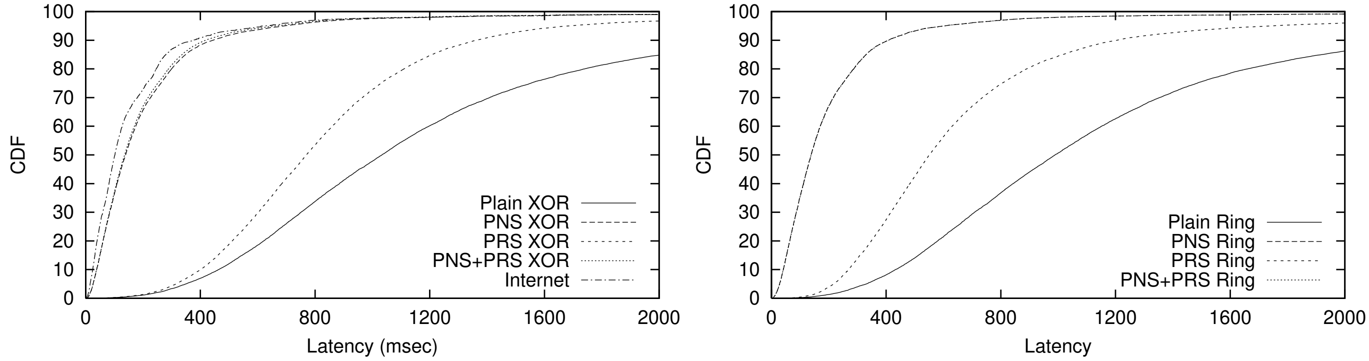

Figure 6: The CDF of path latency distributions for the Plain, PNS, PRS and PNS+PRS versions of XOR (left) and Ring (right) routing geometries. In addition, the CDF for the underlying Internet latency distribution is plotted for comparison. On the right graph the PNS+PRS curve lies on top of the PNS curve, so it isn't directly visible. All algorithms used 1 sequential neighbor and are computed using a real-world latency distribution (marked VA in Figure 4).

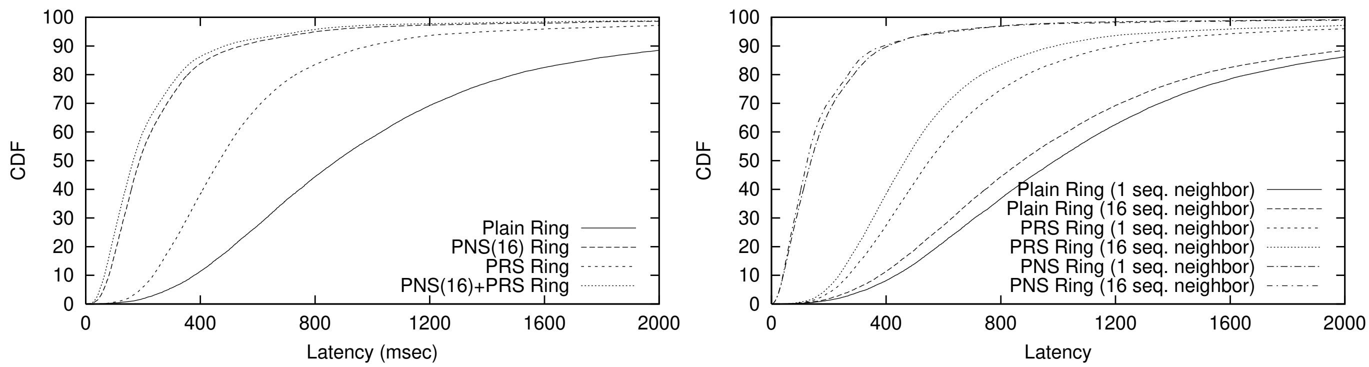

Figure 7: Left: The CDF of path latency distributions for the Plain, PNS(16), PRS and PNS(16)+PRS versions of Ring. These algorithms used 16 sequential neighbors. Right: The CDF of path latency distributions for the Plain, PRS and PNS versions of the Ring using 1 or 16 sequential neighbors as marked in the titles of the curves. These are computed using a real-world latency distribution (marked VA in Figure4).

Geometry	No Proximity	PNS	PRS
XOR	9	9	11
Ring	9	9	9
Tree	10	10	N/A
Hypercube	9	N/A	9

Table 3: These 90^{th} percentile hopcounts for the different DHTs show that the hopcounts do not change significantly when using various proximity methods. The network used had 16384 nodes.

Our results below are obtained over a 16,384 node network. The 90th percentile hopcounts for the different geometries when using various proximity methods are summarized in Table 3 to confirm that the gains in path latency reported below do not come at a significant cost for the path hopcounts.

4.3 Performance Results

For the rest of this section, we address three questions.

Hybrid geometry could be extended to accommodate PRS. We don't pursue that extension here.

Question #1: *Which is more effective, PNS or PRS?* Both methods have been proposed in the literature, but their effectiveness has never been compared. To hold fixed the effect of the underlying geometry, we compare the two approaches in the two geometries that can accommodate both: XOR and Ring. Figure 6 shows results for the Plain, PNS, PRS, and PNS+PRS versions of the XOR and Ring algorithms. In both cases, the PRS version shows a significant improvement over the Plain version, but far more improvement is realized by the PNS version. In addition, adding PRS to PNS gives only a small improvement over the PNS alone.

To understand why PNS is better than PRS, consider a node in a densely packed Ring geometry trying to route to a identifier that is at a distance between $[2^i, 2^{i+1}]$ from itself. With PNS the node would deterministically pick its i^{th} neighbor, but that neighbor is selected from any of the 2^i nodes with identifiers between $[2^i, 2^{i+1}]$. In contrast, PRS can choose from any of its first i neighbors, but each such neighbor is deterministically chosen. Thus, PNS chooses among 2^i options while PRS choses among i options, resulting in improved performance for PNS.

These results reported above are for "ideal" PNS. To see whether the results change when only sampling a small subset, we compare the various design options when $K = 16$, where K is the sample size. The left graph of Figure 7 shows that the results are similar in this case, with PNS(16) performing significantly better than PRS, although the performance improvement of PNS(16)+PRS over PNS(16) alone is somewhat more than the performance improvement of PNS+PRS over PNS.

To see the impact of adding sequential neighbors on the results, we compare the performance of the Plain, PNS and PRS versions of the Ring after adding 16 sequential neighbors in the right graph of Figure 7. While the improvements are noticeable for the Plain and the PRS versions, they are not large enough to affect the comparative results in a significant way.

Until now our results used the latency distribution seen by a node in Virginia, on the east coast of the USA (marked VA in Figure 4). This distribution, while similar to those seen from the west coast of the USA and Europe (marked CA and NL), is considerably different from that seen from the Japan (marked JP). To test the consistency of our results across two very different latency distributions, we compute the performance of the proximity methods over XOR geometry using the JP latency distribution and show the results in Figure 8. Comparing this graph with the left graph of Figure 6, we notice that the relative performance of the proximity methods are very similar, with PNS still outperforming PRS by a wide margin, although the absolute performance differs markedly between the two latency distributions.[13]

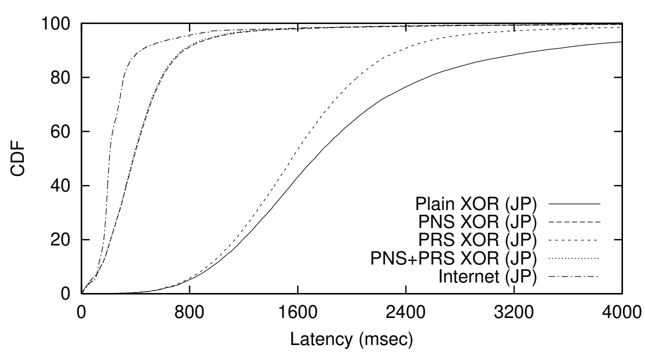

Figure 8: The CDF of path latency distributions for the Plain, PNS, PRS, PNS+PRS versions of the XOR routing geometry. These algorithms used 1 sequential neighbor and are computed using the real-world latency distribution marked JP in Figure 4.

To conclude, when considering path latency, it is important that the geometry accommodate PNS; accommodating PRS, at least for the sake of path latency, does not appear to be important.

Question #2: *Does the underlying geometry matter, other than determining whether PNS and/or PRS can be used?* A geometry's flexibility determines whether it can accommodate PNS and/or PRS, but here we ask whether geometry affects path latency beyond this distinction. Figure 9 compares the performance of three pairs: PNS+PRS Ring versus PNS+PRS XOR, PNS Ring versus PNS Tree (which cannot implement PRS), and PRS Ring versus PRS Hypercube (which cannot implement PNS). The performance of each pair of designs is very close, suggesting that what really matters is the ability to implement PNS and PRS, not the other factors in geometry.

Question #3: *What is the absolute performance of these proximity methods?* The results so far compare the relative performance of proximity methods, but do not address the question of how well they do in an absolute sense. While this question has been addressed in papers proposing individual proximity designs, we revisit it here because our tests use a more realistic latency distribution and therefore may be more indicative of real-world performance. As the absolute performance depends on the exact latency distribution used, in Table 4 we show the median latencies of the various designs for two very different latency distributions. As can be seen, the very best options (PNS and PNS+PRS) fare quite well for either distribution. While plain XOR has a ratio of roughly 10 between the latencies, PNS+PRS XOR has a ratio less than 2. Thus, the available proximity methods can reduce the end-to-end latencies in the overlay to a very small multiple of the underlying Internet path latencies (which is consistent with the findings in [2, 8]).

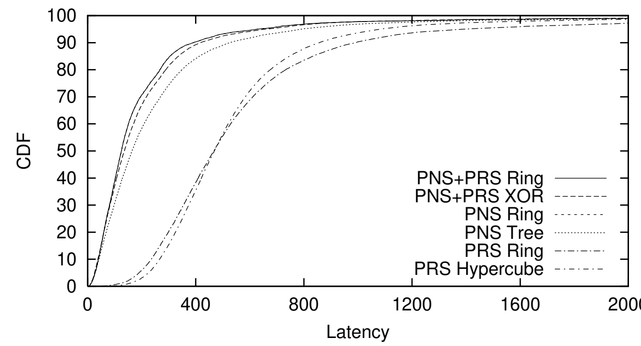

Figure 9: The CDF of path latency distributions for PNS+PRS Ring, PNS+PRS XOR, PNS Ring, PNS Tree, PRS Ring, PRS Hypercube. These are computed using a real-world latency distribution (marked VA in Figure4). The PNS+PRS Ring lies on top of the PNS curve, so it is not directly visible. These algorithms used 1 sequential neighbor.

Routing Algorithm	Median Hopcount	VA Median Latency	JP Median Latency
Internet	1	102	206
XOR	7	1036	1725
PNS XOR	7	139	385
PRS XOR	8	770	1557
PNS+PRS XOR	7	136	381

Table 4: Comparing the median latency and hopcounts of XOR based overlay paths and the underlying IP paths for two extremely different latency distributions (marked VA and JP in Figure 4). The latencies are in milliseconds.

[13] Though for lack of space we don't present performance results for GT-ITM latency distributions, we note here that the results are somewhat different than what we observed for more realistic latency distributions. Specifically, the performance gap between PRS and the Plain is much smaller and the performance gap between PNS and PNS(16) is much larger in the GT-ITM case than for the real-world latency distributions. Thus, we urge caution when using GT-ITM topologies to evaluate proximity methods.

To summarize our discussion in this section, we find that while both neighbor selection (PNS) and route selection (PRS) can help in finding shorter paths, PNS yields significantly better paths than PRS. Further, the effectiveness of these proximity methods does not depend on the choice of the routing geometry. Thus, geometries such as Tree, XOR and Ring that support PNS perform considerably better than geometries such as Hypercube that support only PRS. While XOR and Ring can accommodate both PNS and PRS (unlike Tree), the additional benefit of supporting PRS over PNS appears to be quite limited. However, the ability to accommodate both may be an advantage while using PNS(K), a limited but more practical version of PNS.

5. LOCAL CONVERGENCE

Local convergence is another issue that arises when considering the effects of the underlying network latencies. Local convergence, first identified in [2], is the property that two messages, sent from two nearby (in terms of latency) nodes addressed to the same location, converge at a node near the two sources. This property leads to low latencies and/or bandwidth savings in several different uses of DHTs, including the following three:

Overlay multicast: When setting up a multicast tree [1, 20, 29], one cares about both the lengths of each individual path (considered in the previous section) and the overall efficiency of the entire multicast tree. This latter quantity is improved if DHT routing has the local convergence property mentioned above.

Caching: One way to speed access is to cache pointers along the retrieval path as described in [3, 4]. If the DHT has good local convergence, then any nearby node requesting the same content can make use of these cached content.

Server selection: Similar to caching above, and as explained in [16, 28], clients can more easily find nearby servers if pointers to them are stored along the path to the "root."

In this section we seek to understand the impact of geometry on local convergence. In general, local convergence depends on the nature of the underlying topology and the exact location of the sources and destination. To provide a simplified and controlled experiment, we consider the case of an *isolated domain*. We consider m nodes that are within some very small latency of each other; all nodes in this isolated domain are some large latency away from the other $n - m$ nodes. We then let each node in the isolated domain contact the same (randomly chosen) destination outside of the domain. The measure of local convergence is how many *exit points* there are; that is, how many nodes in the domain relay the message to an off-domain node. In the best case, only one node sends a message off-domain; messages from all other domain nodes converge on this point before leaving the domain.

We test convergence on the Ring, Tree and XOR geometries.[14] Figures 10 and 11 plot the measured number of exit points for increasing domain sizes (n is held fixed while m varies from 1 to $\frac{n}{2}$). Our results can be organized around three questions:

Question #1: *Which is more effective, PNS or PRS?* Figure 10 shows the results for the PNS, PRS, and PNS+PRS versions of both XOR and Ring. In both cases, PNS and PNS+PRS provide almost optimal performance, whereas PRS does little to limit the number of exit points (except for very large m in the case of Ring). Thus, PNS is far more effective than PRS. Combining PRS with PNS helps somewhat for Ring but almost none at all for XOR. The ineffectiveness of PRS isn't surprising because, at the simulated system size of 65536, each node has 16 neighbors. Until the domain size is a large fraction of the total population, it is unlikely that one of these neighbors is within the domain, and so PRS would have little effect.

Question #2: *Does this answer change when only considering PNS(K)?* Figure 11, left, shows the results for the PNS(16), PRS, PNS(16)+PRS versions of Ring. Because the sampling is limited to 16, proximity neighbor selection is not ideal and its effectiveness is greatly reduced. In fact, PRS and PNS(16) are equally ineffective. The combination of the two, PNS(16)+PRS is more effective, particularly at moderate sized domains. When the domain size is small, a limited amount of sampling (whether choosing neighbors or routes) isn't much help. The combination of the two, PNS(16)+PRS, increases the level of sampling and so its effectiveness kicks in for lower values of m. Thus, PRS might play an important role in local convergence if the domain sizes of interest are small and the sampling used to implement PNS is limited.

Question #3: *How does the performance of the various geometries compare?* Figure 11, right, compares the results for PNS Tree, PNS+PRS Ring, and PNS+PRS XOR. As can be seen, these all perform roughly the same. Thus, as with path latency, the biggest difference between geometries is whether or not they can accommodate PNS and/or PRS.

Our results suggest that the relevance of PRS depends on whether or not PNS can be closely approximated (at least more closely than PNS(16) does). If not, then implementing PRS provides significant value; if so, then PRS may not be needed as PNS by itself provides almost all the performance of PNS+PRS. References [8, 2] propose methods for efficiently approximating PNS, but it is not yet known how much better they are than PNS(16) for these scenarios.

6. DISCUSSION

This paper has not introduced any new DHT algorithms, nor has it presented any theorems. However, we hope that it has provided some pieces of insight that will be useful in future DHT routing designs. At a very high level, our findings can be summarized as follows:

Component-based analysis: When comparing DHT algorithms we advocate analyzing the component design decisions separately rather than comparing DHTs as black-boxes or turn-key systems. In particular, this requires separating systems-level design decisions, which usually are independent of the routing, from routing-level design decisions. This is more philosophy than science, but we think our approach, while not as effective in picking the best current design, is more conducive to creating better designs in the future.

[14] We expect PRS Hypercube to offer convergence similar to that of the PRS Ring and hence do not explore the Hypercube here. Similarly, we omit the Hybrid as we expect it to offer performance similar to that of the Ring.

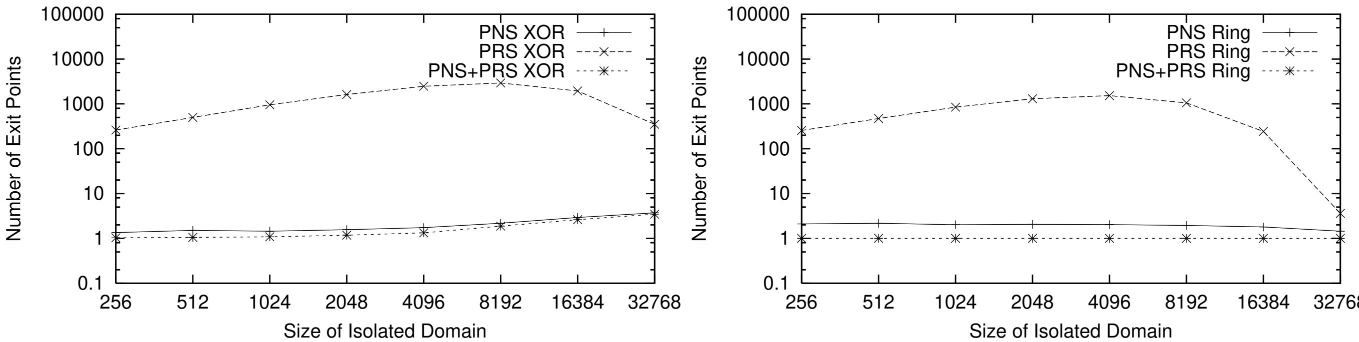

Figure 10: The number of exit points for a system of size 65536 with varying sizes of isolated domains. The left graph shows results for the PNS, PRS and PNS+PRS versions of XOR, while the right graph shows results for the PNS, PRS and PNS+PRS versions of Ring.

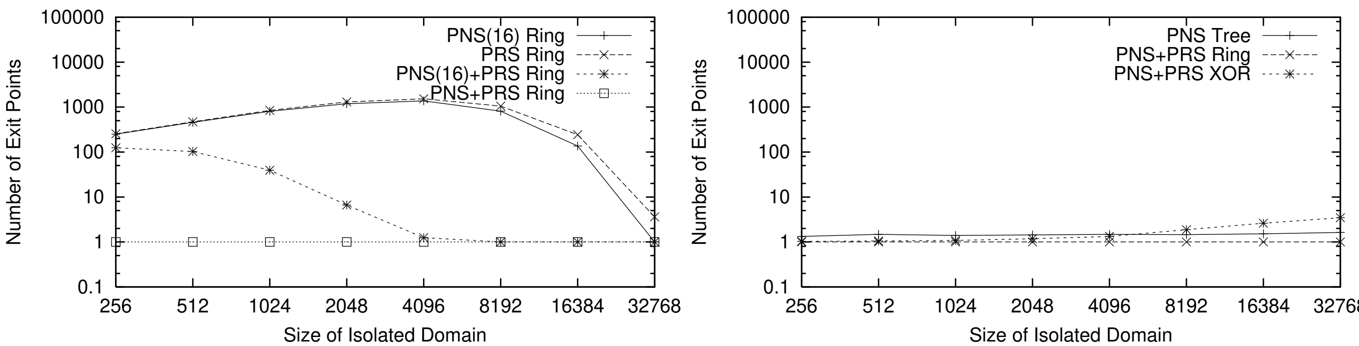

Figure 11: The number of exit points for a system of size 65536 with varying sizes of isolated domains. The left graph shows results for the PNS(16), PRS and PNS(16)+PRS versions of Ring, while the right graph shows results for the PNS Tree, PNS+PRS Ring, and PNS+PRS XOR.

Routing geometry is fundamental: The choice of a routing *geometry* constrains other routing design issues. While there are a myriad of detailed routing designs yet to be explored, the space of routing geometries is more limited. Hence, one might reach some consensus on the best routing geometry well before reaching any consensus about the various design details.

Flexibility is important: The most important difference we noticed between geometries (besides the butterfly geometry's ability to achieve $O(\log n)$ paths with $O(1)$ neighbors) is the degree of *flexibility* they offer. Flexibility describes the amount of freedom available to choose neighbors and next-hop paths. This freedom, in turn, affects the performance in areas such static resilience, path latency, and local convergence.

Ring and XOR are flexible: The Ring and XOR geometries were the only ones we tested that could freely choose both neighbors and routes, so they could implement both PNS and PRS. While PNS is significantly more effective than PRS in dealing with proximity, there are times, as we saw in Section 5, when PRS is an important complement to PNS. Thus, the ability to support both is an advantage.

Why not the Ring? The Ring geometry has unsurpassed flexibility and, in addition, provides natural support for sequential neighbors. It achieved the highest performance in our resiliency tests, and was as good as any other geometry in the proximity metrics of path length and local convergence. Thus, our investigation showed no advantage to the other geometries, and a slight advantage to the Ring. While our initial inclination was to favor more complicated geometries, the question we end this paper with is: why not use ring geometries?

However, we pose this as a question, not a conclusion. There is much more to be done before any definitive judgements can be drawn. Our investigation is incomplete in many aspects. For example, our study could be extended to a wider class of routing geometries, theoretical bounds could be derived for many of our simulation results, and the impact of a routing geometry (and in particular its symmetry or the lack thereof) on the cost of maintaining the associated overlay structure should be studied. Thus, we view our paper as only the first step in a more fundamental investigation of routing algorithms.

7. ACKNOWLEDGEMENTS

We wish to thank several anonymous reviewers and our shepherd Antony Rowstron for their feedback and suggestions. We thank Dahlia Malkhi and David Ratajczak for their help with the implementation of Viceroy.

8. REFERENCES

[1] M. Castro, M. Jones, Anne-Marie Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An Evaluation of Scalable Application-Level Multicast Built Using Peer-To-Peer Overlays. In *Proceedings of the INFOCOM 2003*, San Francisco, April 2003.

[2] Miguel Castro, Peter Drushel, Y.C. Hu, and Antony Rowstron. Exploiting Network Proximity in Peer-to-peer Networks. Technical Report MSR-TR-2002-82, Microsoft Research, 2002.

[3] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, Lake Louise, AB, Canada, October 2001.

[4] Peter Druschel and Antony Rowstron. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, Lake Louise, AB, Canada, October 2001.

[5] IRIS: Infrastructure for Resilient Internet Systems. http://iris.lcs.mit.edu, May 2002.

[6] David Karger Frans Kaashoek. Simple Constant-Space Distributed Hash Tables. In *Proceedings of the IPTPS 2003*, Berkeley, February 2003.

[7] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. One Hop Lookups for Peer-to-Peer Overlays. In *Proceedings of the HotOS-IX 2003*, Hawaii, May 2003.

[8] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed Object Location in a Dynamic Environment. In *Proceedings of the ACM SPAA*, 2002.

[9] Sushant Jain, Ratul Mahajan, and David Wetherall. A Study of Performance Potential of DHT-based Overlays. In *Proceedings of the 4th Usenix Symposium on Internet Technologies and Systems (USITS)*, Seattle, WA, USA, March 2003.

[10] David R. Karger and Matthias Ruhl. Finding Nearest Neighbours in Growth-restricted Metrics. In *Proceedings of the ACM STOC*, Montreal, May 2002.

[11] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the ACM STOC 2000*, 2000.

[12] John Kubiatowicz. Oceanstore: An Architecture for Global-Scalable Persistent Storage. In *Proceedings of the ASPLOS 2000*, Cambridge, MA, USA, November 2000.

[13] Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems; Routing Distances and Fault Resilience. In *Proceedings of the ACM SIGCOMM '03 Conference*, Karlsruhe, Germany, August 2003.

[14] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable Dynamic Emulation of the Butterfly. In *Proceedings of the PODC*, 2002.

[15] Petar Maymounkov and David Mazieres. Kademlia: A Peer-to-peer Information Systems Based on the XOR Metric . In *Proceedings of the IPTPS 2002*, Boston, March 2002.

[16] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proceedings of the ACM SPAA*, June 1997.

[17] CAIDA: The Skitter Measurement Project. www.caida.org/tools/measurement/skitter/index.html, 2002.

[18] Sylvia Ratnasamy. *A Scalable Content-Addressable Network*. PhD thesis, University of California, Berkeley, October 2002.

[19] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, San Diego, CA, USA, August 2001.

[20] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level Multicast using Content-Addressable Networks. In *Proceedings of the NGC 2001*, 2001.

[21] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proceedings of the INFOCOMM*, 2002.

[22] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[23] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-peer File Sharing Systems. In *Proceedings of the Multimedia Computing and Networking Conference (MMCN)*, San Jose, CA, USA, January 2002.

[24] John Kubiatowicz Sean Rhea, Timothy Roscoe. DHTs Need Application-Driven Benchmarks. In *Proceedings of the IPTPS 2003*, Berkeley, February 2003.

[25] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM 2001*, San Diego, CA, USA, August 2001.

[26] Marcel Waldvogel and Roberto Renaldi. Efficient Topology-Aware Overlay Network. In *Proceedings of the HotNets-I 2002*, Princeton, October 2002.

[27] Ben Y. Zhao, Anthony Joseph, and John D. Kubiatowicz. Locality Aware Mechanisms for Large-scale Networks. In *Proceedings of the FuDiCo 02*, Bertinoro, Italy, June 2002.

[28] B.Y. Zhao, K.D. Kubiatowicz, and A.D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley, April 2001.

[29] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proceedings of the NOSSDAV 2001*, 2001.