Topology-aware routing in structured peer-to-peer overlay networks

Miguel Castro[1] Peter Druschel[2] Y. Charlie Hu[3] Antony Rowstron[1]

[1]Microsoft Research, 7 J J Thomson Close, Cambridge, CB3 0FB, UK.
[2]Rice University, 6100 Main Street, MS-132, Houston, TX 77005, USA.
[3]Purdue University, 1285 EE Building, West Lafayette, IN 47907, USA.

Technical Report
MSR-TR-2002-82

Structured peer-to-peer (p2p) overlay networks like CAN, Chord, Pastry and Tapestry offer a novel platform for a variety of scalable and decentralized distributed applications. They provide efficient and fault-tolerant routing, object location and load balancing within a self-organizing overlay network. One important aspect of these systems is how they exploit network proximity in the underlying Internet. We present a study of topology-aware routing approaches in p2p overlays, identify proximity neigbor selection as the most promising technique, and present an improved design in Pastry. Results obtained via analysis and via simulation of two large-scale topology models indicate that it is possible to efficiently exploit network proximity in self-organizing p2p substrates. Proximity neighbor selection incurs only a modest additional overhead for organizing and maintaining the overlay network. The resulting locality properties improve application performance and reduce network usage in the Internet substantially. Finally, we show that the impact of proximity neighbor selection on the load balancing in the p2p overlay is minimal.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

1 Introduction

Several recent systems (e.g., CAN, Chord, Pastry and Tapestry [7, 13, 10, 16, 6]) provide a self-organizing substrate for large-scale peer-to-peer applications. Among other uses, these systems can implement a scalable, fault-tolerant distributed hash table, in which any item can be located within a bounded number of routing hops, using a small per-node routing table. While there are algorithmic similarities among each of these systems, one important distinction lies in the approach they take to considering and exploiting proximity in the underlying Internet. Chord in its original design, for instance, does not consider network proximity at all. As a result, its protocol for maintaining the overlay network is very light-weight, but messages may travel arbitrarily long distances in the Internet in each routing hop.

In a version of CAN, each node measures its network delay to a set of landmark nodes, in an effort to determine its relative position in the Internet and to construct an Internet topology-aware overlay. Tapestry and Pastry construct a topology-aware overlay by choosing nearby nodes for inclusion in their routing tables. Early results for the resulting locality properties are promising. However, these results come at the expense of a significanly more expensive overlay maintenance protocol, relative to Chord. Also, proximity based routing may compromise the load balance in the p2p overlay network. Moreover, it remains unclear to what extent the locality properties hold in the actual Internet, with its complex, dynamic, and non-uniform topology. As a result, the cost and effectiveness of proximity based routing in these p2p overlays remain unclear.

This paper presents a study of proximity based routing in structured p2p overlay networks, and presents results of an analysis and of simulations based on two large-scale Internet topology models. The specific contributions of this paper include

- a comparison of approaches to proximity based routing in structured p2p overlay networks, which identifies proximity neighbor selection in prefix-based protocols like Tapestry and Pastry as the most promising technique;
- improved node join and overlay maintenance protocols for proximity neighbor selection in Pastry, which significancly reduce the overhead of creating and maintaining a topology-aware overlay;
- a study of the costs and benefits of proximity neighbor selection via analysis and simulation based on two large-scale Internet topology models;
- a study of the impact of proximity neighbor selection on the load balancing in the p2p overlay based on simulations on a large-scale topology model.

Compared to the original Pastry paper [10], this work adds a comparison with other proposed approaches to topology-aware routing, new node join and overlay maintenance protocols that dramatically reduce the cost of overlay construction and maintenance, a new protocol to locate a nearby contact node, results of a formal analysis of Pastry's routing properties and extensive simulation results on two different network topology models.

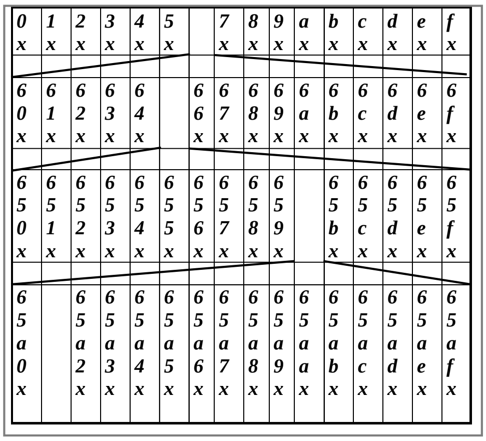
The rest of this paper is organized as follows. Previous work on structured p2p overlays is discussed in Section 2. Approaches to topology-aware routing in p2p overlays are presented in Section 3. Section 4 presents Pastry's implementation of proximity neighbor selection, including new efficient protocols for node join and overlay maintenance. An analysis of Pastry's locality properties follow in Section 5. Section 6 presents experimental results, and we conclude in Section 7.

2 Background and prior work

In this section, we present some background on structured p2p overlay protocols like CAN, Chord, Tapestry and Pastry. (We do not consider unstructured p2p overlays like Gnutella and Freenet in this paper [1, 2]). Space limitations prevent us from a detailed discussion of each protocol. Instead, we give a more detailed description of Pastry, as an example of a structured p2p overlay network, and then point out relevant differences with the other protocols.

2.1 Pastry

Pastry is a scalable, fault resilient, and self-organizing peer-to-peer substrate. Each Pastry node has a unique, uniform randomly assigned *nodeId* in a circular 128-bit identifier space. Given a 128-bit key, Pastry routes an associated message towards the live node whose nodeId is numerically closest to the key. Moreover, each Pastry node keeps track of its neighboring nodes in the namespace and notifies applications of changes in the set.



0x	1x	2x	3x	4x	5x		7x	8x	9x	ax	bx	cx	dx	ex	fx
60x	61x	62x	63x	64x		66x	67x	68x	69x	6ax	6bx	6cx	6dx	6ex	6fx
650x	651x	652x	653x	654x	655x	656x	657x	658x	659x		65bx	65cx	65dx	65ex	65fx
65a0x		65a2x	65a3x	65a4x	65a5x	65a6x	65a7x	65a8x	65a9x	65aax	65abx	65acx	65adx	65aex	65afx

Figure 1: Routing table of a Pastry node with nodeId $65a1x$, $b = 4$. Digits are in base 16, x represents an arbitrary suffix.

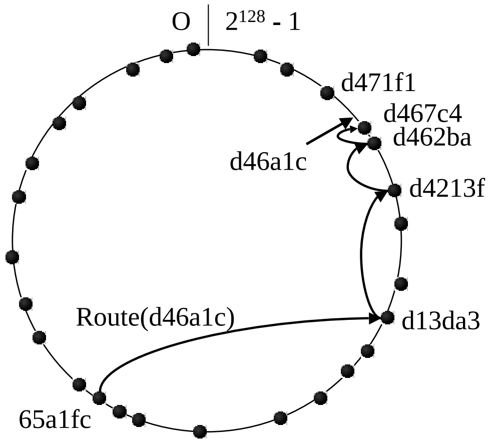


Figure 2: Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

Node state: For the purpose of routing, nodeIds and keys are thought of as a sequence of digits in base 2^b (b is a configuration parameter with typical value 4). A node's routing table is organized into $128/2^b$ rows and 2^b columns. The 2^b entries in row n of the routing table contain the IP addresses of nodes whose nodeIds share the first n digits with the present node's nodeId; the $n + 1$th nodeId digit of the node in column m of row n equals m. The column in row n that corresponds to the value of the $n + 1$'s digits of the local node's nodeId remains empty. Figure 1 depicts a sample routing table.

A routing table entry is left empty if no node with the appropriate nodeId prefix is known. The uniform random distribution of nodeIds ensures an even population of the nodeId space; thus, on average only $\lceil log_{2^b} N \rceil$ levels are populated in the routing table. Each node also maintains a *leaf set*. The leaf set is the set of l nodes with nodeIds that are numerically closest to the present node's nodeId, with $l/2$ larger and $l/2$ smaller nodeIds than the current node's id. A typical value for l is approximately $\lceil 8 * log_{16} N \rceil$. The leaf set ensures reliable message delivery and is used to store replicas of application objects.

Message routing: At each routing step, a node seeks to forward the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the current node's shared prefix. If no such node can be found in the routing table, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node's id. Several such nodes can normally be found in the routing table; moreover, such a node is guaranteed to exist in the leaf set unless the message has already arrived at the node with numerically closest nodeId or its immediate neighbor. And, unless all $l/2$ nodes in one half of the leaf set have failed simultaneously, at least one of those nodes must be live.

The Pastry routing procedure is shown in Figure 3. Figure 2 shows the path of an example message. Analysis shows that the expected number of forwarding hops is slightly below $\lceil log_{2^b} N \rceil$, with a distribution that is tight around the mean. Moreover, simulation shows that the routing is highly resilient to node failures.

2.2 CAN, Chord, Tapestry

Next, we briefly describe CAN, Chord and Tapestry, with an emphasis on the differences of these protocols when compared to Pastry.

Tapestry is very similar to Pastry but differs in its approach to mapping keys to nodes in the sparsely populated id space, and in how it manages replication. In Tapestry, there is no leaf set and neighboring nodes in the namespace are not aware of each other. When a node's routing table does not have an entry for a node that matches a key's nth digit, the message is forwarded to the node with the next higher value in the nth digit, modulo 2^b, found in the routing table. This procedure, called *surrogate routing*, maps keys to a unique live node if the node routing tables are consistent. For fault tolerance, Tapestry inserts replicas of data items using different keys.

Like Pastry, Chord uses a circular id space. Unlike Pastry, Chord forwards messages only in clockwise di-

```
if ($d.isBetween(L_{-l/2}, L_{l/2})$)
    // $d$ is within range of local leaf set (mod $2^{128}$)
    forward to $L_i$, s.th. $|d - L_i|$ is minimal;
else
    // use the routing table
    Let $l = shl(d, a)$;
    if ($R_l^{d_l}$ exists and is live)
        forward to $R_l^{d_l}$;
    else
        // rare case
        forward to $t \in L \cup R$, s.th.
            $shl(t, d) \geq l$,
            $|t - d| < |a - d|$
```

Figure 3: Pastry routing procedure, executed when a message with key d arrives at a node with nodeId a. R_l^i is the entry in the routing table R at column i and row l. L_i is the i-th closest nodeId in the leaf set L, where a negative/positive index indicates counterclockwise/clockwise from the local node in the id space, respectively. $L_{-l/2}$ and $L_{l/2}$ are the nodes at the edges of the local leaf set. d_l represents the l's digit in the key d. $shl(a, b)$ is the length of the prefix shared among a and b, in digits.

rection in the circular id space. Instead of the prefix-based routing table in Pastry, Chord nodes maintain a finger table, consisting of up to 128 pointers to other live nodes. The ith entry in the finger table of node n refers to the live node with the smallest nodeId clockwise from $n + 2^{i-1}$. The first entry points to n's successor, and subsequent entries refer to nodes at repeatedly doubling distances from n. Each node also maintains pointers to its predecessor and to its n successors in the id space (the successor list). Similar to Pastry's leaf set, this successor list is used to replicate objects for fault tolerance. The expected number of routing hops in Chord is $\frac{1}{2}log_2N$.

CAN routes messages in a d-dimensional space, where each node maintains a routing table with $O(d)$ entries and any node can be reached in $O(dN^{1/d})$ routing hops. The entries in a node's routing table refer to its neighbors in the d-dimensional space. Unlike Pastry, Tapestry and Chord, CAN's routing table does not grow with the network size, but the number of routing hops grows faster than $logN$ in this case.

3 Topology-aware routing

In this section, we describe and compare three approaches to topology-aware routing in structured overlay networks that have been proposed, namely *topology-based nodeId assignment*, *proximity routing*, and *proximity neighbor selection* [9].

Proximity routing: With proximity routing, the overlay is constructed without regard for the physical network topology. The technique exploits the fact that when a message is routed, there are potentially several possible next hop neighbors that are closer to the message's key in the id space. The idea is to select, among the possible next hops, the one that is closest in the physical network or one that represents a good compromise between progress in the id space and proximity. With k alternative hops in each step, the approach can reduce the expected delay in each hop from the average delay between two nodes to the expected delay of the nearest among k nodes with random locations in the network. The main limitation is that the benefits depend on the magnitude of k; with practical protocols, k is small. Moreover, depending on the overlay protocol, greedily choosing the closest hop may lead to an increase in the total number of hops taken. While proximity routing can yield significant improvements over a system with no topology-aware routing, its performance falls short of what can be achieved with the following two approaches. The technique has been used in CAN and Chord [7, 4].

Topology-based nodeId assignment: Topology-based nodeId assignment attempts to map the overlay's logical id space onto the physical network such that neighbouring nodes in the id space are close in the physical network. The technique has been successfully used in a version of CAN, and has achieved delay stretch results of two or lower [7, 8]. However, the approach has several drawbacks. First, it destroys the uniform population of the id space, causing load balancing problems in the overlay. Second, the approach does not work well in overlays that use a one-dimensional id space (Chord, Tapestry, Pastry), because the mapping is overly constrained. Lastly, neighboring nodes in the id space are more likely to suffer correlated failures, which can have implications for robustness and security in protocols like Chord and Pastry, which replicate objects on neighbors in the id space.

Proximity neighbour selection: Like the previous technique, proximity neighbor selection constructs a topology-aware overlay. However, instead of biasing the nodeId assignment, the idea is to choose routing table entries to refer to the topologically nearest among all nodes with nodeId in the desired portion of the id space. The success of this technique depends on the degree of freedom an overlay protocol has in choosing routing table entries without affecting the expected number of routing hops. In prefix-based protocols like Tapestry and Pastry, the upper levels of the routing table allow great freedom in this choice, with lower levels leaving exponentially less choice. As a result, the expected delay of the first hop is very low, it increases exponentially with each hop, and the delay of the final hop dominates. As one can show, this leads to low delay stretch and other useful properties. A limitation of this technique is that it does not work for overlay protocols like CAN and Chord, which require that routing table entries refer to specific points in the id space.

Discussion: Proximity routing is the most light-weight technique, since it does not construct a topology-aware overlay. But, its performance is limited since it can only reduce the expected per-hop delay to the expected delay of the nearest among a small number k of nodes with random locations in the network. With topology-aware nodeId assignment, the expected per-hop delay can be as low as the average delay among neighboring overlay nodes in the network. However, the technique suffers from load imbalance and requires a high-dimensional id space to be effective.

Proximity-neighbor selection can be viewed as a compromise that preserves the load balance and robustness afforded by a random nodeId assignment, but still achieves a small constant delay stretch. In the following sections, we show that proximity neighbor selection can be implemented in Pastry and Tapestry with low overhead, that it achieves comparable delay stretch to topology-based nodeId assignment without sacrificing load balancing or robustness, and that is has additional route convergence properties that facilitate efficient caching and multicasting in the overlay. Moreover, we confirm these results via simulations on two large-scale Internet topology models.

4 Proximity neighbor selection: Pastry

This section shows how proximity based neighbor selection is used in Pastry. We describe new node join and overlay maintenance protocols that significantly reduce the overhead compared to the original protocols described in [10]. Moreover, we present a new protocol that allows nodes that wish to join the overlay to locate an appropriate contact node.

It is assumed that each Pastry node can measure its distance, in terms of a scalar proximity metric, to any node with a known IP address. The choice of a proximity metric depends on the desired qualities of the resulting overlay (e.g., low delay, high bandwidth, low network utilization). In practice, average round-trip time has proven to be a good metric.

Pastry uses proximity neighbor selection as introduced in the previous section. Selecting routing table entries to refer to the precisely nearest node with an appropriate nodeId is expensive in a large system, because it requires $O(N)$ communication. Therefore, Pastry uses heuristics that require only $O(log_{2^b}N)$ communication but only ensure that routing table entries are close but not necessarily the closest. More precisely, Pastry ensures the following invariant for each node's routing table:

Proximity invariant: *Each entry in a node X's routing table refers to a node that is near X, according to the proximity metric, among all live Pastry nodes with the appropriate nodeId prefix.*

In Section 4.1, we show how Pastry's node joining protocol maintains the proximity invariant. Next, we consider the effect of the proximity invariant on Pastry's routing. Observe that as a result of the proximity invariant, a message is normally forwarded in each routing step to a nearby node, according to the proximity metric, among all nodes whose nodeId shares a longer prefix with the key. Moreover, the expected distance traveled in each consecutive routing step increases exponentially, because the density of nodes decreases exponentially with the length of the prefix match. From this property, one can derive three distinct properties of Pastry with respect to network locality:

Total distance traveled (delay stretch): The expected distance of the last routing step tends to dominate the total distance traveled by a message. As a result, the average total distance traveled by a message exceeds the distance between source and destination node only by a small constant value.

Local route convergence: The paths of two Pastry messages sent from nearby nodes with identical keys tend to converge at a node near the source nodes, in the proximity space. To see this, observe that in each consecutive routing step, the messages travel exponentially larger distances towards an exponentially shrinking set of nodes. Thus, the probability of a route convergence increases in each step, even in the case where earlier (smaller) routing steps have moved the messages farther apart. This result has significance for caching applications layered on Pastry. Popular objects requested by a nearby node and cached by all nodes along the route are likely to be found when another nearby node requests the object. Also, this property is exploited in Scribe [12] to achieve low link stress in an application level multicast system.

Locating the nearest replica: If replicas of an object are stored on k nodes with adjacent nodeIds, Pastry messages requesting the object have a tendency to first reach a node near the client node. To see this, observe that Pastry messages initially take small steps in the proximity space, but large steps in the nodeId space. Applications can exploit this property to make sure that client requests for an object tend to be handled by a replica that is near the client. Exploiting this property is application-specific, and is discussed in [11].

An analysis of these properties follows in Section 5. Simulation and measurement results that confirm and quantify these properties follow in Section 6.

4.1 Maintaining the overlay

Next, we present the new protocols for node join, node failure and routing table maintenance in Pastry and show how these protocols maintain the proximity invariant. The new node join and routing table maintenance protocols supersede the “second phase” of the join protocol described in the original Pastry paper, which had much higher overhead [10].

When joining the Pastry overlay, a new node with nodeId X must contact an existing Pastry node A. A then routes a message using X as the key, and the new node obtains the nth row of its routing table from the node encountered along the path from A to X whose nodeId matches X in the first $n-1$ digits. We will show that the proximity invariant holds on X's resulting routing table, if node A is near node X, according to the proximity metric.

First, consider the top row of X's routing table, obtained from node A. Assuming the triangle inequality holds in the proximity space, it is easy to see that the entries in the top row of A's routing table are also close to X. Next, consider the nth row of X's routing table, obtained from the node A_n encountered along the path from A to X. By induction, this node is Pastry's approximation to the node closest to A that matches X's nodeId in the first $n-1$ digits. Therefore, if the triangle inequality holds, we can use the same argument to conclude that the entries of the nth row of A_n's routing table should be close to X.

At this point, we have shown that the proximity invariant holds in X's routing table. To show that the node join protocol maintains the proximity invariant globally in all Pastry nodes, we must next show how the routing tables of other affected nodes are updated to reflect X's arrival. Once X has initialized its own routing table, it sends the nth row of its routing table to each node that appears as an entry in that row. This serves both to announce its presence and to propagate information about nodes that joined previously. Each of the nodes that receives a row then inspects the entries in the row, performs probes to measure if X or one of the entries is nearer than the corresponding entry in its own routing table, and updates its routing table as appropriate.

To see that this procedure is sufficient to restore the proximity invariant in all affected nodes, consider that X and the nodes that appear in row n of X's routing table form a group of 2^b nearby nodes whose nodeIds match in the first n digits. It is clear that these nodes need to know of X's arrival, since X may displace a more distant node in one of the node's routing tables. Conversely, a node with identical prefix in the first n digits that is not a member of this group is likely to be more distant from the members of the group, and therefore from X; thus, X's arrival is not likely to affect its routing table and, with high probability, it does not need to be informed of X's arrival.

Node failure: Failed routing tables entries are repaired lazily, whenever a routing table entry is used to route a message. Pastry routes the message to another node with numerically closer nodeId. If the downstream node has a routing table entry that matches the next digit of the message's key, it automatically informs the upstream node of that entry.

We need to show that the entry supplied by this procedure satisfies the proximity invariant. If a numerically closer node can be found in the routing table, it must be an entry in the same row as the failed node. If that node supplies a substitute entry for the failed node, its expected distance from the local node is therefore low, since all three nodes are part of the same group of nearby nodes with identical nodeId prefix. On the other hand, if no replacement node is supplied by the downstream node, we trigger the routing table maintenance task (described in the next section) to find a replacement entry. In either case, the proximity invariant is preserved.

Routing table maintenance: The routing table entries produced by the node join protocol and the repair mechanisms are not guaranteed to be the closest to the local node. Several factors contribute to this, including the heuristic nature of the node join and repair mechanisms with respect to locality. Also, many practical proximity metrics do not strictly satisfy the triangle inequality and may vary over time. However, limited imprecision is consistent with the proximity invariant, and as we will show in Section 6, it does not have a significant impact on Pastry's locality properties.

However, one concern is that deviations could cascade, leading to a slow deterioration of the locality properties over time. To prevent a deterioration of the overall route quality, each node runs a periodic routing table maintenance task (e.g., every 20 minutes). The task performs the following procedure for each row of the local node's routing table. It selects a random entry in the row, and requests from the associated node a copy of that node's corresponding routing table row. Each entry in that row is then compared to the corresponding entry in the local routing table. If they differ, the node probes the distance to both entries and installs the closest entry in its own routing table.

The intuition behind this maintenance procedure is to exchange routing information among groups of nearby nodes with identical nodeId prefix. A nearby node with the appropriate prefix must be know to at least one member of the group; the procedure ensures that the entire group will eventually learn of the node, and adjust their routing tables accordingly.

Whenever a Pastry node replaces a routing table entry because a closer node was found, the previous entry is kept in a list of alternate entries (up to ten such entries are saved in the implementation). When the primary entry fails, one of the alternates is used until and unless a closer entry is found during the next periodic routing table maintenance.

4.2 Locating a nearby node

Recall that for the node join algorithm to preserve the proximity invariant, the starting node A must be close to the new node X, among all live Pastry nodes. This begs the question of how a newly joining node can detect a nearby Pastry node. One way to achieve this is to perform an "expanding ring" IP multicast, but this assumes the availability of IP multicast. In Figure 4, we present a new, efficient algorithm by which a node may discover a *nearby* Pastry node, given that it has knowledge of *some* Pastry node at any location. Thus, a joining node is only required to obtain knowledge of any Pastry node through out-of-band means, as opposed to obtaining knowledge of a nearby node. The algorithm exploits the property that location of the nodes in the seeds' leaf set should be uniformly distributed over the network. Next, having discovered the closest leaf set member, the routing table distance properties are exploited to move exponentially closer to the location of the joining node. This is achieved bottom up by picking the closest node at each level and getting the next level from it. The last phase repeats the process for the top level until no more progress is made.

```
discover(seed)
  nodes = getLeafSet(seed)
  forall node in nodes
      nearNode = closerToMe(node,nearNode)
  depth = getMaxRoutingTableLevel(nearNode)
  while (depth > 0)
      nodes = getRoutingTable(nearNode,depth - -)
      forall node in nodes
          nearNode = closerToMe(node,nearNode)
  end while
  do
      nodes = getRoutingTable(nearNode,0)
      currentClosest = nearNode
      forall node in nodes
          nearNode = closerToMe(node,nearNode)
  while (currentClosest != nearNode)
  return nearNode
```

Figure 4: Simplified nearby node discovery algorithm. *seed* is the Pastry node initially known to the joining node.

5 Analysis

In this section, we present analytical results for Pastry's routing properties. First, we analyze the distribution of the number of routing hops taken when a Pastry message with a randomly chosen key is sent from a randomly chosen Pastry node. This analysis then forms the basis for an analysis of Pastry's locality properties. Throughout this analysis, we assume that each Pastry node has a perfect routing table. That is, a routing table entry may be empty only if no node with an appropriate nodeId prefix exists, and all routing table entries point to the nearest node, according to the proximity metric. In practice, Pastry does not guarantee perfect routing tables. Simulation results presented in Section 6 show that the performance degradation due to this inaccuracy is minimal. In the following, we present the main analytical results and leave out the details of the proofs in Appendix A.

5.1 Route probability matrix

Although the number of routing hops in Pastry is asymptotically $\lceil log_{2^b} N \rceil$, the actual number of routing hops is affected by the use of the leafset and the probability that the message key already shares a prefix with the nodeId of the starting node and intermediate nodes along the routing path. In the following, we analyze the distribution of the number of routing hops based on the statistical population of the nodeId space. Since the assignment of nodeIds is assumed to be randomly uniform, this population can be captured by the binomial distribution (see, for example, [3]). For instance, the distribution of the number of nodes with a given value of the most significant nodeId digit, out of N nodes, is given by $b(k; N, 1/2^b)$.

Recall from Figure 3 that at each node, a message can be forwarded using one of three branches in the forwarding procedure. In case P_A, the message is forwarded using the leaf set L (line 3); in case P_B using the routing table R (line 8); and in case P_C using a node in $L \cup R$ (lines 11-13). We formally define the probabilities of taking these branches as well as of two special cases in the following.

Definition 1 *Let $prob(h, l, N, P_X)$ denote the probability of taking branch $P_X, X \in \{A, B, C\}$, at the $(h+1)$th hop in routing a message with random key, starting from a node randomly chosen from N nodes, with a leaf set of size l. Furthermore, we define $prob(h, l, N, P'_A)$ as the probability that the node encountered after the h-th hop is already the numerically closest node to the message, and thus the routing terminates, and define $prob(h, l, N, P'_B)$ as the probability that the node encountered after the h-th hop already shares the $(h+1)$ digits with the key, thus skipping the $(h+1)$th hop.*

We denote $prob(h, l, N, P_X), h \in [0, 128/b - 1], X \in \{A, A', B, B', C\}$ as the *probability matrix* of Pastry routing. The following Lemma gives the building block for deriving the full probability matrix as a function of N and l.

Lemma 1 *Assume branch P_B has been taken during the first h hops in routing a random message D, i.e. the message D is at an intermediate node X which shares the first h digits with D. Let K be the total number of random uniformly distributed nodeIds that share the first h digits with D. The probabilities in taking different paths at the $(h+1)$th hop is*

$$\begin{pmatrix} prob(h,l,K,P_A) \\ prob(h,l,K,P'_A) \\ prob(h,l,K,P_B) \\ prob(h,l,K,P'_B) \\ prob(h,l,K,P_C) \end{pmatrix} = \sum_{d=0}^{2^b-1} \sum_{j_0=0}^{K} b(j_0; K, \frac{1}{2^b}) \cdot$$

$$\sum_{j=0}^{K-j_0} b(j; K-j_0, \frac{d}{2^b-1}) \cdot prob\_pabc(j, j_0, K-j_0-j, h, l)$$

where $prob\_pabc(j_l, j_c, j_r, h, l)$ calculates the five probabilities assuming there are j_l, j_c, j_r nodeIds that shared the first h digits with D, but whose $(h+1)$th digits are smaller than, equal to, and larger than that of D, respectively.

Since the randomly uniformly distributed nodeIds that fall in a particular segment of the namespace containing a fixed prefix of h digits follow the binomial distribution, the hth row of the probability matrix can be calculated by summing over all possible nodeId distributions in that segment of the namespace the probability of each distribution multiplied by its corresponding probability vector given by Lemma 1. Figure 5 plots the probabilities of taking branches P_A, P_B, and P_C at each actual hop (i.e. after the adjustment of collapsing skipped hops) of Pastry routing for $N = 60000$, with $l = 32$ and $b = 4$. It shows that the $log_{16}(N)$-th hop is dominated by P_A hops while earlier hops are dominated by P_B hops. The above probability matrix can be used to derive the distribution of the numbers of routing hops in routing a random message. Figure 6 plots this distribution for $N = 60000$ with $l = 32$ and $b = 4$. The probability matrix can also be used to derive the expected number of routing hops in Pastry routing according to the following theorem.

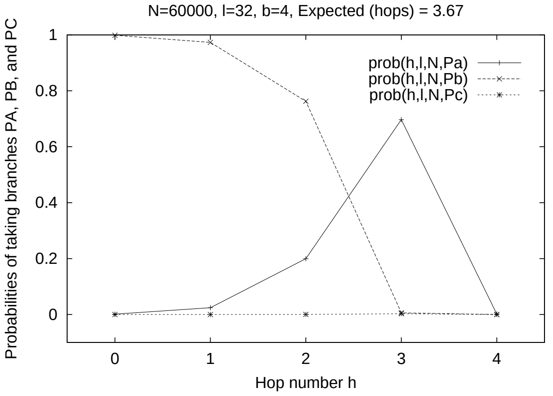


Figure 5: Probabilities $Pr(h, l, N, P_A)$, $Pr(h, l, N, P_B)$, $Pr(h, l, N, P_C)$ and expected number of hops for $N = 60000$, with $l = 32$ and $b = 4$. (From analysis.)

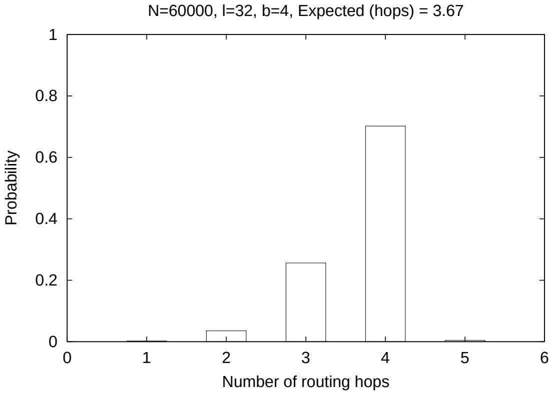


Figure 6: Distribution of the number of routing hops per message for $N = 60,000$, with $l = 32$ and $b = 4$. (From analysis.)

Theorem 1 *Let the expected number of additional hops after taking P_C for the first time, at the hth hop, be denoted as $C_{P_C}(h, l, N, P_C)$. The expected number of routing hops in routing a message with random key D starting from a node randomly chosen from the N nodes is*

$$\sum_{h=0}^{128/b-1} prob(h, l, N, P_A) - prob(h, l, N, P'_A) + \\ prob(h, l, N, P_B) - prob(h, l, N, P'_B) + \\ prob(h, l, N, P_C) + C_{P_C}(h, l, N, P_C) \cdot prob(h, l, N, P_C)$$

5.2 Expected routing distance

The above routing hop distribution is derived solely based on the randomly uniform distribution of nodeIds in the namespace. Coupled with proximity neighbor selection in maintaining the entries in Pastry's routing tables, the routing hop distribution can be used to analyze the expected total route distance.

To make the analysis tractable, it is assumed that the locations of the Pastry nodes are random uniformly distributed over the surface of a sphere, and that the proximity metric used by Pastry equals the geographic distance between pairs of Pastry nodes on the sphere. The uniform distribution of node locations and the use of geographic distance as the proximity metric are clearly not realistic. In Section 6 we will present two sets of simulation results, one for conditions identical to those assumed in the analysis, and one based on Internet topology models. A comparison of the results indicates that the impact of our assumptions on the results is limited.

Since Pastry nodes are uniformly distributed in the proximity space, the average distance from a random node to the nearest node that shares the first digit, the first two digits, etc., can be calculated based on the density of such nodes. The following Lemma gives the average distance in each hop traveled by a message with a random key sent from a random starting node, as a function of the hop number and the hop type.

Lemma 2 **(1)** *In routing message D, after h P_B hops, if $R_h^{D_h}$ is not empty, the expected $hop\_dist(h, R, P_B)$ is $Rcos^{-1}(1 - \frac{2^{b(h+1)+1}}{N})$.*
(2) *In routing message D, if path P_A is taken at any given hop, the hop distance $hop\_dist(h, R, P_A)$ is $\frac{\pi \cdot R}{2}$.*
(3) *In routing message D, after h hops, if path P_C is taken, the hop distance $hop\_dist(h, R, P_C)$ is $hop\_dist(h-1, R, P_B)$, which with high probability is followed by a hop taken via P_A, i.e. with distance $\frac{\pi \cdot R}{2}$.*

The above distance $hop\_dist(h, R, P_B)$ comes from the density argument. Assuming nodeIds are uniformly distributed over the surface of the sphere, the average distance of the next P_B hop is the radius of a circle that contains on average one nodeId (i.e. the nearest one) that share $(h + 1)$ digits with D.

Given the vector of the probabilities of taking branches P_A, P_B, and P_C at the actual hth hop (e.g. Figure 5), and the above vector of per-hop distance for the three types of hops at the hth hop, the average distance of the hth actual hop is simply the dot-product of the two vectors, i.e. the weighted sum of the hop distances by the probabilities that they are taken. These results are presented in the next section along with simulation results.

5.3 Local route convergence

Next, we analyze Pastry's route convergence property. Specifically, when two random Pastry nodes send a message with the same randomly chosen key, we analyze the expected distance the two messages travel in the proximity space until the point where their routes converge, as a function of the distance between the starting nodes in the proximity space.

To simplify the analysis, we consider three scenarios. In the worst-case scenario, it is assumed that at each routing hop prior to the point where their routes converge, the messages travel in opposite directions in the proximity space. In the average-case scenario, it is assumed that prior to convergence, the messages travel such that their distance in the proximity space does not change. In the best case scenario, the messages travel towards each other in the proximity space prior to their convergence.

For each of the above three scenarios, we derive the probability that the two routes converge after each hop. The probability is estimated as the intersecting area of the two circles potentially covered by the two routes at each hop as a percentage of the area of each circle. Coupling this probability vector with the distance vector (for different hops) gives the expected distance till route convergence.

Theorem 2 *Let $C1$ and $C2$ be the two starting nodes on a sphere of radius R from which messages with an identical, random key are being routed. Let the distance between $C1$ and $C2$ be $d0$. Then the expected distance that the two messages will travel before their paths merge is*

$$dist(d0,R) = \sum_{j=0}^{log_{2^b}N} \prod_{i=0}^{i<j} (1 - prob\_hop(i,d0,R))hop\_dist(j,R)$$

where $prob\_hop(j,d0,R) = \frac{S(hop\_dist(j,R),dj,R)}{S_{surface}(hop\_dist(j,R),R)}$, $dj = d0 + 2 \cdot \sum_{k=0}^{k<j} hop\_dist(j,R)$ in the worst case, or $dj = d0$ in the average case, or $dj = max(0, d0 - 2 \cdot \sum_{k=0}^{k<j} hop\_dist(j,R))$ in the best case, respectively, $S(r,d,R)$ denotes the intersecting area of two circles of radius r centered at two points on a sphere of radius R that are a distance of $d < 2r$ apart, and $S_{surface}(r,R)$ denotes the surface area of a circle of radius r on a sphere of radius R.

Figure 7 plots the average distance traveled by two messages sent from two random Pastry nodes with the same random key, as a function of the distance between the two starting nodes. Results are shown for the "worst case", "average case", and "best case" analysis.

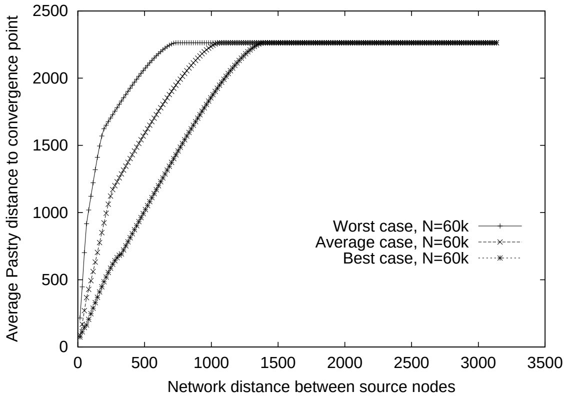


Figure 7: Distance among source nodes routing messages with the same key, versus the distance traversed until the two paths converge, for a 60,000 node Pastry network, with l=32 and b=4. (From analysis.)

6 Experimental results

Our analysis of proximity neighbor selection in Pastry has relied on assumptions that do not generally hold in the Internet. For instance, the triangle inequality does not generally hold for most practical proximity metrics in the Internet. Also, nodes are not uniformly distributed in the resulting proximity space. Therefore, it is necessary to confirm the robustness of Pastry's locality properties under more realistic conditions. In this section, we present experimental results quantifying the performance of proximity neighbor selection in Pastry under realistic conditions. The results were obtained using a Pastry implementation running on top of a network simulator, using Internet topology models. The Pastry parameters were set to $b = 4$ and the leafset size $l = 32$. Unless otherwise stated, results where obtained with a simulated Pastry overlay network of 60,000 nodes.

6.1 Network topologies

Three simulated network topologies were used in the experiments. The "Sphere" topology corresponds to the topology assumed in the analysis of Section 5. Nodes are placed at uniformly random locations on the surface of a sphere with radius 1000. The distance metric is based on the topological distance between two nodes on the sphere's surface. Results produced with this topology model should correspond closely to the analysis, and it was used primarily to validate the simulation environment. However, the sphere topology is not realistic, because it assumes a uniform random distribution of nodes on the Sphere's surface, and its proximity space is very regular and strictly satisfies the triangle inequality.

A second topology was generated using the Georgia Tech transit-stub network topology model [15]. The round-trip delay (RTT) between two nodes, as provided by the topology graph generator, is used as the proximity metric with this topology. We use a topology with 5050 nodes in the core, where a LAN with an average of 100 nodes is attached to each core node. Out of the resulting 505,000 LAN nodes, 60,000 randomly chosen nodes form a Pastry overlay network. As in the real Internet, the triangle inequality does not hold for RTTs among nodes in the topology model.

Finally, we used the Mercator topology and routing models [14]. The topology model contains 102,639 routers and it was obtained from real measurements of the Internet using the Mercator program [5]. The authors of [14] used real data and some simple heuristics to assign an autonomous system to each router. The resulting AS overlay has 2,662 nodes. Routing is performed hierarchically as in the Internet. A route follows the shortest path in the AS overlay between the AS of the source and the AS of the destination. The routes within each AS follow the shortest path to a router in the next AS of the AS overlay path.

We built a Pastry overlay with 60,000 nodes on this topology by picking a router for each node randomly and uniformly, and attaching the node directly to the router with a LAN link. Since the topology is not annotated with delay information, the number of routing hops in the topology was used as the proximity metric for Pastry. We count the LAN hops when reporting the length of the Pastry routes. This is conservative because the cost of these hops is usually negligible and Pastry's overhead would be lower if we did not count LAN hops.

6.2 Pastry routing hops and distance ratio

In the first experiment, 200,000 lookup messages are routed using Pastry from randomly chosen nodes, using a random key. Figure 8 shows the number of Pastry routing hops and the distance ratio for the sphere topology. Distance ratio is defined as the ratio of the distance traversed by a Pastry message to the distance between its source and destination nodes, measured in terms of the proximity metric. The distance ratio can be interpreted as the penalty, expressed in terms of the proximity metric, associated with routing a messages through Pastry instead of sending the message directly in the Internet.

Four sets of results are shown. "Expected" represents the results of the analysis in Section 5. "Normal routing table" shows the corresponding experimental results with Pastry. "Perfect routing table" shows results of experiments with a version of Pastry that uses perfect routing table. That is, each entry in the routing table is guaranteed to point to the nearest node with the appropriate nodeId prefix. Finally, "No locality" shows results with a version of Pastry where the locality heuristics have been disabled.

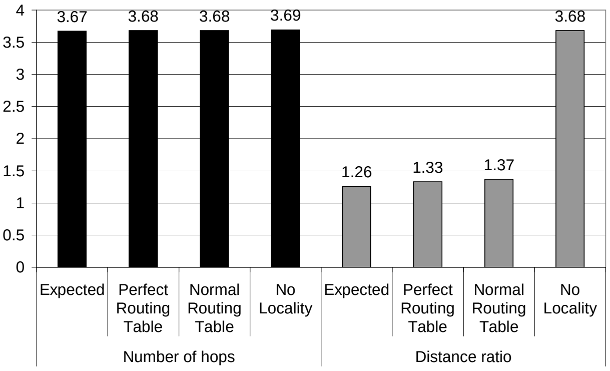


Figure 8: Number of routing hops and distance ratio, sphere topology.

All experimental results correspond well with the results of the analysis, thus validating the experimental apparatus. As expected, the expected number of routing hops is slightly below $log_{16}60,000 = 3.97$ and the distance ratio is small. The reported hop counts are virtually independent of the network topology, therefore we present them only for the sphere topology.

The distance ratio obtained with perfect routing tables is only marginally better than that obtained with the real Pastry protocol. This confirms that the node join protocol produces routing tables of high quality, i.e., entries refer to nodes that are nearly the closest among nodes with the appropriate nodeId prefix. Finally, the distance ratio obtained with the locality heuristics disabled is significantly worse. This speaks both to the importance of topology-aware routing, and the effectiveness of proximity neighbor selection.

6.3 Routing distance

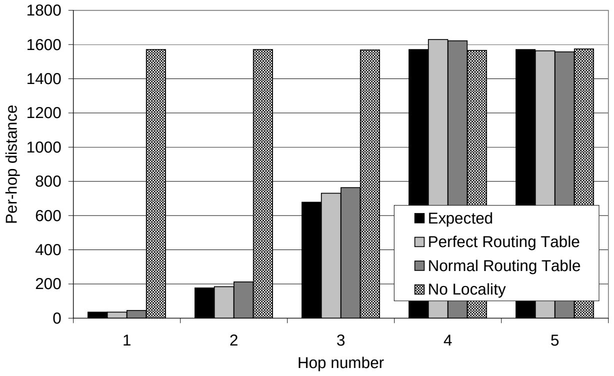


Figure 9: Distance traversed per hop, sphere topology.

Figure 9 shows the distance messages travel in each consecutive routing hop. The results confirm the exponential increase in the expected distance of consecutive hops up to the fourth hops, as predicted by the analysis. Note that the fifth hop is only taken by a tiny fraction (0.004%) of the messages. Moreover, in the absence of the locality heuristics, the average distance traveled in each hop is constant and corresponds to the average distance between nodes ($1571 = (\pi \times r)/2$, where r is the radius of the sphere).

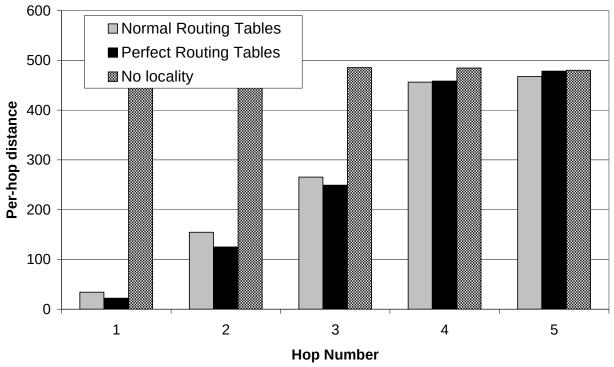


Figure 10: Distance traversed per hop, GATech topology.

Figures 10 and 11 show the same results for the GATech and the Mercator topologies, respectively. Due to the non-uniform distribution of nodes and the more complex proximity space in these topologies, the expected distance in each consecutive routing step no longer increases exponentially, but it still increases monotonically. Moreover, the node join algorithm continues to produce routing tables that refer to nearby nodes, as indicated by the modest difference in hop distance to the perfect routing tables in the first three hops.

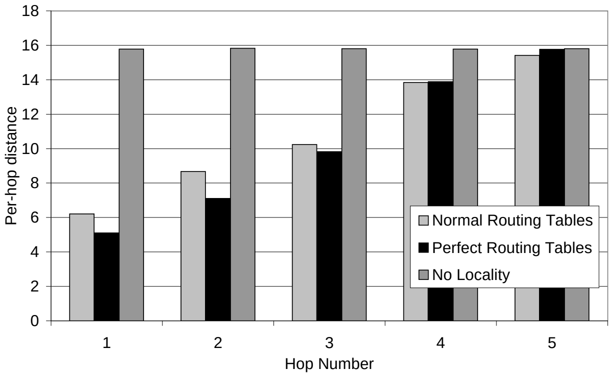


Figure 11: Distance traversed per hop, Mercator topology.

The proximity metric used with the Mercator topology makes proximity neighbor selection appear in an unfavorable light. Since the number of nodes within k IP routing hops increases very rapidly with k, there are very few “nearby” Pastry nodes. Observe that the average distance traveled in the first routing hop is almost half of the average distance between nodes (i.e., it takes almost half the average distance between nodes to reach about 16 other Pastry nodes). As a result, Pastry messages traverse relatively long distances in the first few hops, which leads to a relatively high distance ratio. Nevertheless, these results demonstrate that proximity neighbor selection works well even under adverse conditions.

Figures 12, 13 and 14 show raster plots of the distance messages travel in Pastry, as a function of the distance between the source and destination nodes, for each of the three topologies, respectively. Messages were sent from 20,000 randomly chosen source nodes with random keys in this experiment. The mean distance ratio is shown in each graph as a solid line.

The results show that the distribution of the distance ratio is relatively tight around the mean. Not surprisingly, the sphere topology yields the best results, due to its uniform distribution of nodes and the geometry of its proximity space. However, the far more realistic GATech topology yields still very good results, with a mean distance ratio of 1.59, a maximal distance ratio of about 8.5, and distribution that is fairly tight around the mean. Even the least favorable Mercator topology yields good results, with a mean distance ration of 2.2 and a maximum of about 6.5.

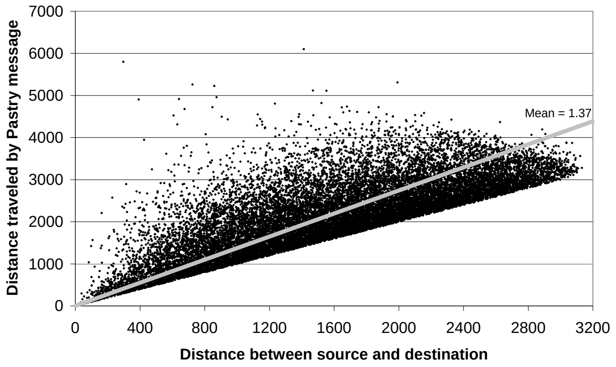


Figure 12: Distance traversed versus distance between source and destination, sphere topology.

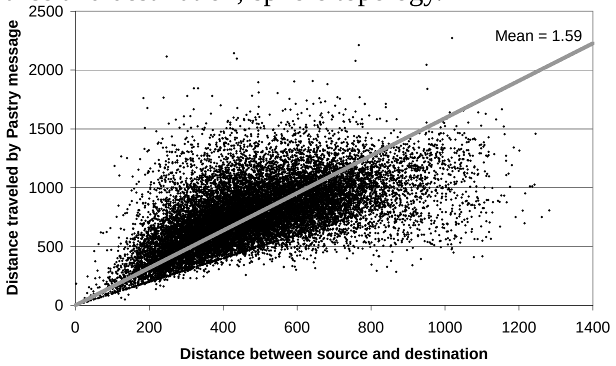


Figure 13: Distance traversed versus distance between source and destination, GATech topology.

6.4 Local route convergence

The next experiment evaluates the local route convergence property of Pastry. In the experiment, 10 nodes were selected randomly, and then for each of these nodes, 6,000 other nodes were chosen such that the topological distance between each pair provides good coverage of the range of possible distances. Then, 100 random keys were chosen and messages where routed via Pastry from each of the two nodes in a pair, with a given key.

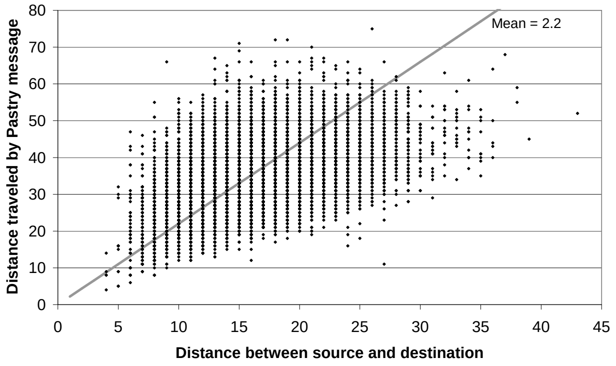


Figure 14: Distance traversed versus distance between source and destination, Mercator topology.

To evaluate how early the paths convergence, we use the metric $(\frac{c_d}{c_d+s_c^1} + \frac{c_d}{c_d+s_c^2})/2$ where, c_d is the distance traveled from the node where the two paths converge to the destination node, and s_c^1 and s_c^2 are the distances traveled from each source node to the node where the paths converge. The metric expresses the average fraction of the length of the paths traveled by the two messages that was shared. Note that the metric is zero when the paths converge in the destination. Figures 15, 16 and 17 show the average of the convergence metrics versus the distance between the two source nodes. As expected, when the distance between the source nodes is small, the paths are likely to converge quickly. This result is important for applications that perform caching, or rely on efficient multicast trees [11, 12].

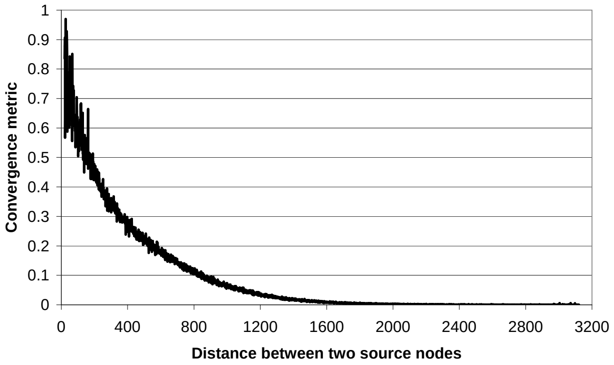


Figure 15: Convergence metric versus the distance between the source nodes, sphere topology.

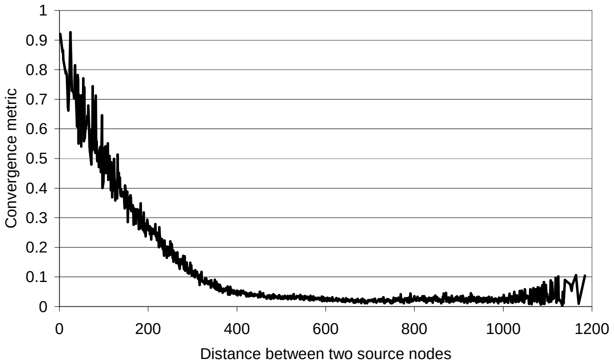


Figure 16: Convergence metric versus distance between the source nodes, GATech topology.

6.5 Overhead of node join protocol

Next, we measure the overhead incurred by the node join protocol to maintain the proximity invariant in the routing tables. We quantify this overhead in terms of the number of *probes*, where each probe corresponds to the

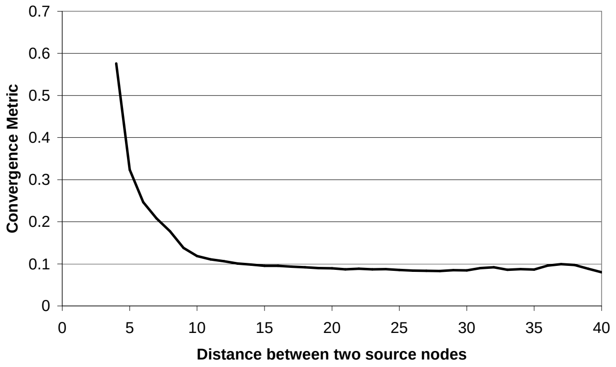


Figure 17: Convergence metric versus distance between the source nodes, Mercator topology.

communication required to measure the distance, according to the proximity metric, among two nodes. Of course, in our simulated network, a probe simply involves looking up the corresponding distance according to the topology model. However, in a real network, probing would likely require at least two message exchanges. The number of probes is therefore a meaningful measure of the overhead required to maintain the proximity invariant.

The average number of probes performed by a newly joining node was 29, with a minimum of 23 and a maximum of 34. These results were virtually independent of the overlay size, which we varied from 1,000 to 60,000 nodes. In each case, the probes performed by the last ten nodes that joined the Pastry network were recorded, which are the nodes likely to perform the most probes given the size of the network at that stage. The corresponding average number of probes performed by other Pastry nodes during the join was about 70, with a minimum of 2 and a maximum of 200.

It is assumed here that once a node has probed another node, it stores the result and does not probe again. The number of nodes contacted during the joining of a new node is $(2^b - 1)log_{2^b}N + l$, where N is the number of Pastry nodes. This follows from the expected number of nodes in the routing table, and the size of the leaf set. Although every node that appears in the joining node's routing table receives information about all the entries in the same row of the joining node's routing table, it is very likely that the receiving node already knows many of these nodes, and thus their distance. As a result, the number of probes performed per node is low (on average less than 2). This means that the total number of nodes probed is low, and the probing is distributed over a large number of nodes. The results were virtually identical for the GATech and the Mercator topologies.

6.6 Node failure

In the next experiment, we evaluate the node failure recovery protocol (Section 4.1) and the routing table maintenance (Section 4.1). Recall that leaf set repair is instantaneous, failed routing table entries are repaired lazily upon next use, and a periodic routing table maintenance task runs periodically (every 20 mins) to exchange information with randomly selected peers.

In the experiment, a 50,000 node Pastry overlay is created based on the GATech topology, and 200,000 messages from random sources with random keys are routed. Then, 20,000 randomly selected nodes are made to fail simultaneously, simulating conditions that might occur in the event of a network partition. Prior to the next periodic routing table maintenance, a new set of 200,000 random message are routed. After another periodic routing table maintenance, another set of 200,000 random messages are routed.

Figure 18 shows both the number of hops and the distance ratio at various stages in this experiment. Shown are the average number of routing hops and the average distance ratio, for 200,000 messages each before the failure, after the failure, after the first and after the second round of routing table maintenance. The "no failure" result is included for comparison and corresponds to a 30,000 node Pastry overlay with no failures. Moreover, to isolate the effects of the routing table maintenance, we give results with and without the routing table maintenance enabled.

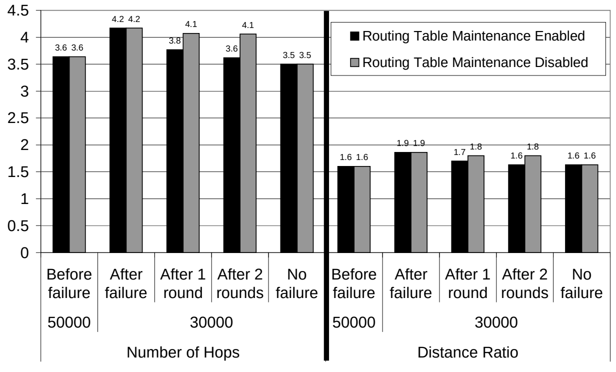


Figure 18: Routing hops and distance ratio for a 50,000 node Pastry overlay when 20,000 nodes simultaneously fail, GATech topology.

During the first 200,000 message transmissions after the massive node failure, the average number of hops and average distance ratio increase only mildly (from 3.54 to 4.17 and 1.6 to 1.86, respectively). This demonstrates the robustness of Pastry in the face of massive node failures. After each round, the results improve and approach those before the failure after two rounds.

With the routing table maintenance disabled, both the number of hops and the distance ratio do not recover as quickly. Consider that the routing table repair mechanism is lazy and only repairs entries that are actually used. Moreover, a repair generally involves an extra routing hop, because the message is routed to a node that does not share a longer prefix with the key. Each consecutive burst of 200,000 messages is likely to encounter different routing table entries that have not yet been fixed (about 95,000 entries were repaired during each bursts). The periodic routing table maintenance, on the other hand, replaces failed entries that have not yet been used as part of its routine. It is intuitive to see why the distance ratio recovers more slowly without routing table maintenance. The replacement entry provided by the repair mechanisms is generally relatively close, but not necessarily among the closest. The periodic routing table maintenance performs probing and is likely to replace such an entry with a better one. Finally, we point out that routing table maintanance also takes care of changing distances among nodes over time.

We also measured the cost of the periodic routing table maintenance, in terms of network probes, to determine the distance of nodes. On average, less than 20 nodes are being probed each time a node performs routing table maintenance, with a maximum of 82 probes. Since the routing table maintenance is performed every 20 minutes and the probes are likely to target different nodes, this overhead is not significant.

6.7 Load balance

Next, we consider how maintaining the proximity invariant in the routing tables affects load balance in the Pastry routing fabric. In the simple Pastry algorithm without the locality heuristics, or in protocols like Chord that don't consider network proximity, the "indegree" of a node, i.e., the number of routing table entries referring to a any given node, should be balanced across all nodes. This is a desirable property, as it tends to balance message forwarding load across all participating nodes in the overlay.

When routing tables entries are initialized to refer to the nearest node with the appropriate prefix, this property may be compromised, because the distribution of indegrees is now influenced by the structure of the underlying physical network topology. Thus, there is an inherent tradeoff between proximity neighbor selection and load balance in the routing fabric. The purpose of the next experiment is to quantify the degree of imbalance in indegrees of nodes, caused by the proximity invariant.

Figure 19 shows the cumulative distribution of indegrees for a 60,000 node Pastry overlay, based on the GATech topology. As expected, the results show that the distribution of indegrees is not perfectly balanced. The results also show that the imbalance is most significant at the top levels of the routing table (not shown in the graph), and that the distribution has a thin tail. This suggests that it is appropriate to deal with these potential hotspots reactively rather than proactively. If one of the nodes with a high indegree becomes a hotspot, which will depend on the workload, it can send backoff messages. The nodes that receive such a backoff message find an alternative node for the same slot using the same technique as if the node was faulty. Since the most significant imbalance occurs at the top levels of the routing table, changing routing table entries to point to an alternative node will not increase the distance ratio significantly. There are many alternative nodes that can fill out these slots and the distance traversed in the first hops accounts for a small fraction of the total distance traversed. We conclude that imbalance in the routing fabric as a result of the proximity invariant does not appear to be a significant problem.

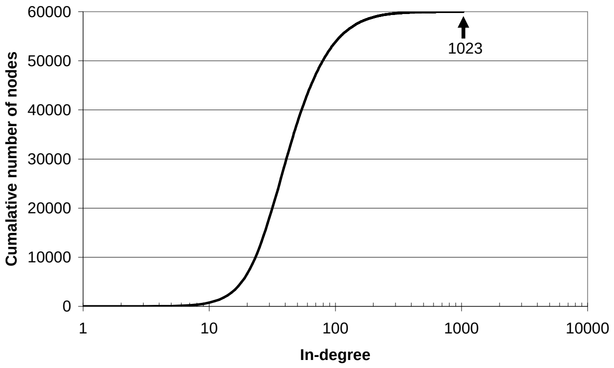


Figure 19: Indegree distribution of 60,000 Pastry nodes, GATech topology.

6.8 Discovering a nearby seed node

Next, we evaluate the discovery algorithm used to find a nearby node, presented in Section 4.2. In each of 1,000 trials, we chose a pair of nodes randomly among the 60,000 Pastry nodes. One node in the pair is considered

	Exact closest	Average Distance	Average RT0 Distance	Number Probes
Sphere	95.3%	11.0	37.1	157
GATech	83.7%	82.1	34.1	258
Mercator	32.1%	2.6	6.0	296

Table 1: Results for the closest node discovery algorithm.

the joining node that wishes to locate a nearby Pastry node, the other is treated as the seed Pastry node known to the joining node. Using this seed node, the node discovery algorithm was used to discover a node near the joining node, according to the proximity metric. Table 1 shows the results for the three different topologies. The first column shows the number of times the algorithm produced the closest existing node. The second column shows the average distance, according to the proximity metric, of the node produced by the algorithm, in the cases where the nearest node was not found. For comparison, the third column shows the average distance between a node and its row zero routing table entries. The fourth column shows the number of probes performed per trial.

In the sphere topology, over 95% of the found nodes are the closest. When the closest is not found, the average distance to the found node is significantly less than the average distance to the entries in the first level of the routing table. More interestingly, this is also true for the Mercator topology, even though the number of times the closest node was found is low with this topology. The GATech result is interesting, in that the fraction of cases where the nearest node was found is very high (almost 84%), but the average distance of the produces node in the cases where the closest node was not found is high. The reason is that the highly regular structure of this topology causes the algorithm to sometimes get into a “local minimum”, by getting trapped in a nearby network. Overall, the algorithm for locating a nearby node is effective. Results show that the algorithms allows newly joining nodes to efficiently discover a nearby node in the existing Pastry overlay.

6.9 Testbed measurements

Finally, we performed preliminary measurements in a small testbed of about 20 sites throughout the US and Europe. The measured results were as expected, but the testbed is too small to obtain interesting an representative results. We expect that a current initiative by a number of organizations to put together a larger wide-area testbed will allow us to include such results in the final version of this paper.

7 Conclusion

This paper presents a study of topology-aware routing in structured p2p overlay protocols. We compare approaches to topology-aware routing and identify proximity neighbor selection as the most promising technique. We present improved protocols for proximity based neighbor selection in Pastry, which significantly reduce the overhead of topology-aware overlay construction and maintenance. Analysis and simulations confirm that proximity neighbor selection yields good performance at very low overhead. We conclude that topology-aware routing can be accomplished effectively and with low overhead in a self-organizing, structured peer-to-peer overlay network.

References

[1] The Gnutella protocol specification, 2000. http://dss.clip2.com/GnutellaProtocol04.pdf.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, July 2000. ICSI, Berkeley, CA, USA.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[5] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *Proc. 19th IEEE INFOCOM*, pages 1371–1380, Tel Aviv, Israel, March 2000. IEEE.

[6] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS’02)*, Boston, MA, Mar. 2002.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, Aug. 2001.

[8] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. 21st IEEE INFOCOM*, New York, NY, June 2002.

[9] S. Ratnasamy, S. Shenker, and I. Stoica. Routing algorithms for dhts: Some open questions. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Boston, MA, Mar. 2002.

[10] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.

[11] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[12] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Third International Workshop on Networked Group Communications*, Nov. 2001.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[14] H. Tangmunarunkit, R. Govindan, D. Estrin, and S. Shenker. The impact of routing policy on internet paths. In *Proc. 20th IEEE INFOCOM*, Alaska, USA, Apr. 2001.

[15] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM96*, 1996.

[16] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.

Appendix A: Proofs of analytical results

We give proofs for the Lemmas and Theorems stated in Section 5. Their numberings here may be different from before.

A.1 Route probability matrix

Lemma 1 states the probability of taking path P_A at a specific hop during the routing of a random message.

Lemma 1 *Assume path P_B has been taken during the first h hops in routing message with key D, i.e. the message is at a node X that shares the first h digits with D. If there exist j_l nodeIds smaller than D that share the same h digits with D, and j_r nodeIds larger than D that share the same h digits with D, then the probability that node X will forward the message using P_A (i.e., D is within X's leaf set) is*

$$prob\_pa(j_l, j_r, h, l) = \begin{cases} \frac{l}{j_l+j_r} & \textit{if } h = 0 \\ \frac{\mathit{min}(j_l, l/2)+\mathit{min}(j_r, l/2)}{j_l+j_r} & \textit{if } h > 0 \end{cases}$$

Proof: Assume the numerically closest nodeId always shares some prefix with the message key. When the message key is within $l/2$ nodes from the boundary of the subdomain sharing the same prefix of h digits, the number of nodes in the subdomain whose leaf-sets cover D drops to $\min(j_l, l/2) + \min(j_r, l/2)$, and thus the probability that P_A will be taken next is $\frac{\min(j_l+l/2)+\min(j_r+l/2)}{j_l+j_r}$. In the very first hop, the prefix is of zero length, thus there is no bundary effect.

Since the numerically closest nodeId to message D may not share any leading digits with the key, the above probability fails to account for one additional routing hop in such cases. Since this case is rare in practice, it has virtually no effect on the above probability. □

Lemma 2 *Assume branch P_B has been taken during the first h hops in routing a random message D, i.e. the message D is at a node X which shares the first h digits with D. Let K be the total number of random uniformly distributed nodeIds that share the first h digits with D. The probabilities in taking different branches at the $(h+1)$th hop is*

$$V(h, l, K) = \begin{pmatrix} prob(h, l, K, P_A) \\ prob(h, l, K, P'_A) \\ prob(h, l, K, P_B) \\ prob(h, l, K, P'_B) \\ prob(h, l, K, P_C) \end{pmatrix} = \sum_{d=0}^{2^b-1} \sum_{j_h=0}^{K} b(j_h; K, \frac{1}{2^b}) \cdot$$

$$\sum_{j=0}^{K-j_h} b(j; K - j_h, \frac{d}{2^b - 1}) \cdot prob\_pabc(j, j_h, K - j_h - j, h, l)$$

Proof: There are 2^b subdomains of nodeIds that share the first h digits as D but differ in the $(h+1)$th digit. With equal probabilities, D and X can fall into any of these 2^b subdomains. Within each subdomain, the number of nodes j_h follows the binomial distribution, i.e. with probability $b(j_h; K, \frac{1}{2^b})$, j_h nodes can end up in each subdomain. Depending on which subdomain D falls into, there can be between 0 and up to $2^b - 1$ subdomains to the left of D's subdomain. If there are d subdomains to the left of D's subdomain, the number of nodeIds j in those subdomains follows binomial distribution $b(j; N - j_h, \frac{d}{2^b - 1})$.

Each iteration in the innermost summation corresponds to a particular distribution of j_l,j_c, and j_r, the number of nodeIds to the left of, within the same as, to the right of D's subdomain. In the formula above, these values are j, j_h, and $K - j_h - j$. The vector function $prob\_pabc(j_l, j_c, j_r, h, l)$ takes such a distribution, and assumes equal probability that X can be any of the $j_l + j_c + j_r$ nodeIds and D can be anywhere in the name space spanned by these nodeIds, and calculates the probabilities that node X will forward message D using P_A, P'_A, P_B, P'_B, or P_C, respectively.

Function $prob\_pabc(j_l, j_c, j_r, h, l)$ is calculated as follows. If $j_c = 0$, D's subdomain is empty, the next routing hop takes either P_A or P_C. The probability of P_A is $prob\_pa(j_l, j_r, h, l)$, and that of P_A is $1 - prob\_pa(j_l, j_r, h, l)$. Since we assume uniform distribution of $(j_l + j_r)$ in the subdomain of the namespace, the probability of P'_A, i.e. X is numerically closest to D, is $1/(j_l + j_r)$. If $j_c > 0$, D's subdomain is not empty, the next routing hop takes either P_A or P_B, and the probability of P'_A is $1/(j_l + j_c + j_r)$. Since there are j_c nodeIds that share the first $(h + 1)$ digits with D, there can be $(j_c + 1)$ intervals in D's subdomain that D can fall in with equal probability. For each interval, probabilities of P_A and P_B are caculated before. Furthermore, if P_A is not taken and P_B is taken, there is a certain probability that X is among the j_c nodes that already shares the next digit with D, in which case the next hop is skipped. This probability contributes to P'_B. The probability equals the number of nodeIds among the j_c nodes that are not within $l/2$ from D, over the $(j_l + j_c + j_r)$ possible candidates of X. □

Lemma 3 *Let N be the total number of random uniformly distributed nodeIds. Row 0 of the route probability matrix, i.e. the probabilities in taking different branches at the first hop in routing a random message from a random starting nodeId, is $M(0, l, N) = V(0, l, N)$.*

To calculate the probabilities at subsequent hops, note that value j_0 gives the number of nodeIds that share the first digit with message D, and the probabilities at the second hop is conditioned on the j_0 value. One way of calculating them is thus to repeat the above case analysis recursively using j_0 at the second hop, j_1 at the third hop, etc. The following theorem gives the probabilities at the $(h + 1)$th hop.

Theorem 1 *Let N be the total number of random uniformly distributed nodeIds. Row h of the route probability matrix, i.e. the probabilities in taking different branches at the $(h + 1)$th hop in routing a random message D starting from a random nodeId, is*

$$M(h, l, N) = \begin{pmatrix} prob(h, l, N, P_A) \\ prob(h, l, N, P'_A) \\ prob(h, l, N, P_B) \\ prob(h, l, N, P'_B) \\ prob(h, l, N, P_C) \end{pmatrix} = 2^b \sum_{j_0=0}^{N} b(j_0; N, \frac{1}{2^b}) \cdot$$

$$2^b \sum_{j_1=0}^{j_0} b(j_1; j_0, \frac{1}{2^b}) \cdot \ldots \cdot 2^b \sum_{j_{h-1}=0}^{j_{h-2}} b(j_{h-1}; j_{h-2}, \frac{1}{2^b}) \cdot V(h, l, j_{h-1})$$

Theorem 2 *Let the expected number of additional hops after first time taking P_C at the hth hop be denoted as $C_{P_C}(h, l, N, P_C)$. The expected number of routing hops in routing a message with random key D starting from a node randomly chosen from the N nodes is*

$$\sum_{h=0}^{128/b-1} prob(h, l, N, P_A) - prob(h, l, N, P'_A) +$$
$$prob(h, l, N, P_B) - prob(h, l, N, P'_B) +$$
$$prob(h, l, N, P_C) + C_{P_C}(h, l, N, P_C) \cdot prob(h, l, N, P_C)$$

Proof: The sum $prob(h, l, N, P_A) + prob(h, l, N, P_B) + prob(h, l, N, P_C)$ is the probability that the routing takes the $(h + 1)$th hop, and out of $prob(h, l, N, P_B)$, with probability $prob(h, l, N, P'_B)$, the routing skips future hops by one, and out of $prob(h, l, N, P_A)$, with probability $prob(h, l, N, P'_A)$, the routing skips the P_A hop at the $(h + 1)$th hop. If the intermediate node after taking h hops shares additional digits other than the $(h + 1)$th digit, the additional skipped hops will be accounted for by $prob(h + 1, l, N, P'_B)$, $prob(h + 2, l, N, P'_B)$, etc. □

Lemma 4 *Assume that branch P_B was taken during the first h hops in routing a message with key D, branch P_C is taken in the $(h + 1)$th hop, and there are k nodeIds sharing the first $(h + 1)$ digits as message D. The probability that the routing finishes in the next hop is $prob\_pc(k, l) = \sum_{j=1}^{l/2} b(j; k, \frac{1}{2^b}) + \sum_{j=l/2+1}^{k} b(j; k, \frac{1}{2^b}) \cdot \frac{l/2}{j}$.*

Proof: Let the node reached after taking branch P_C for the first time be node Q. Recall Q is selected from $L \cup R$ that is numerically closest to D whose nodeId also shares the first h digits with D. Let S_Q be the set of nodeIds in Q's subdomain, i.e. sharing the first $(h+1)$ digits with Q. The routing finishes in one step after Q if

- $S_Q \leq l/2$. The probability is $\sum_{j=1}^{l/2} b(j;k,\frac{1}{2^b})$. Or,
- $S_Q > l/2$, and the P_C hop reached one of the rightmost $l/2$ nodes in S_Q. The probability of $S_Q > l/2$ is $\sum_{j=l/2+1}^{k} b(j;k,\frac{1}{2^b})$. The probability of later is $\frac{l/2}{S_q}$, because under random uniform distribution, the probability of any of the rightmost $l/2$ nodes in S_q shows up as any node's routing table entry is $\frac{l/2}{S_Q}$.

□

The distribution of $prob(h,l,N,P_C)$ shows that its value only becomes not insignificant when $h = log_{2^b}N$, when the value k above follows binomial distribution $b(k;N,1/2^{bh})$. In such cases, $prob\_pc(k,l) > 0.997$. Thus, for non-insignificant values of $prob(h,l,N,P_C)$, with very high probability, the routing takes one extra hop after taking P_C, i.e. $C_{P_C}(h,l,N,P_C) \approx 1$.

A.2 Local route convergence

Theorem 3 *Let $C1$ and $C2$ be the two starting nodes on a sphere of radius R from which messages with an identical, random key are being routed. Let the distance between $C1$ and $C2$ be $d0$. Then the expected distance that the two messages will travel before their paths merge is*

$$dist(d0,R) = \sum_{j=0}^{log_{2^b}N} \prod_{i=0}^{i<j} (1 - prob\_hop(i,d0,R))hop\_dist(j,R)$$

where $prob\_hop(j,d0,R) = \frac{S(hop\_dist(j,R),dj,R)}{S_{surface}(hop\_dist(j,R),R)}$, $dj = d0 + 2 \cdot \sum_{k=0}^{k<j} hop\_dist(j,R)$ *in the worst case, or* $dj = d0$ *in the average case, or* $dj = max(0, d0 - 2 \cdot \sum_{k=0}^{k<j} hop\_dist(j,R))$ *in the best case, respectively,* $S(r,d,R)$ *denotes the intersecting area of two circles of radius* r *centered at two points on a sphere of radius* R *that are a distance of* $d < 2r$ *apart, and* $S_{surface}(r,R)$ *denotes the surface area of a circle of radius* r *on a sphere of radius* R.

Proof: Without loss of generality, we assume the two starting nodes are on the equator of the sphere. The expected distance of the first hop traveled in Pastry routing is $hop\_dist(0,R)$. If we draw two circles around the two starting nodes with radius of $hop\_dist(0,R)$, the intersecting area of the two circles will be $S(hop\_dist(0,R),d,R)$. Thus the probability that, after the first hop, the two paths converge into the same node is $prob\_hop(0,d0,R) = S(hop\_dist(0,R),d0,R)/S_{surface}(hop\_dist(0,R),R)$.

If the two paths did not converge after the first hop, in the worst case, the two messages move in opposite directions, and the distance between the two nodes reached by the two messages is $d1 = d0 + 2hop\_dist(0,R)$; in the best case, $d1 = max(0, d0 - 2hop\_dist(0,R))$; and since with equal probability the hop may move in all directions, $d1 = d0$ in the average case.

Since the expected distance of the second hop traveled in Pastry routing is $hop\_dist(1,R)$, the probability that after second hop, the two paths converge into the same node is $prob\_hop(1,d0,R) = S(hop\_dist(1,R),d1,R)/S_{surface}(hop\_dist(1,R),R)$.

The analysis for subsequent hops is analogous. □