Incrementally Improving Lookup Latency in Distributed Hash Table Systems

Hui Zhang [*] Ashish Goel [†] Ramesh Govindan [‡]

Abstract

Distributed hash table (DHT) systems are an important class of peer-to-peer routing infrastructures. They enable scalable wide-area storage and retrieval of information, and will support the rapid development of a wide variety of Internet-scale applications ranging from naming systems and file systems to application-layer multicast. DHT systems essentially build an overlay network, but a path on the overlay between any two nodes can be significantly different from the unicast path between those two nodes on the underlying network. As such, the lookup latency in these systems can be quite high and can adversely impact the performance of applications built on top of such systems.

In this paper, we discuss a random sampling technique that incrementally improves lookup latency in DHT systems. Our sampling can be implemented using information gleaned from lookups traversing the overlay network. For this reason, we call our approach *lookup-parasitic random sampling* (LPRS). LPRS is fast, incurs little network overhead, and requires relatively few modifications to existing DHT systems.

For idealized versions of DHT systems like Chord, Tapestry and Pastry, we analytically prove that LPRS can result in lookup latencies proportional to the average unicast latency of the network, provided the underlying physical topology has a power-law latency expansion. We then validate this analysis by implementing LPRS in the Chord simulator. Our simulations reveal that LPRS-Chord exhibits a qualitatively better latency scaling behavior relative to unmodified Chord.

Finally, we provide evidence which suggests that the Internet router-level topology resembles power-law latency expansion. This finding implies that LPRS has significant practical applicability as a general latency reduction technique for many DHT systems. This finding is also of independent interest since it might inform the design of latency-sensitive topology models for the Internet.

1 Introduction

Distributed Hash Table (DHT) systems [18, 27, 23, 30] are a new class of peer-to-peer networks which provide routing infrastructures for scalable information storage and retrieval. Such systems comprise a collection of nodes (usually end systems) organized in an overlay network. They support scalable and distributed storage and retrieval of $\langle key, data \rangle$ pairs on the overlay network. They do this by associating each node in the network with a portion of the key space; all data items whose keys fall into a node's key space are stored at that node. Given a key, retrieving the data item corresponding to the key involves a *lookup* operation, *viz.*, routing the retrieval request to the corresponding node. Different DHT systems differ in the details of the routing strategy as well as in the organization of the key space.

[*] Department of Computer Science, University of Southern California. This research was supported in part by NSF grant CCR0126347. Email: huizhang@usc.edu.

[†] Departments of Management Science and Engineering and (by courtesy) Computer Science, Stanford University. This research was done while the author was at the University of Southern California, and was supported in part by NSF grant CCR0126347 and NSF Career grant No. 0133968. Email: ashishg@stanford.edu.

[‡] Department of Computer Science, University of Southern California. This research was supported in part by NSF grant CCR0126347. Email: ramesh@usc.edu.

With the advent of this class of systems, researchers have proposed a wide variety of applications, services, and infrastructures built on top of a DHT system. Examples of such proposals include systems for wide-area distributed storage [6, 13], naming [5], web caching [10], application-layer multicast [20], event notification [24], indirection services [26], and DoS attack prevention [12]. As such, DHT systems hold great promise for the rapid deployment of Internet-scale applications and services.

However, because in most DHT systems a request will take $\theta(logN)$ overlay hops in average (N: the network size), the routing latency between two nodes on the overlay network can be different from the unicast latency between those two nodes on the underlying network. The ratio of these two quantities is called the *latency stretch* (or, simply, the stretch). A DHT network might have a large latency stretch; this happens if the overlay network topology is not congruent with the underlying IP topology. A single logical hop on the overlay network could incur several hops in the underlying IP network (Figure 1). Large latency stretch factors can clearly result in poor *lookup performance*, and can adversely affect the performance of applications and services that use DHT systems.

Sophisticated algorithms are known for designing DHT systems with low latency. Plaxton *et al.* [17] give an elegant algorithm for a DHT system that achieves nearly optimal latency on graphs that exhibit power-law expansion, while preserving the scalable routing properties of the DHT system. However, a direct implementation of this algorithm requires pairwise probing between nodes to determine latencies, and is not likely to scale to large overlays.

To improve latency stretch, existing DHT systems propose different practical approaches that are closely tailored to their routing strategies. For example, CAN [19] proposes a distributed binning scheme to construct an overlay topology which closely resembles the underlying IP topology. Tapestry [22] enhances its network with a probabilistic location algorithm based on *attenuated bloom filter* data structure. For each request, a Tapestry node uses the probabilistic algorithm to search every node within a small number of hops (hops on the underlying topology) before forwarding the request to the next overlay node. Pastry[4] improves latency by selecting a nearby node when initializing the routing table at a new node. These approaches *heuristically* improve latency stretch, in the sense that they have not (yet, to our knowledge) related their latency improvement techniques to the latency properties of the underlying topology. It may well be possible that these heuristics (or others based on Plaxton's work) exhibit good latency stretch on real router-level topologies.

In this paper, we propose a latency improvement technique that has several desirable properties: it is simple, applicable to a class of DHT systems, *incrementally* but rapidly improves their latency performance, and can be theoretically shown to work well on a large class of graphs.

Our technique can be applied to a *class* of DHT systems that uses "geometric" routing (*i.e.*, where the search space for the target reduces by a constant factor after each hop). This class includes Chord, Tapestry, and Pastry, but not CAN. In order to improve the latency in these systems *without* radically altering the routing scheme, the latency for each (or most) of these hops must be improved. Thus, each node must have a pointer to a good (i.e. low-latency) node in each geometrically decreasing "range" of the key-space. One simple, and somewhat optimistic, way for a node to obtain a good pointer for a range is to randomly sample a small number of nodes from this range, measure the latency to each sampled node, and use the one with the smallest latency as the pointer for this range. Surprisingly, this simple scheme gives near-optimum latency stretch for a large class of networks with a small number of samples. Most of this paper is devoted to the description, analysis, and simulation of this random sampling scheme.

We use Chord to illustrate how this idea may be efficiently implemented in a practical DHT system. Each overlay node on the path followed by a lookup request samples its distance to the lookup target (of course, after the lookup has been successfully resolved). In doing so, each node obtains a distance *sample* to one of its key-space ranges, which it

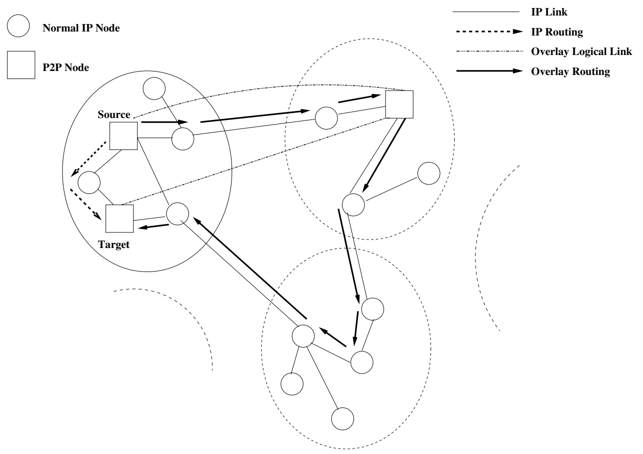

Figure 1: In a DHT system, an overlay path can differ significantly from unicast path.

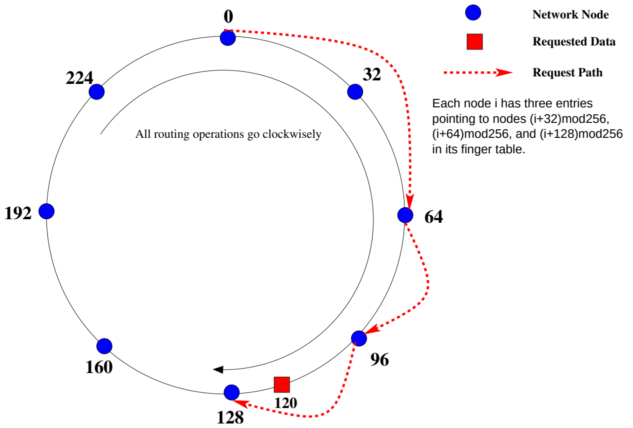

Figure 2: A Chord network with 8 nodes and $m = 8$.

uses to update its routing table. We call this technique *lookup-parasitic random sampling* (LPRS), since it piggybacks on lookup operations in Chord. The Chord routing scheme remains largely unchanged. LPRS *incrementally* obtains low-latency routing tables, unlike other schemes which compute low-latency tables when a node joins the overlay.

Next, we analyze the random sampling scheme for idealizations of DHT systems which organize their key-space as a cycle (like Chord) and those which organize their key-space as a hypercube (like Pastry and Tapestry). We prove that if the underlying physical network has an exponential expansion in terms of latency, then no significant latency improvement is possible without significantly changing the underlying routing schemes. However, if the underlying physical network has a power-law expansion in terms of latency, then we prove that the random sampling scheme improves the average latency from $\Theta(L \log N)$, which these systems achieve without any latency improvement, to $\Theta(L)$. Here, N is the number of nodes in the DHT system, and L is the average latency between pairs of nodes in the underlying physical network. Since L is a lower bound on the average latency, the simple random sampling strategy achieves near-optimum average latency, after a fairly small (poly-logarithmic) number of samples. Furthermore, the first few samples offer most of the latency improvement (Sections 4 and 5.6.1); this indicates that the strategy would work well in the presence of node dynamics.

We perform extensive simulations to evaluate the performance improvement due to LPRS in realistic settings (as opposed to the idealized setting in which we obtain our theoretical guarantees). We modified the SFS Chord simulator [31] to implement LPRS. For simple topologies with power-law-latency expansion (rings and meshes), the latency stretch of LPRS-Chord (Chord augmented with LPRS) is quite small and appears to be independent of network size, whereas the latency stretch of Chord is much larger and appears to grow as the logarithm of the network size. For topologies with exponential latency expansion (PLRG [1] and random graphs [2]), the improvement due to LPRS is not as significant. We also simulate the time-evolution of the latency stretch with LPRS-Chord, and find that relatively few samples are required to bring the overall system to an acceptable performance regime. Also, we find that even with a Zipf-ian document popularity, LPRS-Chord converges quite quickly to a desirable latency stretch.

To determine what latency expansion real-life networks have, we measure the latency expansion of a router-level topology gathered by mapping the Internet [9]. We use geographic distance as an approximation for the propagation latency between two nodes. Our results suggest that the Internet exhibits power-law expansion in terms of latency. Our simulations of Chord and LPRS-Chord on Internet router-level topologies show the same kind of qualitative improvement due to LPRS as on rings and meshes. This indicates that random sampling has significant practical applicability as a general latency reduction technique for many DHT systems.

Our finding about the latency expansion properties of Internet router-level graphs has independent interest. The Internet is widely believed to have an exponential expansion in terms of hop-count [16], and this has influenced the design of topology generators [28]. Future topology generators that incorporate link latency will be impacted by our finding.

The remainder of this paper is organized as follows. Section 2 introduces Chord briefly. In Section 3 we describe LPRS and its implementation in Chord. Section 4 presents theoretical results that relate the performance of random sampling to the underlying topology. In Section 5 we simulate the LPRS scheme for different topologies, and also evaluate the latency expansion of real-life networks. We discuss related work in Section 6, and Section 7 presents our conclusions. Among other things, Section 7 discusses future research directions relating to the *dynamics* of DHT systems, which is an issue we have not addressed in this paper in any depth.

2 Chord

In this section, we briefly describe the Chord [27] DHT system. From our perspective, Chord is a representative of the class of DHT systems that use "geometric" routing. As we discuss in Section 3.5, our random sampling scheme – LPRS – applies more generally to such systems. However, for concreteness, we describe and evaluate LPRS in the context of Chord.

Like all other DHT systems, Chord supports scalable storage and retrieval of arbitrary $< key, data >$ pairs. Data items are stored in a distributed manner among a set of nodes. The Chord protocol constructs and manages an overlay network on top of this set of nodes. The overlay is built to facilitate rapid lookup on a key using a relatively little routing information at each node in the overlay network.

To do this, Chord assigns each overlay node in the network an m-bit identifier (called the node ID). This identifier can be chosen by hashing the node's address using a hash function such as SHA-1. Similarly, each key is also assigned an m-bit identifier (following [27], we use key and identifier interchangeably). Chord uses *consistent hashing* to assign keys to nodes. Each key is assigned to that node in the overlay whose node ID is equal to the key identifier, or follows it in the identifier circle (the circle of numbers from 0 to $2^m - 1$). That node is called the *successor* of the key (*e.g.*, in Figure 2, the successor of key 120 is the node 128, the node following 120 in a *clockwise sense* on the Chord circle). An important consequence of assigning node IDs using a random hash function is that location on the Chord circle has no correlation with the underlying physical topology.

The Chord protocol enables fast, yet scalable, mapping of a key to its assigned node. It maintains at each node a *finger* table having at most m entries. The i-th entry in the table for a node whose ID is n contains the pointer to the first node, s, that succeeds n by at least 2^{i-1} on the ring, where $1 \leq i \leq m$. Node s is called the i-th finger of node n. Figure 2 shows an example of 8-node Chord network with $m = 8$. In this example, node 0's first six finger entries all point to node 32, while the last two point to nodes 64 and 128 respectively. Finger tables for other nodes can be similarly defined.

Suppose node n wishes to lookup the node assigned to a key k (*i.e.*, the successor node x of k). To do this, node n searches its finger table for that node j whose ID immediately precedes k, and passes the lookup request to j. j then recursively[1] repeats the same operation; at each step, the lookup request progressively nears the successor of k. At the

[1] Chord lookups can also traverse the circle iteratively.

end of this sequence, x's predecessor returns x's identity (its IP address) to n, completing the lookup. Because of the way Chord's finger table is constructed, the first hop of the lookup from n covers (at least) half the identifier space (clockwise) between n and k, and each successive hop covers an exponentially decreasing part. In Figure 2, a lookup for key 120 at node 0 hops through nodes 64 and 96 before reaching 128. It is easy to see that the average number of hops for a lookup is $O(\log N)$ in an N-node network. It is also easy to see that the total distance traversed between n and the successor of k on the overlay network may be significantly longer than the unicast distance between those two nodes.

Our brief description of Chord has omitted several details of the Chord protocol, including procedures for constructing finger tables when a node joins, and maintaining finger tables in the face of node dynamics (the Chord *stabilization* procedure). The interested reader is referred to [27] for these details.

3 Lookup-Parasitic Random Sampling in Chord

Routing in several DHT systems such as Chord, Pastry, and Tapestry has an interesting property: the search space for the key decreases by a constant factor (i.e., geometrically) after each hop. In order to improve the latency *without* radically altering the routing scheme, the latency for each (or most) of these hops must be improved. Thus, each node must have a pointer to a good (i.e. low-latency) node in each geometrically decreasing "range" of the key-space. One simple, and somewhat optimistic, way for a node to obtain a good pointer for a range is to randomly sample a small number of nodes from this range, measure the latency to each sampled node, and use the one with the smallest latency as the pointer for this range. Surprisingly, this simple scheme gives almost-optimum latency stretch for a large class of networks with a small number of samples.

In this section we first describe the above ideas in more detail, and then present a particular realization of random sampling in Chord that we call Lookup-Parasitic Random Sampling (LPRS). We finish this section by discussing the application of LPRS to other DHT systems.

3.1 Random Sampling for Latency Improvement

Let N be the number of nodes in the network. Before proceeding, we need to introduce the following three terms:

Range: For a given node in a Chord overlay with ID j (we will henceforth refer to such a node simply as node j), its i-th *range* is the interval of the key space defined as $[j + 2^{i-1}, j + 2^i)$, where $1 \leq i \leq m$.

Latency expansion: Let $N_u(x)$ denote the number of nodes in the network that are within latency x of u. Informally, for $d \geq 1$, a family of graphs has a *d-power-law latency expansion* if $N_u(x)$ grows (i.e. "expands") proportionally to x^d, for all nodes u. Some simple examples of graph families with power-law latency expansion are rings ($d = 1$), lines ($d = 1$), and meshes ($d = 2$). Informally, a family of graphs has *exponential latency expansion* if $N_u(x)$ grows proportionally to α^x for some constant $\alpha > 1$. Formal definitions of power-law and exponential latency expansion are in appendix A.

Sampling: When we say node x *samples* node y, we mean that x measures the latency (*e.g.*, by using ping) to y. Depending on the measured value, x may update its finger table.

Now consider the i-th range of a node n. Suppose there are l nodes whose IDs lie in this range. Because the IDs of the l nodes are chosen randomly, the latency of each of these nodes from n is likely to be random. The key idea behind random sampling is that node n picks a small sample from among the l nodes, and sets as its i-th successor (*i.e.* its i-th finger table entry) that node from the sample which is closest to itself in terms of latency[2].

Whether this scheme succeeds, and if it does, how large the sample needs to be, depends on the latency expansion characteristics of the underlying physical topology. Intuitively, in a network with exponential latency expansion, an overwhelming majority of the nodes will be very far from a node u, and finding the closest node from a small sample is unlikely to significantly improve the latency of the overall scheme. However, in a network with power-law latency expansion, a node u only needs to sample a small number of nodes from each range in order to find a nearby node.

It turns out that it is possible to make these intuitions precise. In a network with d-power-law latency expansion, if each node on an N-node Chord overlay obtains $(\log N)^d$ uniform samples from each of its $\log N$ ranges, the average distance traversed by a lookup is $\Theta(L)$ where L is the average latency between a pair of nodes in the original graph (we prove this formally in Section 4). In a network with exponential latency expansion, no significant improvement can be obtained. For both classes of networks, the original implementation of Chord would result in $\Theta(L \log N)$ average latency.

Two questions then remain:

- How does each node efficiently obtain $(\log N)^d$ samples from each range?
- Do real networks have power-law latency expansion characteristics?

We address the first question next. We discuss the second in Section 5, where we present evidence that suggests that real networks resemble graphs with power-law latency expansion.

3.2 Lookup-Parasitic Random Sampling (LPRS)

Consider a lookup for key D which starts at node s. Let t be the target node for the lookup. Since the mapping of documents to nodes is random, node s can consider t to be a random sample from the largest range (*i.e.*, the entire network). Since the search space decreases geometrically along the request path, successive nodes in the path can consider node t to be a random sample from successively smaller ranges (We make these notions precise in Section 4).

This suggests a simple technique to get fast random sampling in terms of ranges for Chord nodes: when a request completes, each node on the request path samples the target, and updates its finger table accordingly. We call this technique Lookup-Parasitic Random Sampling (LPRS) since all information the random sampling scheme needs is contained in the path that a lookup request traverses. A naive, simpler strategy might have been for just the source to sample the target. Such a scheme is highly unlikely to result in any samples from the smaller ranges, and hence would not lead to optimum latency improvement with few samples.

3.3 LPRS-Chord

We use the term *LPRS-Chord* to denote Chord augmented with LPRS. Three modifications[3] to Chord are required in order to implement LPRS-Chord.

[2]This changes the Chord invariant, in which the i-th successor of a node n is the first node on the Chord circle whose ID is equal or greater than $n + 2^{i-1}$. As we show later (Section 3.4), this does not alter the $O(\log n)$ behavior of the average number of overlay hops for each lookup.

[3]Before considering LPRS-Chord, we wondered if a simple modification to Chord's routing could give us significant latency improvements. Specifically, we simulated the following change to Chord's routing algorithm: pick the nearest (in terms of latency on the underlying topology) successor in the finger table that reduces the distance in ID-space to key k by at least $1/4$. Essentially, this technique greedily picks a low-latency successor "far" enough away, and is a simple proximity routing scheme. Our simulations show that this approach does not perform significantly better than Chord–it reduces the physical distance traversed per hop, but increases the number of hops on the overlay.

First, LPRS-Chord needs additional information to be carried in each lookup message. Specifically, each intermediate hop appends its IP address to a lookup message. This is a reasonable implementation choice since each request takes $O(\log N)$ hops on the Chord overlay (and our modifications don't break this property, see below). When the lookup reaches its target, the target informs each listed hop of its identity. Each intermediate hop then sends one (or a small number) of pings to get a reasonable estimate of the latency to the target. It is well known that latency estimation using pings is not perfect (*e.g.*, due to routing asymmetry), but this methodology should suffice for us, since we are not interested in making fine-grained latency distinctions. For example, it is less important for an LPRS-Chord node to correctly pick the closer of two potential successors the latency to whom is comparable (*e.g.*, both are topologically equidistant from the node), than to pick the closer of the two potential successors the latency to whom differs significantly (*e.g.*, because one is on the East Coast and the other on the West Coast).

Second, LPRS-Chord requires the use of recursive lookups, rather than iterative lookups. Each node incrementally improves its own finger table for low latency hops to successors. Consider a lookup from source s to target t traversing through node i. Suppose i's successor towards t is j. An iterative lookup would have s contact i, obtain the identity of j, and proceed thus. This can invalidate our latency optimizations, since the latency from s to j is clearly not optimized by LPRS-Chord. Only i picks j as its successor for the appropriate range with a view to reducing latency. For this reason, LPRS-Chord requires recursive lookups (where i passes on the request to j, and j continues the lookup recursively).

Finally, what remains to be defined is how LPRS-Chord updates its finger table entries based on the samples obtained. The following algorithm describes how a node n updates its finger table when it has a distance estimate for a Chord node p. This simplified description assumes, however, that each node n *always maintains its predecessor and successor* in the Chord ring. This information is needed to enable the Chord join, leave, and stabilization algorithms to continue to work correctly. These algorithms do not require any modifications in LPRS-Chord [4].

1. A node n maintains one finger table entry for each of its m ranges.
2. For node p, n finds the range k it lies in.
3. If the p is closer than the current k-th successor for node n, make p the new k-th successor of n (In practice, an implementation might maintain several "candidate" k-successors for fast fail-over. In this case, LPRS would replace the furthest candidate successor.)

Figure 3 shows the pseudocode for some of LPRS-Chord's procedures. This omits several details of the implementation, such as the details in `evict_finger` about preserving the correct immediate predecessor and successor entries.

3.4 Discussion

There are three issues that need to be discussed with regard to LPRS-Chord.

The first is that LPRS-Chord breaks Chord's invariant in constructing finger table entries. Chord's invariant is that in any node n's finger table, the i-th successor is the first node whose ID is equal to or larger than $n+2^{i-1}$. In LPRS-Chord, this invariant is no longer true. The i-th finger table entry points to *some* "nearby" node whose ID lies in the interval $[n+2^{i-1}, n+2^{i})$.

However, the following theorem holds for LPRS-Chord:

[4] In dynamic scenarios, the stabilization algorithm needs modification due to the change of Chord invariants. Instead of refreshing the first node in each range, the stabilization algorithm in LPRS-Chord should refresh the known nearest node in each range.

```
//node n processes a lookup request r for the target key r− >x

n.processRequest(request r)
        //if n is the immediate predecessor of r− >x
        if( between(r− >x, n− >id, getSuccessor(n)) || r− >x == getSuccessor(n) )
        ..... . . . . // normal Chord operations
        ....//uni-directional sampling to the destination node
        ....s = getNode(getSuccessor(n)); //s is the destination node
        ....m = getNode(r− >initiator); //m is the source node
        ....m.evict_finger(s);
        ....for(i=0; i< r− >stack.num; i++)
        ........m = getNode(r− >stack.nodes[i]); //m is the (i + 1)th hop in the request path
        ........m.evict_finger(s);
        ....return; // request resolved !

        //Otherwise, n is an intermediate node in the request path
        //push node ID into the path node stack
        r− >stack.nodes[r− >stack.num++] = n− >id;
        . . . . . . // normal Chord operations


n.evict_finger(node d)
        //find the range k d belongs to among all of n's ranges
        k = find_Range(n, d);
        finger_to_evict = NULL;
        if( !(finger_to_evict = farthest(n− >finger_list, k)) )
        ....//if no finger for range k in current finger list
        ....finger_to_evict = farthest(n− >finger_list, ALL_RANGES));
        ....delete_finger(n− >finger_list, finger_to_evict);
        ....insert_finger(n− >finger_list, d);
        else //we will evict the farthest finger among all fingers for range k
        ....if( latency(n, finger_to_evict) ≥ latency(n, d) )
        ........delete_finger(n− >finger_list, finger_to_evict);
        ........insert_finger(n− >finger_list, d);
        return;
```

Figure 3: Pseudocode for LPRS-Chord lookups and finger-table management

Theorem 1 *LPRS-Chord resolves a lookup in $O(\log N)$ hops on an N-node overlay network.*

Proof: Suppose that node n is the origin of a lookup request for key k, and that k resides in n's i-th range. Let t be the target of the lookup for k. In Chord, the first hop from n towards k would have been the Chord node with the lowest ID among all Chord nodes in $[n + 2^{i-1}, n + 2^i)$ (*i.e.*, Chord's i-th successor, assuming there is at least one node in that range). In LPRS-Chord, that hop may not be feasible, since the finger entry for the i-th range may point to a node that has an ID greater than t. In that case, the next hop for LPRS-Chord would be some node in the $(i-1)$th range of node n; this hop would cover at least one-fourth of the distance to t in the key space. Recursively, each hop covers at least one-fourth of the remaining distance in the key-space towards t, resulting in $O(\log N)$ hops.
■

In practice, as our simulations show (Section 5), LPRS-Chord's average number of overlay hops is comparable to or better than Chord's.

A second issue is the particular choice of sampling strategy in LPRS-Chord. Although we have described the strategy in which each node on the lookup path samples the target, there are many other *conceivable* sampling strategies, some of which may be easier to implement in practice. Consider a lookup that is initiated by node x_0, then forwarded to node x_1, x_2, ..., and finally reaches the request terminator, node x_n. The following schemes suggest themselves:

1. Node x_i samples node x_n, $0 \leq i < n$
2. Node x_n samples nodes x_0, ..., x_{n-1}
3. Node x_i samples node x_{i-1}, $1 \leq i \leq n$
4. Node x_0 samples node x_n
5. Node x_i samples node x_0, $1 \leq i \leq n$

Of these, the first scheme is the one that LPRS-Chord uses. Schemes 4 and 5 result in most of the samples being from the largest range and hence, do not result in optimum latency improvement. Schemes 2 and 3 are interesting. Since only uni-directional routing is allowed on the Chord ring, these two schemes suffer from the same defects as schemes 4 and 5 for Chord, and should result in sub-optimal improvement. Pastry and Tapestry, however, use bi-directional routing. Hence, these schemes may perform well for Pastry and Tapestry, and deserve further exploration. We do a comparison of these schemes in the context of Chord in appendix B and the simulation results support our intuition.

Finally, we note that a practical implementation may choose not to sample on every request, trading off the time to converge to a "good" latency for reduced sampling overhead. There are several ways to do this. For example, an adaptive approach might work as follows: A node initially samples the target of every request, but when it notices its latency estimates to a particular range not improving significantly, it reduces the sampling rate for that range. We do not discuss these implementation details in this paper.

3.5 Generalizations

Revisiting Section 3.1, we see that the general property of Chord that random sampling leverages is that the Chord routing table leads to geometric routing (defined more formally in the next section). The random sampling strategy should more generally apply to any DHT system that uses geometric routing. Indeed, geometric routing forms the basis of our proofs about the random sampling property (Section 4).

We observe that *both Tapestry and Pastry use geometric routing*. In both of these schemes, a lookup for a key is routed as follows: the lookup initiator finds the neighbor whose ID's most significant bit matches that of the key. This neighbor looks at the next bit, and finds that neighbor whose ID matches the corresponding bit in the key, and so on. The details of the schemes vary slightly; in Tapestry, for example, the comparison is not bit-by-bit, but digit-by-digit. In principle, however, this kind of forwarding results in a logarithmic number of hops in the key space.

Thus, LPRS can be retrofitted into both Pastry and Tapestry quite easily (Pastry and Tapestry implement a simpler heuristic for latency reduction, which we discuss in Section 6). There is one difference in detail, however, between Chord and these two systems. In Chord, lookups traverse the key space in one direction around the Chord circle. Lookups in these other two DHT systems can traverse the key space in either direction. Thus, these systems can sample in both directions (schemes 2 and 3 in Section 3.4).

Interestingly, CAN's routing tables do not satisfy the geometric property. CAN uses a d-dimensional Cartesian coordinate space and each node only maintains the pointers to its $2d$ neighbors. Thus, each hop on the CAN overlay traverses an approximately uniform distance in the key space. For this reason, LPRS does not apply to CAN, which needs to use a different strategy to effect latency reduction (Section 6).

4 Analysis

In this section, we formally analyze the performance of random sampling for routing schemes such as Chord, Pastry, and Tapestry. We model the underlying network as an undirected graph $G = (V, E)$ with $N = |V|$ nodes and a latency function l defined on the set of links. For node $v \in V$, $\textsc{Id}(v)$ denotes the location of v in the virtual name-space. For document D, we will assume that $h(D)$ denotes the location, in the virtual name-space, of the node where D is stored. We extend the latency function to all pairs of nodes by defining the latency between a pair of nodes to be the smallest latency along a path between the nodes. For ease of exposition we will assume that $N = 2^k$. We will study two kinds of systems:

Hypercube systems: Here, the virtual name-space is structured as a k-dimensional hypercube H_k. Each point in this space can be represented as a k-bit number. We will assume that $\textsc{Id} : V(G) \to \{0, 1, \ldots, 2^k - 1\}$ is a one-one, onto function chosen uniformly at random from the space of all such functions. This is an idealization of systems such as Pastry and Tapestry.

For two points $x, y \in H_k$, their bit-difference $\textsc{Diff}(x, y)$ is the most significant bit-position in which they are different. For example, $\textsc{Diff}(x, x) = 0$, $\textsc{Diff}(0, 1) = 1$, and $\textsc{Diff}(10110, 10011) = 3$. An i-range of node u, denoted $R_i(u)$, is the set of all nodes v such that $\textsc{Diff}(\textsc{Id}(u), \textsc{Id}(v)) = i$.

Linear Systems: Here, the virtual name space is structured as a a cycle of length N. We will assume that $\textsc{Id} : V(G) \to \{1, 2, \ldots, N\}$ is a one-one, onto function chosen uniformly at random from the space of all such functions. This is an idealization of systems such as Chord.

The difference from point x to point y in the virtual name-space, denoted $\textsc{Dist}(x, y)$ is $(y + N - x) \bmod N$. An i-range of node u, denoted $R_i(u)$, is the set of all nodes v such that $2^{i-1} \leq \textsc{Dist}(\textsc{Id}(u), \textsc{Id}(v)) < 2^i$.

Geometric routes: A route from u to v is geometric if it is of the form $\langle w_i, w_{i-1}, \ldots, w_1, w_0 \rangle$, where

1. $w_i = u$, $w_0 = v$, and
2. for any $0 < j \leq i$, $\textsc{Diff}(\textsc{Id}(w_0), \textsc{Id}(w_{j-1})) < \textsc{Diff}(\textsc{Id}(w_0), \textsc{Id}(w_j))$ if the virtual name-space is a hypercube, and $\textsc{Dist}(\textsc{Id}(w_{j-1}), \textsc{Id}(w_0)) \leq \textsc{Dist}(\textsc{Id}(w_j), \textsc{Id}(w_0))/c$ for some constant $c > 1$, if the virtual name space is linear.

The intuition behind this definition is quite simple. If the destination node lies in the i-range of a node w, then it lies in the $i-1$-th or smaller range of the node which occurs just after w in a geometric route. Since the size of an i-range is exponential in i, the definition of geometric routes ensures that the "search-space" for the destination node decreases at least geometrically, i.e., by at least a constant factor, after each successive hop.

Frugal routing: A DHT routing scheme is said to be *frugal* if it

1. uses only geometric routes, and

2. if w is an intermediate node in the route, v is the destination, and $v \in R_i(w)$, then the node after w in the route depends only on w and i.

Note that when a request is issued for locating node v, all that is known to the source and any intermediate nodes is $\text{ID}(v)$. Still, this is sufficient for a source or intermediate node w to determine $\text{DIFF}(w, v)$ or $\text{DIST}(w, v)$ and hence determine the value i such that $v \in R_i(w)$. In this section, we will only study DHT routing schemes that are frugal. This includes Chord, Pastry, and Tapestry.

Let $\Delta = \max_{x \in V, y \in V} l(u, v)$ denote the diameter of graph G. Let $N_u(x)$ denote the set of all nodes $v \neq u, v \in V$ such that $l(u, v) \leq x$.

4.1 Bounds on latency of frugal schemes

We will study the latency of a lookup request made by an arbitrary node to a document chosen uniformly at random. Recall that $\text{ID}()$ is a random function, and hence making a request to a random document is the same as making a request to a random node. Further, there is no relation between a node's position in the physical topology and its position in the virtual overlay topology.

Theorem 2 *For any frugal DHT scheme, the maximum latency of a request is $O(\Delta \log N)$.*

Proof: Any geometric route can have at most $O(\log N)$ hops, and the latency of each hop can be at most Δ. ■

Theorem 3 *If G is drawn from a family of graphs with exponential latency expansion, then the expected latency of any frugal DHT scheme is $\Omega(\Delta \log N)$.*

The proof of the above theorem is technical, and is deferred to appendix A. Theorems 2 and 3 together imply that if a frugal DHT scheme is intended to run over graphs with exponential latency expansion, then any optimizations we perform can only yield a constant factor improvement. In contrast, if G is drawn from a family of graphs with d-power-law latency expansion, then there exist frugal DHT routing schemes which have expected latency $O(\Delta)$. We will now present one such scheme, based on simple random sampling, and analyze the resulting latency.

4.2 The Random Sampling Algorithm

We will present and analyze our algorithm only for the case when the virtual space is organized as a hypercube. When the virtual space is linear, the algorithm needs to take into account the fact that the routing becomes "uni-directional". However, the same analysis holds with minor technical differences; a detailed discussion is omitted for ease of exposition.

The random sampling algorithm is described in Figure 4. The parameter J, the number of random samples to be taken from each range, is given as an input to the algorithm. The routing scheme in Figure 4 is clearly frugal. The following theorem analyzes the expected latency.

Theorem 4 *If we follow the random sampling algorithm above, and G is drawn from a family of graphs with d-power-law latency expansion, then the expected latency of a request is $O(\Delta) + O(\Delta \log N / J^{1/d})$.*

Proof: Let $u \in V$ be an arbitrary node. We will overload notation to use $l(u, S)$ to denote $\min_{v \in S} l(u, v)$. During the initialization phase, node u obtains J samples from the set $R_i(u)$, but they may not all be distinct. Let $S_i(u)$ represent

Initializing the Routing Tables:

For each node $u \in V$ and each $0 < i <= \log N$,

1. Pick J nodes from $R_i(u)$ uniformly at random, with replacement.
2. Measure the latency from u to each of these J nodes.
3. Let $v_i(u)$ denote the node with minimum latency (among those picked in step 1) from u.

The Routing Phase:

Suppose u receives/generates a lookup request for document D.

1. If u is the destination node (i.e. u is the node in charge of storing document D) then u services the request.
2. Else, determine i such that $h(D) \in R_i(u)$. Node u forwards the request to $v_i(u)$.

Figure 4: **Random Sampling Algorithm**$(G, \textsc{Id}, J)$

the set of distinct samples from $R_i(u)$ obtained by u, and let $l_i(u)$ represent the latency $l(u, S_i(u))$. Since the function $\textsc{Id}()$ is random, $S_i(u)$ is chosen uniformly at random from all subsets of V of size $|S_i(v)|$.

We will now need the following two facts; the proofs are involved and are deferred to appendix A.

Claim 4.1 *Let S be a subset of V of size p chosen uniformly at random from all such subsets. Then* $\mathbf{E}[l(u, S)] = O(\Delta/|S|^{1/d})$.

Claim 4.2 $\mathbf{E}[1/|S_i(u)|^{1/d}] = O((1/\min\{J, 2^i\})^{1/d})$.

The first claim asserts that the distance from u to the "closest" node in S varies as $|S|^{-1/d}$; this claim depends crucially on the fact that G has a power-law latency expansion.

From claims 4.1 and 4.2, we can conclude that $\mathbf{E}[l_i(u)] = O(\Delta/(\min\{J, 2^i\})^{1/d})$. Using linearity of expectations, and the fact that u was arbitrary, we obtain the following bound on the expected latency, L_G, of a geometric route:

$$\begin{aligned} L_G &\leq O\left(\Delta \sum_{i=1}^{\log N} 1/(\min\{J, 2^i\})^{1/d}\right) \\ &= O\left(\Delta \sum_{1 \leq i \leq \log J} (2^{1/d})^{-i}\right) + O\left(\Delta \sum_{\log J < i \leq \log N} 1/J^{1/d}\right) \end{aligned}$$

The first summation above is the sum of a geometric series, and can be upper-bounded by $O(\Delta)$ since d is a constant. the second summation has at most $\log N$ terms, and hence can be upper-bounded by $O(\Delta \log N / J^{1/d})$. ■

4.3 Discussion

While the above analysis is involved, it conveys a simple message: if the underlying network has d-power-law latency expansion, then for $J = \log^d N$, a simple random sampling scheme results in $O(\Delta)$ expected latency. For example, if $N = 1,000,000$ and $d = 2$, each node need only sample 0.8% of the nodes for the network to achieve asymptotically optimal latency performance. This is in stark contrast to theorem 3 for networks with exponential latency expansion, and motivates the question of whether real-life networks demonstrate power-law or exponential latency expansion; this question is addressed in section 5.

The following additional observations are also worth making:

1. The proof of theorem 4 would not go through if the random sampling algorithm simply obtained $J \log N$ random samples from V as opposed to getting J samples from each of the $\log N$ ranges. Intuitively, for geometric routing to result in good performance, we need to do "geometric sampling". This issue is further explored in appendix B.

2. The above algorithm and the analysis do not assume anything about the *mechanism* for obtaining the samples. This gives us significant flexibility: the samples can be obtained either upon network creation/node joining or by piggybacking over normal DHT functionality, as in section 3.

3. Let $X(J) = (\Delta \log N)/J^{1/d}$. Then
$$X(J-1) - X(J) \geq \frac{\Delta \log N}{dJ^{1+1/d}}.$$
Thus, the improvement in the guarantee offered by theorem 4 is most significant for the first few samples. This is supported by our simulations (Section 5.6.1).

4. Let L denote the average latency between pairs of nodes in the network. For graphs with d-power-law latency expansion as well as for graphs with exponential latency expansion, $L = \Theta(\Delta)$. Therefore, all the theorems in this section could have been stated in terms of L as opposed to Δ.

5 Simulation Results

In this section, we evaluate the performance of LPRS-Chord through simulation. The goal of this evaluation is to validate the analytical results of Section 4. That analysis assumed idealized models of Chord (and Pastry and Tapestry, for that matter), and our validation ensures that those models have not omitted important details of the Chord protocol. Furthermore, our simulations help quantify the performance of LPRS-Chord vis-a-vis unmodified Chord; our analysis only describes the asymptotic behavior of these two systems.

We also address one important issue that determines the practical applicability of this work: What are the expansion characteristics of real router-level topologies?

5.1 Simulation Framework

We modified the SFS Chord simulator [31] to implement LPRS. The Chord simulator is event-driven and simulates insertion and deletion of "documents" into a Chord network. It also simulates Chord protocol actions during node joins and leaves, and the Chord stabilization actions. In the Chord simulator, the default value of m is 24 and the default routing table size is 40.

Our modifications to the Chord simulator are essentially as described in Section 3.3. First, we changed Chord's finger table maintenance algorithms to implement random sampling. Next, we modified each document retrieval request (*i.e.*, the `find_doc` request in Chord simulator) to use recursive lookups (the default mode in the simulator is to use iterative lookups). Finally, we attached the addresses of each intermediate hop to the request message. We then modified the `find_doc` procedure to return to each intermediate node the target's address, so that each node can implement the sampling algorithms described in Section 3.2.

5.2 Simulation Methodology

In all our experiments, the input to the simulator is a network topology of size N. This topology represents the underlying physical connectivity between nodes (except when we simulate Chord over an Internet router-level topology; we discuss our methodology for this in Section 5.5). For simplicity, we assume that *all* nodes in this topology participate in the Chord overlay.

Given a network topology, a simulation run for LPRS-Chord consists of three distinct phases.

In the first phase, N nodes join the network one-by-one. During this phase, we use the Chord join algorithms to build the finger tables. At the end of this phase, then, the finger tables are exactly what unmodified Chord would have built (*i.e.*, they maintain the Chord invariant). This allows us to measure how much of a latency improvement LPRS-Chord obtains over unmodified Chord.

In the second phase, each node on average inserts four documents into the network. In principle, during this phase we could have enabled the sampling and finger table replacement in LPRS-Chord, since document insertion involves a lookup request. However, for the ease of understanding our results, we chose to enable these sampling actions only in the next phase.

In the third phase, each node generates, on average $3 \log N$ `find_doc` requests. One request is generated on each simulation clock tick. In most of our simulations, the targets for these requests are chosen uniformly. However, we also discuss the impact of a Zipf-ian document popularity on LPRS-Chord.

5.3 Performance Metrics

The focus of our paper has been on improving the lookup latency in Chord and other DHT systems. Our primary metric, then, is:

average latency stretch: (henceforth, just stretch) the ratio of the average latency for each lookup to the average latency between each pair of nodes on the underlying topology.

In some places, we discuss two related metrics:

hops: the average number of hops that each request takes on the overlay, and

hop-reach: the average latency on the underlying network incurred by each hop on the overlay network.

We introduce these two metrics in order to dissect LPRS-Chord and to more clearly understand its contributions to latency improvements.

5.4 Impact of Topology

Our first set of simulations investigates the performance of LPRS-Chord for different underlying physical topologies. We do this because Section 4 indicates that the performance of LPRS-Chord depends crucially on the topology.

Specifically, we simulate LPRS-Chord and unmodified Chord over the ring, mesh, random graph, and power-law random graph topologies, for a variety of network sizes ranging from 100 to 6400. For each network size, we compute the stretch, hops and hop-reach metrics. For LPRS-Chord we compute these metrics when each node has performed $2 \log N$ lookup requests in the third phase of the simulation run. Our simulations are quite time consuming, and we have only been able to conduct 3 simulation runs for each topology size. However, we note that the computed stretch from these three runs varies very little. We believe that more statistically robust experiments will not invalidate our conclusions.

5.4.1 Topologies with Power-law Expansion

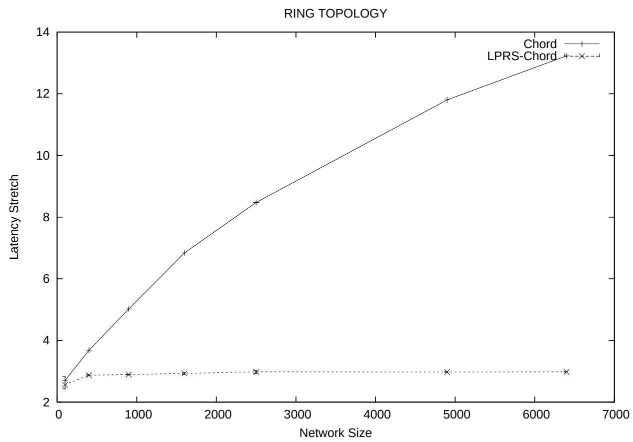

Figure 5: Ring Stretch

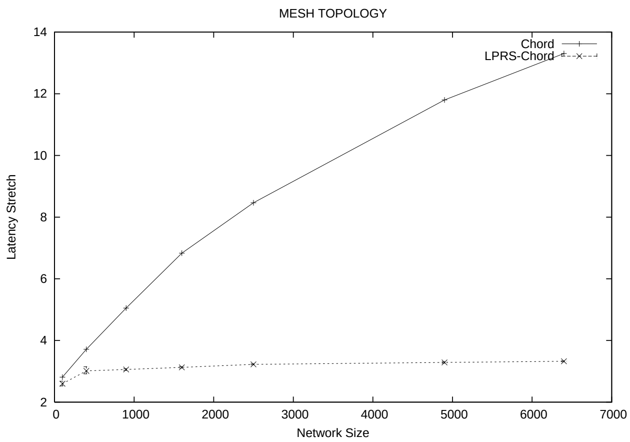

Figure 6: Mesh Stretch

In this first set of experiments, we consider the ring and the mesh (or grid) topologies. In these topologies, we assume that each link has unit latency. These topologies are known to have power-law expansion, with $d = 1$ for the ring, and $d = 2$ for the mesh.

Figures 5 and 6 plot the stretch of the ring and the mesh as a function of topology size. To understand these graphs, recall that the analysis of Section 4 predicts that the latency in LPRS-Chord is proportional to the average path length. This corresponds to a latency stretch that is independent of network size. Indeed, that is what we observe in these two graphs. For both the ring and the mesh, the stretch is close to three across the range of topologies we consider. Note that LPRS is *qualitatively* different from unmodified Chord. As predicted by our analysis, the stretch of unmodified Chord increases logarithmically with topology size, and is independent of the underlying topologies.

There are two ways to improve stretch: reducing the number of hops a lookup traverses on the overlay, and reducing the physical distance traversed per overlay hop. To better understand where LPRS-Chord's performance improvements are coming from, we examine our two other metrics for the ring (the results for the mesh are qualitatively similar, and therefore omitted) topology: hops and hop-reach.

Figures 7 and 8 plot these two quantities for LPRS-Chord and unmodified Chord. LPRS-Chord explicitly targets the reduction in hop-reach, by trying to find physically nearby successors. It is therefore not surprising that LPRS-Chord's hop-reach is significantly lower than that of original Chord.

What is interesting is that LPRS-Chord noticeably improves upon Chord's hops performance. LPRS-Chord does not explicitly target this in its design. In fact, LPRS-Chord breaks Chord's invariant that the finger for the i-th range is the first node whose ID is equal to or larger than $n + 2^{i-1}$. This invariant is sufficient (but not necessary – see Theorem 1 for example) to ensure Chord's $O(\log N)$ performance bound. In Section 3.4, we claim even with this design change LPRS-Chord can resolve lookups using $O(\log N)$ hops on the overlay. Figure 7 validates this claim. In addition, it shows

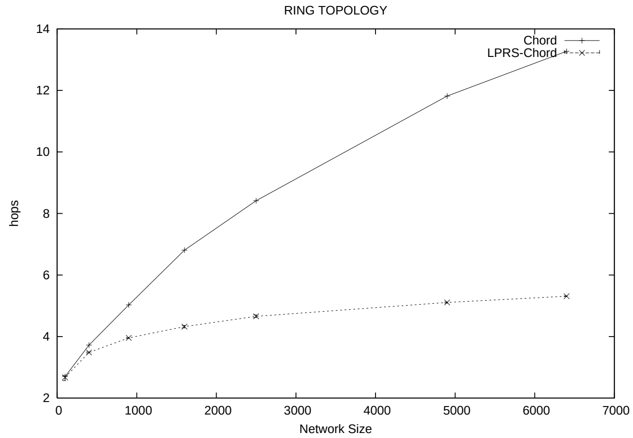

Figure 7: Ring Hops

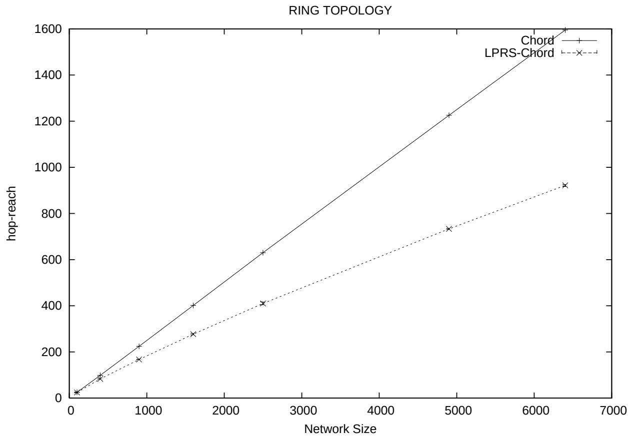

Figure 8: Ring Reach

that LPRS-Chord actually does *quantitatively better* (*i.e.* actually takes fewer hops on the overlay). [5]

5.4.2 Topologies with Exponential Expansion

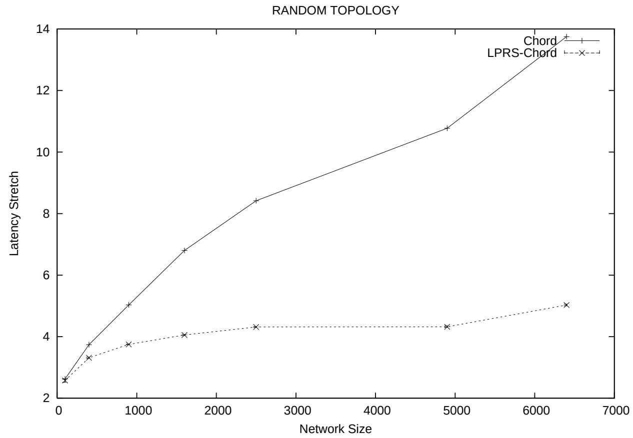

Figure 9: Random Stretch

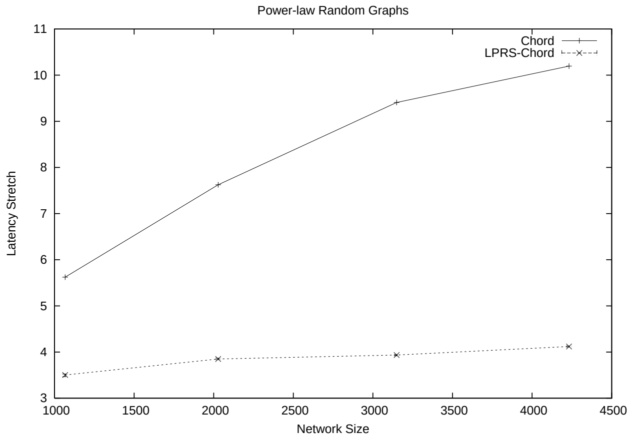

Figure 10: PLRG Stretch

Our second set of experiments simulates Chord over two families of topologies known to have exponential expansion: the classical random graph [2], and the power-law random graph (PLRG) [1]. In these topologies as well, each link has unit latency. In our random graphs, each node has a uniform probability $\frac{\log N}{N}$ of connecting to another node. Our power-law random graphs were generated with a β parameter of 0.7.

Figures 9 and 10 describe the stretch of LPRS-Chord and unmodified Chord as a function of topology size. Notice that even for these topologies, LPRS-Chord has at least a factor of two smaller latency stretch compared to unmodified Chord. However, in these topologies, the performance improvement is markedly less dramatic; as topology size increases, the stretch of LPRS-Chord also increases. In fact, we have verified that this performance improvement comes almost entirely

[5] It might be tempting to assume that if Chord could be modified to focus on hop-count reduction, it would exhibit latency stretch comparable to LPRS-Chord. However, since Chord does not make any routing decisions on the basis of latency, each of the $\Theta(\log N)$ hops must still be of expected length $\Theta(L)$, resulting in a latency of $\Theta(L \log N)$.

from LPRS-Chord using fewer overlay hops, and *not* from the reduction in hop-reach derived by random sampling. We omit those graphs for brevity.

These results also validate another analytical result from Section 4. We argued there that in graphs with exponential expansion, random sampling is not likely to result in orders-of-magnitude latency improvement.

5.5 LPRS-Chord and the Internet router-level topology

Our analysis and simulations suggest that on graphs with power-law expansion, LPRS-Chord exhibits a qualitatively different stretch scaling from that of unmodified Chord. On graphs with exponential expansion, however, LPRS-Chord provides some, but not as dramatic, improvement. The applicability of our result to real-world DHT systems, then, seems to rest on the following question: what are the latency expansion characteristics of the Internet router-level topology?

The literature on topology modeling and characterization has generally assumed that real Internet router-level topologies have an exponential expansion [16][6]. In fact, recent work [28] has shown that the PLRG is a reasonable model for the Internet router-level topology, particularly when attempting to evaluate large-scale metrics.

However, our simulations show that LPRS-Chord's performance over the PLRG topology is good, but does not represent a compelling improvement over unmodified Chord in terms of hop-reach. Does this imply that LPRS is only marginally useful? No. Previous studies have *defined expansion in terms of router-level hops*. We conjecture that *when expansion is defined in terms of latency* (as in Section 3), the Internet router-level topology exhibits *power-law expansion*.

We used a large router-level topology dataset to validate this conjecture. This dataset was gathered using a methodology similar to prior work on topology discovery: traceroutes from several locations (in this case, six nodes on the NIMI infrastructure) to random destinations chosen from BGP routing tables. Router interfaces were disambiguated using techniques described in [9].

How do we measure *latency* on an Internet router-level topology? We argue that, for the purpose of improving the performance of DHT systems, the propagation latency between two nodes is the component that any latency improvement scheme should attempt to adapt to. The expansion characteristic of the Internet router-level topology that we are interested in, then, is the propagation latency expansion.

To measure the propagation latency expansion of the Internet router-level topology, we make the following simplifying assumption: the propagation latency between any two nodes on the router-level topology is well approximated by the geographic distance between those two nodes. This particular assumption has not been validated by any measurement study. There has been at least one measurement study that suggests that geographic distance between hosts correlates well with the propagation latency between them [15]. However, it is generally believable that the propagation latency of a link on the router-level topology can be well estimated by the geographic distance between the ends of the link. The one exception to this is the case when a link in the router-level topology corresponds to a link-layer tunnel or circuit that is actually routed through one or more geographic locations before reaching the end of the link. At the very least, the geographic distance can be used to establish a lower bound on the propagation latency for such a link.

To assign geographic location to nodes in our router-level topology, we used the Geotrack tool [15]. This tool uses heuristics based on DNS names to infer node locations. In many cases, Geotrack cannot assign locations to nodes. In these cases, a node was assigned the location of the topologically nearest node for which Geotrack could assign a location (if there was more than one such neighbor, we randomly picked one).

[6] There is some debate about this. Earlier work, based on a smaller dataset had claimed that the expansion of router-level topologies better matched a power-law [8].

Having assigned geo-locations to all nodes, we could compute the latency expansion properties of the Internet router-level graph. However, our router-level topology contained upwards of 320,000 nodes that were mapped to 603 distinct cities. To tractably compute the latency expansion of this rather large topology we randomly sampled 92,824 node pairs and computed the geographic distance between them.[7]

Before we present our results, we mention that our procedure is approximate for several reasons. The incompleteness of topology discovery and interface disambiguation (or alias resolution) methods is well documented [9] [25]. Geo-location techniques such as Geotrack are also approximate; in particular, our approximation of placing un-resolvable nodes near resolvable ones can underestimate the actual geographic distance between two nodes. Despite this, we believe our general conclusions from this experiment will hold up to scrutiny, since we have been careful to make qualitative judgments from a very large dataset.

Figure 11 plots the latency expansion of the router-level graph. It includes, for calibration, the hop-count expansions of the ring and PLRG topologies, as well as the hop-count expansion of the router-level graph itself. We immediately make the following observations. The hop-count expansion of the Internet router-level graph resembles that of the PLRG and can be said to be exponential. However, the *latency* expansion of the Internet router-level graph is *significantly more gradual* than its hop-count counterpart and more closely matches the ring topology.

We believe, then, that the Internet latency expansion more closely resembles a power-law. That is, the router-level graph looks more "mesh"-like when the distances of links are considered.

This would argue that LPRS-Chord is a very desirable latency improvement scheme for wide-area DHT infrastructures. In fact, we have verified this by simulating LPRS-Chord on subgraphs of the router-level topology. We sampled several subgraphs of different sizes. Nodes in these subgraphs consist of "edge" nodes in the router-level graphs (arbitrarily defined as those nodes with degree one or two). Each pair of nodes on these subgraphs is connected by a link whose latency is the geographic distance between those two nodes in the underlying topology.

On each subgraph, we ran both Chord and LPRS-Chord, assuming that each node on the subgraph was part of the Chord overlay. Figure 12 depicts the stretch for the two schemes as a function of network size. We note the striking similarity of this graph and those of Figures 5 and 6. In particular, the stretch of LPRS-Chord is essentially independent of network size, and below three, whereas that of unmodified Chord increases with network size.

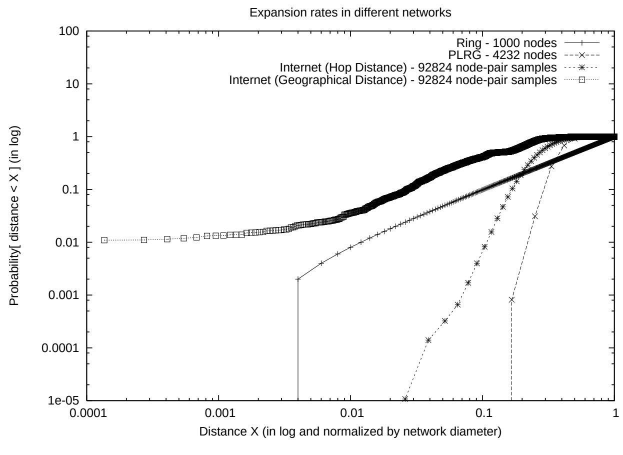

Figure 11: Latency expansion

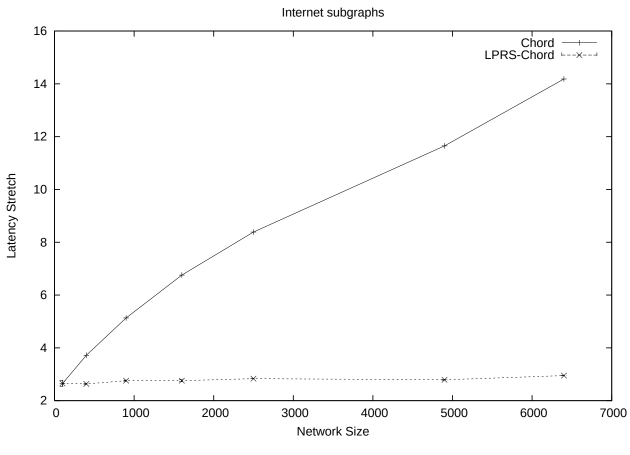

Figure 12: Stretch on the router-level graph

[7] We also attempted other sampling techniques: one-to-all distance measurement for up to 1000 nodes, and full connection distance measurement for large subgraphs up to 6400 nodes. All the expansion curves converged to one another as the number of sampled nodes increased.

5.6 Other Issues

This section addresses two aspects of LPRS-Chord's performance that we have not validated or explored thus far. First, LPRS-Chord incrementally improves latency using lookups; we discuss the convergence time properties of LPRS-Chord. Second, our analysis of LPRS-Chord has assumed a uniform distribution of requests to targets. In practice, targets are likely to be Zipf-ian, and we explore the implications of this for LPRS-Chord through simulation.

5.6.1 Convergence Time

How many lookups does it take before LPRS-Chord converges to a low stretch value on graphs with power-law expansion? To understand this, Figure 13 plots the performance evolution of LPRS-Chord over time for network sizes 900 and 6400 on the ring topology (for the mesh and the Internet graph, the results are similar and omitted). In these graphs, the x-axis describes the number of `find_doc` requests generated per node and normalized by $\log N$ (10 for 900 nodes and 13 for 6400 nodes). We calculated average latency stretch every N requests.

This figure validates our contention in Section 4 that LPRS-Chord is *fast*. When each node has generated on average $3 \log N$ samples (about 39 requests in the 6400 node case), the network has converged to its eventual stretch value of slightly under three. Furthermore, even with a few samples, the latency stretch improvements are quite dramatic. After $\log N$ samples per node the latency stretch is within 30% of the final stretch, and after $2 \log N$ samples the improvement is within 10%.

This bears out our analysis in Section 4: relatively few samples are needed for latency improvements, and the improvement from the initial samples is quite dramatic.

In our simulations, we have not explicitly modeled node dynamics (joins and leaves). However, Figure 13 gives us a handle on the behavior of LPRS-Chord in the face of dynamics. For systems in which a significant fraction of nodes join and leave the system before each node has had a chance to generate $O(\log N)$ requests, clearly LPRS-Chord's performance will be similar to that of unmodified Chord. Quantifying this fraction (and more generally understanding how LPRS-Chord performance degrades with increasing instability) is the subject of future work. Since node-dynamics is less of an issue with DHT systems used as an *infrastructure*, we expect LPRS-Chord to work well for this class of DHT systems.

5.6.2 Impact of skewed request distributions

Our analysis has assumed that targets are uniformly accessed. In practice, the popularity of documents stored in DHT systems is likely to be Zipf-ian (as it is, for example, for Web access patterns [3]). How, then, are our results affected by a different pattern of access to documents?

We simulated LPRS-Chord on a 1600 node ring network. Unlike the previous simulations, the targets of each node's `find_doc` requests are chosen from a Zipf distribution. Each node generates one request every clock tick. Figure 14 plots the evolution of stretch as a function of simulation time, for $3 \log N$ (=32) generated requests. We compute the stretch at every clock tick. (The evolution of stretch for other topologies is qualitatively similar, and we omit these results for brevity).

We find that even with a Zipf-ian popularity, LPRS-Chord eventually converges to a very desirable stretch. After 32 requests per node, the stretch resulting from Zipf-ian access to documents is less than 10% higher than the stretch resulting from a uniform access to documents. This is easily explained; as long as each node gets enough samples from

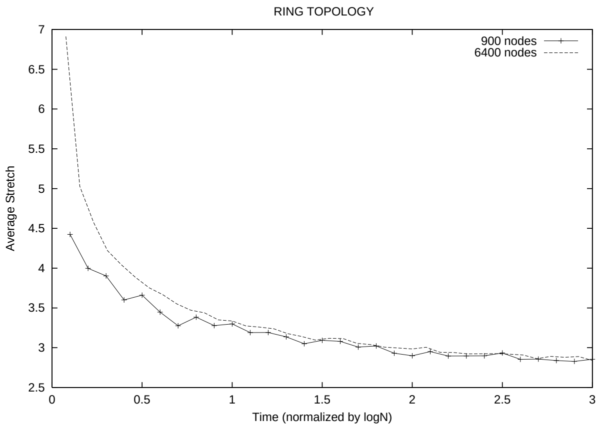

Figure 13: Convergence Time

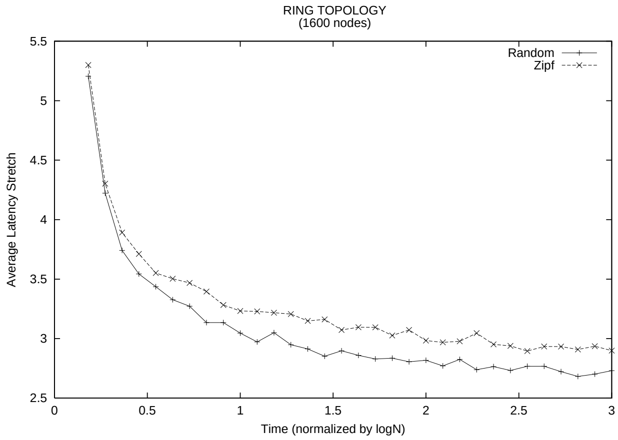

Figure 14: Impact of Zipf-ian document popularity

each of its ranges, LPRS-Chord will eventually converge to a good stretch value. Zipf-ian access does not prevent nodes from obtaining these samples–it just takes a little longer for some nodes to get $O(\log N)$ samples.

Equally interesting is the fact that the Zipf-ian access curve also exhibits an initial steep reduction in stretch. This is good news and suggests that relatively few samples are required to bring the overall system to an acceptable performance regime.

6 Related Work

In early seminal work on the problem of reducing latency in DHT systems, Plaxton *et al.* [17] showed that with carefully pre-configured routing tables, a data replication system can achieve asymptotically optimum request latency if the underlying topology has powerlaw latency expansion. Specifically, their scheme guarantees that the latency for each request will be at most a constant factor larger than the minimum possible latency *for that request*. This is stronger than the guarantee for our random sampling scheme, where the expected latency is at most a constant factor larger than the minimum expected latency. However, in their algorithm, each node needs to find the closest node in each range, which seems prohibitively expensive.

CFS [6],a Chord-based storage system, chooses proximity routing to improve its latency performance. As defined in [21], proximity routing is when routing choices are based on a weighting matrix of progress in overlay space against cost in latency. While CFS evaluates its lookup improvements using simulations, we are able to rigorously establish LPRS-Chord's performance on a large class of graphs.

Current Tapestry/Pastry systems implement several heuristic latency optimizations, often informed by the algorithm of Plaxton *et al.* The first problem they address is to initialize the routing table of a new node that joins the systems, using the routing table of a nearby node. This is akin to the *nearest-neighbor search* problem and its difficulty lies in the dynamic membership of P2P systems. Castro *et al.* [4] proposed the following heuristic for this problem in the context of Pastry. Each Pastry node maintains a list of leaf nodes uniformly distributed over the underlying network, as well as a routing table in which the entry for every range refers to a nearby node from that range. When a new node i wants to locate a nearby node, it first chooses a random node x and performs a "coarse-granularity" search by finding the nearest node y among x's leaf node set. Then, i narrows down its search area using y's routing table, and picks up the nearest node z among all entries in y's routing table. Node i then repeats the above process starting with z's routing tables. This iterative process continues until no improvement is made; the final node is declared as the *nearest neighbor*. This nearest-neighbor discovery algorithm can potentially take a long time since it needs to be executed sequentially. In addition, nearby nodes need to share their routing information explicitly for routing table maintenance, which further increases the overhead. In LPRS, the effort gets piggybacked over normal DHT functionality, and gets spread out over time.

Rhea and Kubiatowicz [22] observed that large routing stretches happen mostly when the target node is close to the source. They proposed a new probabilistic location algorithm for Tapestry based on an “attenuated bloom filter” data structure. The attenuated bloom filter of a node records the information about the data stored at all nodes up to D hops away *in the underlying network*. For each request, a Tapestry node first searches every node within D physical hops (*i.e.*, checks the information recorded in the attenuated bloom filter) before forwarding the request over the overlay network. Therefore, unnecessary overlay forwarding is avoided when the requested data is physically nearby.

Karger and Ruhl [11] use similar sampling ideas to obtain an elegant data structure for the nearest neighbor problem in graphs with power-law latency expansion. The basic idea in their scheme is that if a lot of nodes are sampled from a small ball, then with high probability, one of these points will lie in a ball with half the radius. They use this idea recursively to derive a hierarchical structure that they call the ”metric skip-list”. While the intuition behind their scheme is similar to that behind ours, there are several fundamental differences:

1. Their algorithm finds a nearby copy of the requested document in Chord but does not address the problem which we consider, i.e., improving the *lookup latency* of finding a document. For example, if there is just one copy of a document in the system, their scheme would have a lookup latency no better than that of Chord.

2. They do not present an explicit mechanism for efficient sampling. In contrast, the random sampling in our scheme can be piggybacked over existing DHT functionality.

Ratnasamy *et al.* [19] proposed a distributed binning scheme for CAN in an attempt to make the overlay topology resemble the underlying IP topology. Their scheme needs a set of k well distributed landmarks on the Internet. The whole CAN key space (d-dimensional Cartesian coordinate space) is divided into $k!$ bins, with each bin corresponding to an ordering of the k landmarks. A new node finds its bin by ordering its distance to the landmarks, and then choosing a random point in this bin as its position on the key space. This increases the likelihood that nearby nodes will be mapped into nearby bins.

Waldvogel and Rinaldi [29] proposed a heuristic in the context of *Mithos*, a topology-aware content-addressable overlay network. This is similar to the binning scheme described above in that *Mithos* is embedded into a multi-dimensional space, and neighboring nodes in the virtual space are also close to each other in the underlying network. However, unlike the distributed binning scheme, the ID (position in the virtual space) for a new *Mithos* node x is calculated based on the IDs of its neighbors and their latency from x. Their approach for nearest-neighbor-search is very similar to that used in Pastry. For routing, each *Mithos* node establishes a link to the closest neighbor in each quadrant, and forwards messages to its chosen neighbor in the same quadrant as the final destination. Therefore, the size of the routing table at each node increases exponentially with the dimension of the key space. In their simulations, 4 dimensions are shown to be enough for promising performance.

Eugene and Zhang [7] proposed a new approach to map P2P nodes into points in Euclidean space so that distances in the Euclidean space approximate distances in the underlying space. This approach needs a set of landmarks and a pre-processing phase during which the landmarks interact to compute their own coordinates in the Euclidean space. Each new node which joins the system derives its own coordinates based on its distances to the landmarks and the coordinates of the landmarks.

7 Conclusions and Future Work

We have described LPRS, a fast random sampling technique for DHT systems that use geometric routing. We have analytically shown that on graphs with power-law latency expansion, LPRS can result in an average lookup latency that is proportional to the average unicast latency. Analysis of a very-large Internet router-level topology dataset also shows that the latency expansion of the Internet resembles a power-law. This immediately implies that LPRS is a very practical approach to reducing lookup latency in DHT systems. In fact, we believe this work increases the practicality of DHT systems as infrastructures that provide low-latency wide-area storage and rendezvous.

While it is highly encouraging that LPRS converges quite quickly (on the order of 10s of lookups even for relatively large overlays), it is important to understand the tradeoff between traffic and the level of node dynamics and their impact on convergence time. We are currently evaluating this tradeoff, and our preliminary results suggest that LPRS-Chord exhibits low stretch even in highly dynamic scenarios (*e.g.*, the rate of node joins and leaves is comparable to the rate of lookups).

References

[1] W. Aiello, F. Chung, and L. Lu. *A Random Graph Model for Massive Graphs.* the 32nd Annual Symposium on Theory of Computing, 2000.

[2] B. Bollobas. *Random Graphs.* Academic Press, Inc. Orlando, Florida, 1985.

[3] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker. *Web Caching and Zipf-like Distribution: Evidence and Implications.* IEEE INFOCOM, 1999.

[4] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. *Exploiting Network Proximity in Peer-to-Peer Overlay Networks.* Technical report MSR-TR-2002-82, 2002.

[5] R. Cox, A. Muthitacharoen, R. Morris. *Serving DNS Using Chord.* the First International Workshop on Peer-to-peer Systems (IPTPS'02). March, 2002.

[6] F. Dabek, M. F. Kasshoek, D. Karger, R. Morris, and I. Stoica. *Wide-area Cooperative Storage with CFS.* ACM SOSP'01. October 2001.

[7] T.S. Eugene, and H. Zhang. *Predicting Internet Network Distance with Coordinates-based Approaches.* IEEE INFOCOM, 2002.

[8] C.Faloutsos, P. Faloutsos, and M.Faloutsos. *On Power-law Relationships of the Internet Topology.* ACM SIGCOMM, 1999.

[9] R. Govindan and H. Tangmunarunkit. *Heuristics for Internet Map Discovery.* IEEE INFOCOM, 2000.

[10] S. Iyer, A. Rowstron, and P. Druschel. *SQUIRREL: A Decentralized Peer-to-peer Web Cache.* the 21st ACM Symposium on Principles of Distributed Computing (PODC2002). July 2002.

[11] D.R. Karger and M. Ruhl. *Finding Nearest Neighbors in Growth-restricted Metrics.* ACM Symposium on Theory of Computing (STOC '02). May 2002.

[12] A. Keromytis,, V. Misra and D. Rubenstein. *SOS: Secure Overlay Services.* ACM SIGCOMM, 2002.

[13] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. *Oceanstore: An Architecture for Global-scale Persistent Storage.* the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.

[14] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge, 1995.

[15] V.N. Padmanabhan, L. Subramanian. *An Investigation of Geographic Mapping Techniques for Internet Hosts.* ACM SIGCOMM, 2001.

[16] G. Phillips, S. Shenker, H. Tangmunarunkit. *Scaling of Multicast Trees: Comments on the Chuang-Sirbu Scaling Law.* ACM SIGCOMM, 1999.

[17] C.G. Plaxton, R. Rajaraman, and A.W Richa. *Accessing Nearby Copies of Replicated Objects in A Distributed Environment.* the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures, 1997.

[18] S. Ratnaswamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A Scalable Content-addressable Network.* ACM SIGCOMM, 2001.

[19] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. *Topologically-Aware Overlay Construction and Server Selection.* IEEE INFOCOM, 2002.

[20] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. *Application-level Multicast Using Content-addressable Networks.* NGC 2001. Nov. 2001.

[21] S. Ratnasamy, S. Shenker, and I. Stoica. *Routing Algorithms for DHTs: Some Open Questions.* the First International Workshop on Peer-to-peer Systems (IPTPS'02). March, 2002.

[22] S. C. Rhea, J. Kubiatowicz. *Probabilistic Location and Routing.* IEEE INFOCOM, 2002.

[23] A. Rowstron and P. Druschel. *Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems.* International Conference on Distributed Systems Platforms (Middleware), Nov 2001.

[24] A. Rowstron, A.M. Kermarrec, M. Castro, and P. Druschel. *Scribe: The Design of A Large-scale Event Notification Infrastructure.* the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Nov. 2001.

[25] N. Spring, R. Mahajan, D. Wetherall. *Measuring ISP Topologies with Rocketfuel*. ACM SIGCOMM, 2002.

[26] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, S. Surana. *Internet Indirection Infrastructure.* ACM SIGCOMM, 2002.

[27] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. *Chord: A Peer-to-peer Lookup Service for Internet Applications.* ACM SIGCOMM, 2001.

[28] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, W. Willinger, *Network Topology Generators: Degree-Based vs. Structural.* ACM SIGCOMM 2002.

[29] M. Waldvogel, R. Rinaldi. *Efficient Topology-Aware Overlay Network.* First Workshop on Hot Topics in Networks (HotNets-I).October 2002

[30] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. *Tapestry: An Infrastructure for Fault-resilient Wide-area Location and Routing.* Tech. Report UCB/CSD-01-1141, U.C. Berkeley, April 2001.

[31] Chord simulator. *http://pdos.lcs.mit.edu/cgi-bin/cvsweb.cgi/sfsnet/simulator*

A Definitions and Proofs

Next, we will give formal definitions of power-law and exponential latency expansions, and then prove theorem 3, claim 4.1, and claim 4.2.

Let $\Delta = \max_{x \in V, y \in V} l(u, v)$ denote the diameter of graph G. Let $N_u(x)$ denote the set of all nodes $v \neq u, v \in V$ such that $l(u, v) \leq x$.

Power-law latency expansion: A family of graphs has a *power-law latency expansion* if there exist constants $c_1, c_2 > 0$ and $d_1, d_2 \geq 1$ such that for all graphs $G = (V, E)$ in the family, and each node $u \in V$, the following are true:

1. $|N_u(x)| \leq c_1 x^{d_1}$ for all $x \geq 0$, and
2. $|N_u(x)| \geq c_2 x^{d_2}$ for all x, where $1 \leq x \leq ((|V| - 1))/c_2)^{1/d_2}$.

Essentially, power-law latency expansion means that $|N_u(x)|$ grows like a polynomial in x. Some simple examples of graph families with power-law latency expansion are rings, lines, (both have $d_1 = d_2 = 1$), and grids ($d_1 = d_2 = 2$), where each edge has latency 1. We will say that a family of graphs has d-power-law expansion if $d_1 = d_2 = d$ in the above definition.

Exponential latency expansion: A family of graphs has an *exponential latency expansion* if there exist constants $c_1, c_2 > 0$ and $\alpha_1, \alpha_2 > 1$ such that for all graphs $G = (V, E)$ in the family, and each node $u \in V$, the following are true:

1. $|N_u(x)| \leq c_1 \alpha_1^x$ for all x, and
2. $|N_u(x)| \geq c_2 \alpha_2^x$ for all x, where $1 \leq x \leq \log_\alpha((|V| - 1)/c_2)$.

Proof of theorem 3: *If G is drawn from a family of graphs with exponential latency expansion, then the expected latency of any frugal DHT scheme is $\Omega(\Delta \log N)$.*

The basic idea is that in a graph with exponential latency expansion, an overwhelming majority of nodes are very far from any given node u. Therefore, even the closest node in $R_i(u)$ is likely to be $O(\Delta)$ away from u for most values of i, and hence, a large number of hops in a geometric route must incur an expected latency of at least $\Omega(\Delta)$. We will present a brief proof for the hypercube case; the arguments for the linear case are very similar and are omitted.

Let $L(i)$ denote the expected latency of the route chosen by the DHT scheme from an arbitrary node u to a node v chosen uniformly at random from $R_0(u) \cup R_1(u) \cup \ldots \cup R_i(u)$. We will obtain a recurrence for $L(i)$. Clearly, $L(\log N)$ represents the latency from an arbitrary node to a random destination.

Let $l_i(u)$ denote the expected minimum latency between u and a node in $R_i(u)$. Since the function $\textsc{Id}()$ is random, the set $R_i(u)$ itself is chosen uniformly at random from all subsets of V of size 2^{i-1}. If G has exponential latency expansion, then it follows that $l_i(u) = \Omega((1 - i/\log N)\Delta)$ (we omit the details from this version).

Node v lies in $R_i(u)$ with probability $1/2$ and in $R_0(u) \cup R_1(u) \cup \ldots \cup R_{i-1}(u)$ with probability $1/2$. If v lies in $R_i(u)$, then the next hop must be to a node $w \in R_i(u)$ which depends only on i; hence v is still a uniformly random sample from $R_0(w) \cup R_1(w) \cup \ldots \cup R_{i-1}(w)$. We can now derive the following recurrence:

$$L(i) \geq L(i - 1) + l_i(u)/2.$$

Therefore, $L(\log N) = \Omega(\Delta \sum_{i=1}^{\log N} (1 - i/\log N)) = \Omega(\Delta \log N)$. ■

Proof of claim 4.1: *Let S be a subset of V of size p chosen uniformly at random from all such subsets. Then* $\mathbf{E}[l(u, S)] = O(\Delta/|S|^{1/d})$.

$$\mathbf{E}[l(u, S)] = \int_0^\Delta \mathbf{Pr}[l(u, S) > x] dx.$$

The event that $l(u, S) > x$ is the same as the event that the two sets $N_u(x)$ and S are disjoint, and hence has probability $(1 - N_u(x)/N)^p$, where $p = |S|$. Since G belongs to a family of graphs with d-power-law latency expansion, $N_u(x)/N \geq$

$(cx/\Delta)^d$ for some constant c and $\forall x \geq 1$. This, in turn, implies that

$$\mathbf{E}[l(u,S)] \leq 1 + \int_1^{\Delta} \left(1 - (cx/\Delta)^d\right)^p dx.$$

Using the fact that $1 - z \leq e^{-z}$ for all $z \geq 0$, and expanding the range of integration, we obtain

$$\begin{aligned}
\mathbf{E}[l(u,S)] &< 1 + \int_0^{\infty} e^{-p(cx/\Delta)^d} dx \\
&= 1 + \Delta/(cp^{1/d}) \int_1^{\infty} e^{-y^d} dy \\
&= O(\Delta/p^{1/d}).
\end{aligned}$$

■

Proof of claim 4.2: $\mathbf{E}[1/|S_i(u)|^{1/d}] = O((1/\min\{J, 2^i\})^{1/d})$.

Notice that $f(x^{1/d})$ is a concave function of x if $x \geq 0, d \geq 1$. Hence, Jensen's inequality implies that for $d \geq 1$,

$$\mathbf{E}[1/|S_i(u)|^{1/d}] \leq (\mathbf{E}[1/|S_i(u)|])^{1/d}.$$

Hence, we only need to prove the claim for $d = 1$.

Consider the system where there are B bins and M balls, and each ball is thrown in a bin chosen uniformly at random, and independent of all other balls. Let $X(B,M)$ denote the number of non-empty bins at the end. This system corresponds to our system, with $J = M, 2^i = B$, and $|S_i(u)| = X(B,M)$. So our goal now is to evaluate $\mathbf{E}[1/X(B,M)]$.

First, consider the case $M \leq B$. This is a well studied system, and we will use standard Chernoff bound tricks [14] to obtain $\mathbf{E}[1/X(B,M)] = \Theta(1/M)$. Let γ represent the probability that $X(B,M) < M/3$. Then $\mathbf{E}[1/X(B,M)] \leq \gamma \cdot (1/1) + (1-\gamma) \cdot (1/(M/3))$. Applying Chernoff bounds, it is easy to see that $\gamma = o(1/M)$, which immediately yields $\mathbf{E}[1/X(B,M)] = O(1/M)$.

When $M > B$, we divide the experiment in two parts. We first throw B balls in B bins, and obtain $X(B,B)$ non-empty bins. We then throw the remaining $M - B$ balls. This can only increase the number of non-empty bins, and hence, $X(B,M) \geq X(B,B)$. Therefore, $\mathbf{E}[1/X(B,M)]$ is at most $\mathbf{E}[1/X(B,B)]$, which is $\theta(1/B)$ from the first case.

Putting the two cases together gives $\mathbf{E}[1/X(B,M)] = O(1/\min\{B,M\})$, which completes the proof of the claim.

■

B Comparison of different sampling strategies for LPRS-Chord

Although we have described (in section 3) the strategy in which each node on the lookup path samples the target, there are many other *conceivable* sampling strategies, some of which may be easier to implement in practice. Consider a lookup that is initiated by node x_0, then forwarded to node x_1, x_2, ..., and finally reaches the request terminator, node x_n. The following schemes suggest themselves:

1. Node x_i samples node x_n, $0 \leq i < n$
2. Node x_n samples nodes x_0, ..., x_{n-1}
3. Node x_i samples node x_{i-1}, $1 \leq i \leq n$
4. Node x_0 samples node x_n
5. Node x_i samples node x_0, $1 \leq i \leq n$

Of these, the first scheme is the one that LPRS-Chord uses. Schemes 4 and 5 result in most of the samples being from the largest range and hence, do not result in optimum latency improvement. Schemes 2 and 3 are interesting. Since only uni-directional routing is allowed on the Chord ring, these two schemes suffer from the same defects as schemes 4 and 5 for Chord, and should result in sub-optimal improvement. Pastry and Tapestry, however, use bi-directional routing. Hence, these schemes may perform well for Pastry and Tapestry, and deserve further exploration.

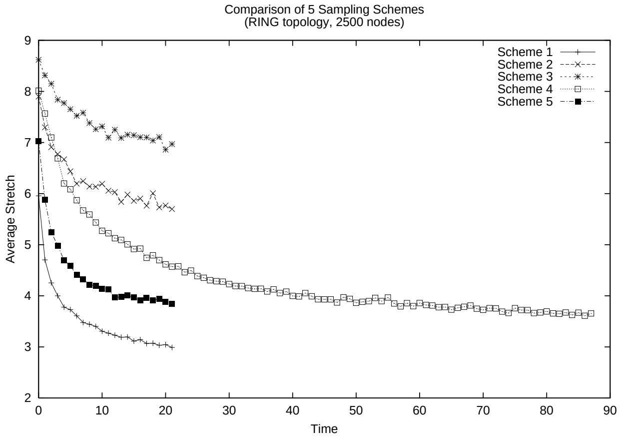

Figure 15: 5 Sampling schemes: latency stretch

We have simulated these other schemes to validate the above analysis. The simulation methodology is the same as that in section 5. We ran the simulations over a 2500-node Chord network on ring topology. Each node generates requests at a uniform rate; each request is to a randomly chosen target/document. Scheme 4 was simulated four times longer than the other schemes during the third phase. Since the average number of hops for a request was roughly four, each scheme had roughly the same number of samples. Figure 15 describes the results. Time on x-axis is in term of number of `find_doc` requests generated per node. The simulation results support our intuition.