Exploiting network proximity in 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.

Abstract

Generic 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. These systems 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. In this paper, we present a comprehensive study of the network locality properties of a p2p overlay network. 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. A simple heuristic measures a scalar proximity metric among a small number of nodes, incurring 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 study the impact of proximity-based routing on the load balancing in the p2p overlay.

1 Introduction

Several recent systems (CAN [6], Chord [10], Pastry [7] and Tapestry [14]) provide a self-organizing substrate for large-scale peer-to-peer applications. These systems can be viewed as providing 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, for instance, does not currently 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 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 exploit locality by measuring a proximity metric among pairs of nodes, and by choosing nearby nodes for inclusion in their routing tables. Early results for the resulting locality properties are promising. In Tapestry and Pastry, for instance, the average total “distance” traveled by a message is only a small and constant factor larger than the “distance” between source and destination in the underlying network. However, these results come at the expense of 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.

To address these questions, this paper presents results of a comprehensive study of Pastry’s locality properties via analysis and via simulations based on two large-scale Internet topology models. Moreover, we propose an improved node join and failure protocol that substantially decreases the overlay maintenance cost relative to the original implementation [7], at the expense of a negligible reduction in the quality of the Pastry’s routing properties. The results indicate that the locality properties are robust on a variety of network topology models. Moreover, the load imbalance caused by the proximity based routing is modest, and hot spots can be easily dispersed without affecting overall route quality. While our analysis and simulations are based on Pastry, many of our results apply to Tapestry and Chord as well. We conclude that it is possible to exploit network proximity in p2p overlay networks with low overhead and without compromising their self-organizing and load balancing properties.

The rest of this paper is organized as follows. Related work is discussed in Section 2. In Section 3, we provide a brief overview of the Pastry protocol. Pastry’s locality properties, and the new protocols for node joining and failure recovery are presented in Section 4. An analysis of Pastry’s locality properties follows in the Section 5. Section 6 presents experimental results, and we conclude in Section 7.

2 Related work

CAN [6], Chord [10] and Tapestry [14] are closely related to Pastry. Each of these protocols form a self-organizing overlay network and provide a load-balanced, fault-tolerant distributed hash table, in which items can be inserted and looked up in a bounded number of forwarding hops. CAN, Tapestry and Pastry each use heuristics to exploit proximity in the Internet, and the resulting properties have been studied in prior work. To the best of our knowledge, this paper is the first study that looks at both costs and benefits of proximity based routing in a p2p overlay, and considers the impact of node failures on those costs and benefits.

Pastry and Tapestry are related to the work by Plaxton et al. [5] and to routing in the landmark hierarchy [12]. The approach of routing based on address prefixes, which can be viewed as a generalization of hypercube routing, is common to all these schemes. However, neither Plaxton nor the landmark approach are fully self-organizing. Pastry and Tapestry differ in their approach to locating the numerically closest node in the sparsely populated nodeId space, and for managing replication. Pastry uses overlapping sets of neighboring nodes in the nodeId space (leaf sets), both to locate the destination in the final routing hop, and to store replicas of data items for fault tolerance. Tapestry uses a different concept called surrogate routing to locate the destination, and it inserts replicas of data items using different keys. The approach to achieving network locality is very similar in both systems.

The Chord protocol is closely related to Pastry and Tapestry, but instead of routing based on address prefixes, Chord forwards messages based on numerical difference with the destination address. Unlike Pastry and Tapestry, Chord currently makes no explicit effort to exploit network proximity. However, locality heuristics similar to the ones used in Pastry could presumably be added to Chord.

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. Unlike Pastry, Tapestry and Chord, the CAN routing table does not grow with the network size, but the number of routing hops grows faster than $logN$. The work on CAN [6] explored two techniques to improve routing performance by using information about the network topology. In the first technique, each node measures the RTT to each of its neighbors and messages are forwarded to the neighbor with the maximum ratio of progress to RTT. This technique differs from the one used in Pastry because the set of neighbors of a node is chosen without regard to their proximity; this has the disadvantage that all neighbors may be quite far from the node. The second technique measures the distances between each node and a set of landmark servers to compute the coordinates of the node in the CAN space such that neighbors in the CAN space are topologically close. This technique can achieve good performance but it has the disadvantage that it is not fully self-organizing; it requires a set of well-known landmark servers. In addition, it may cause significant imbalances in the distribution of nodes in the CAN space that lead to hotspots.

Existing applications built on top of Pastry include PAST [8] and SCRIBE [9]. Other peer-to-peer applications that were built on top of generic routing and location substrates are OceanStore [3] (Tapestry) and CFS [2] (Chord).

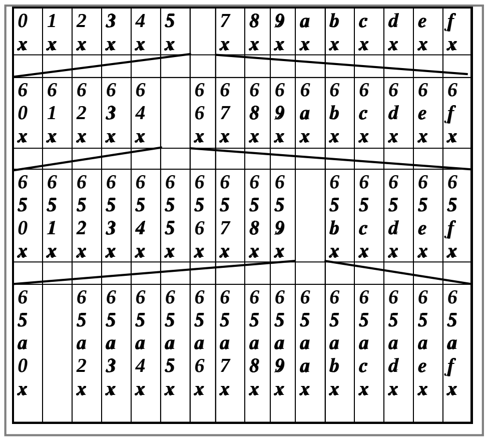
3 Pastry

Pastry is described in detail in [7], and a brief summary is provided here. Pastry is a generic, efficient, 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 application of changes in the set. These capabilities can be used to build a distributed, fault-tolerant hashtable, which in turn can be used to support a variety of decentralized, distributed applications.

Assuming a network consisting of N nodes, the expected number of forwarding hops to deliver a messages with a random key is $< \lceil log_{2^b} N \rceil$ (b is a configuration parameter with typical value 4). The tables required in each node have only $O(log_{2^b} N)$ entries, where each entry maps an existing nodeId to the associated node's IP address. Upon a node failure or the arrival of a new node, the invariants in all affected tables can be restored by exchanging $O(log_{2^b} N)$ messages. In the following paragraphs, we briefly sketch the Pastry routing scheme.

Node state: For the purposes of routing, nodeIds and keys are thought of as a sequence of digits in base 2^b. A node's routing table is organized into $128/2^b$ rows and $2^b - 1$ columns. The $2^b - 1$ entries in row n of the routing table refer to nodes whose nodeIds share the first n digits with the present node's nodeId; the $n + 1$th nodeId digit of a node in column m of row n equals m. The column in row n corresponding to the value of the $n + 1$'s digits of the local node's nodeId remains empty. Figure 1 depicts an example routing table.

A routing table entry is also 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



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. The IP address associated with each entry is not shown.

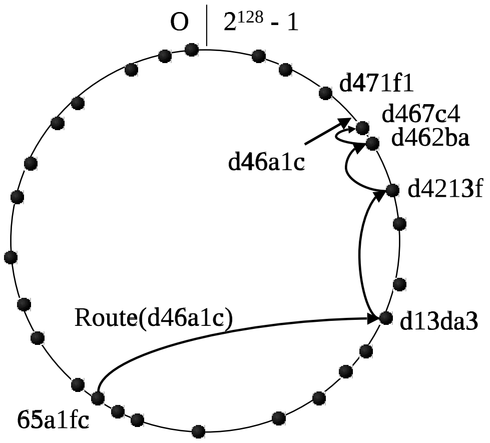


Figure 2: Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

populated in the routing table. Each node maintains IP addresses for the nodes in its *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 $\lceil 8 * log_{16} N \rceil$.

Message routing: At each routing step, a node normally forwards 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 prefix that the key shares with the present node's id. If no such node is known, 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. 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[1]. 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. A deterministic upper bound for the number of routing hops is 128/b+1, assuming correct routing tables and no concurrent node failures. Moreover, simulation shows that the routing is highly resilient to node failures.

To achieve self-organization, Pastry must dynamically maintain its node state, i.e., the routing table and leaf set, in the presence of new node arrivals, node failures, node recoveries, and network partitions.

[1]The last clause takes care of a pathological case where the numerically closest node does not share a nodeId prefix with the key.

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

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.

Node addition: A newly arriving node with the new nodeId X can initialize its state by asking any existing Pastry node A to route a special message using X as the key. The message is routed to the existing node Z with nodeId numerically closest to X. X then obtains the leaf set from Z and the ith row of the routing table from the node encountered along the route from A to Z whose nodeId matches X in the first $i - 1$ digits. Using this information, X can correctly initialize its own routing table and leaf set. Finally, X announces its presence to the initial members of its leaf set, which in turn update their own leaf sets and routing tables. Given an appropriate leaf set size (e.g., $> 2^b$), one can show that, with high probability, all nodes whose routing tables are affected by X's arrival are notified.

Node failure: When a node fails, the leaf sets and routing tables of a number of other nodes must be updated. Leaf set memberships is actively maintained. The members of each leaf set periodically exchange keep-alive messages. If a node is unresponsive for a period T, it is presumed failed. All members of the failed node's leaf set are then notified and they update their leaf sets. Since the leaf sets of nodes with adjacent nodeIds overlap, this update is trivial.

Routing table entries that refer to failed nodes are repaired lazily. During message forwarding, when a routing table entry is found that is either empty or the referenced node has failed, Pastry routes the message to another node with numerically closer nodeId (lines 11-13 in Figure 3). 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.

Node recovery: The node recovery protocol is optimized to reduce the overhead of temporary node failures. A recovering Pastry node first attempts to contact nodes in its last known leaf set and obtains their leaf sets. If the numerical range of nodeIds in one of those sets still includes the recovering node's nodeId, the node updates its own leaf set based on the information it receives and then notifies the current members of its leaf set of its presence. Otherwise, the node follows the normal protocol for node addition.

Network partitions: A network partition can cause the apparent simultaneous failure of a large number of nodes at random points in the nodeId space. In extreme cases, such a partition could cause the failure of $l/2$ nodes with adjacent nodeIds. This (rare) case requires a special recovery procedure, since live nodes that are separated by $l/2$ or more failed nodes in the nodeId space are not aware of each other. Briefly, the live nodes at the edges of such a sequence of failed nodes locate each other by sending messages towards the other using their remaining live routing table entries, then form a new leaf set.

4 Pastry locality properties

This section focuses on Pastry's locality properties. Pastry seeks to exploit proximity in the underlying Internet, by routing through as short a path as possible, finding nearest copies of objects, etc. It relies on a scalar proximity metric that indicates the "distance" between any given pair of Pastry nodes. It is assumed that each Pastry node can measure or otherwise obtain the distance between itself and any node with a known IP address. Furthermore, it is assumed that the proximity metric reflects static properties of the underlying physical network, rather than prevailing traffic conditions.

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, metrics such as round-trip time (minimum of a series of pings), bandwidth (measured, for instance, using packet pair techniques), the number of IP routing hops (measured using traceroute), or some combination thereof could be used. Choosing an appropriate proximity metric for p2p overlay networks is the subject of future work and beyond the scope of this paper.

Pastry's locality properties derive from its attempt to minimize the distance, according to the proximity metric, to each of the nodes that appear in a node's routing table, subject to the constraints imposed on nodeId prefixes. It is expensive to achieve this goal precisely 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 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. Analysis and simulations on two Internet topology models presented in Section 6 confirm this.

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 [9] 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 [8].

An analysis of these properties follows in Section 5. Simulation results that confirm and quantify these properties on two Internet topology models follow in Section 6.

4.1 Node addition and failure

Next, we present the Pastry node join protocol and show how this protocol maintains the proximity invariant.

First, recall from Section 3 that a new node X must contact an existing Pastry node A when joining the system. 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 X, according to the proximity metric.

First, consider the top row of X's routing table, obtained from node A. Assuming the triangulation 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 triangulation 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 Recall that 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 (lines 11-13 in Figure 3). 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.

4.2 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 triangulation 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.

4.3 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 an 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.

In this section, we have shown at an intuitive level why the Pastry node join protocol preserves the proximity invariant, and how Pastry's locality properties can be derived from the proximity invariant. However, as part of this argument, we have relied on a few assumptions that do not generally hold in the Internet. For instance, the triangulation 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 using simulations on Internet topology models. Results of simulations based two Internet topology models will be presented in Section 6.

```
(1)  discover(seed)
(2)      nodes = getLeafSet(seed)
(3)      forall node in nodes
(4)          nearNode = closerToMe(node,nearNode)
(5)      depth = getMaxRoutingTableLevel(nearNode)
(6)      while (depth > 0)
(7)          nodes = getRoutingTable(nearNode,depth - -)
(8)          forall node in nodes
(9)              nearNode = closerToMe(node,nearNode)
(10)     end while
(11)     do
(12)         nodes = getRoutingTable(nearNode,0)
(13)         currentClosest = nearNode
(14)         forall node in nodes
(15)             nearNode = closerToMe(node,nearNode)
(16)     while (currentClosest != nearNode)
(17)     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, with the appropriate nodeId prefix. 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. Due to space constraints, the details of the analysis and the proofs are omitted here; they are avaliable at http://dosa.ecn.purdue.edu:8080.

5.1 Route probability matrix

The analysis of the distribution of the number of routing hops is 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, [1]). 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

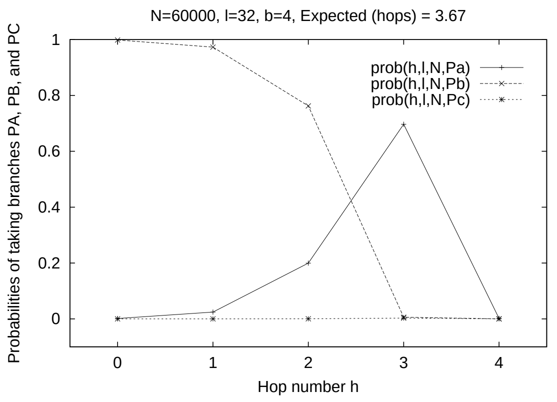


Figure 5: Probabilities $Pr(h,l,N,P_A)$, $Pr(h,l,N,P_B)$, $Pr(h,l,N,P_C)$ and expected number of hops for $N = 60000$, with $l = 32$ and $b = 4$. (From analysis.)

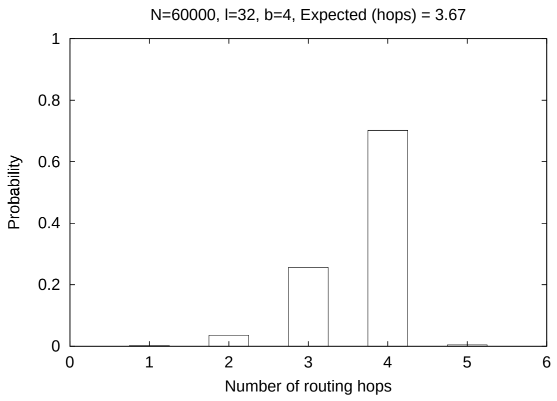


Figure 6: Distribution of the number of routing hops per message for $N = 60,000$, with $l = 32$ and $b = 4$. (From analysis.)

derive the expected number of routing hops in Pastry routing according to the following theorem.

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

Next, we analyze the expected distance a message travels in the proximity space, while it is being routed through Pastry. 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.

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 properties when two random Pastry nodes send a message with the same randomly chosen key. Specifically, we are interested in the distance the 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.

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.

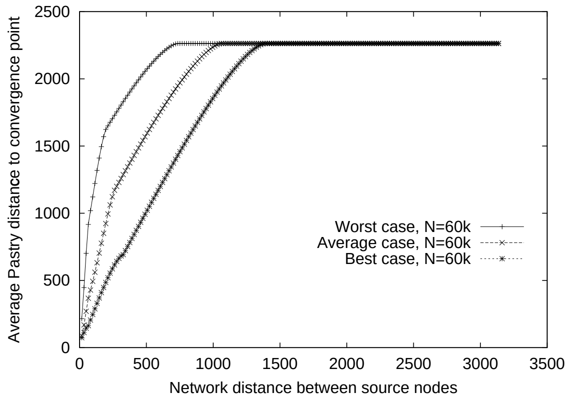


Figure 7: Distance among source nodes routing messages with the same key, versus the distance traversed until the two paths converge, for a 60,000 node Pastry network, with l=32 and b=4. (From analysis.)

6 Experimental results

In this section, we present experimental results quantifying Pastry's locality properties. All results were obtained using a Pastry implementation running on top of a network simulator. 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 triangulation inequality.

A second topology was generated using the Georgia Tech transit-stub network topology model [13]. 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 triangulation inequality does not hold for RTTs among nodes in the topology model.

Finally, we used the Mercator topology and routing models [11]. The topology model contains 102,639 routers and it was obtained from real measurements of the Internet using the Mercator program [4]. The authors of [11] 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.

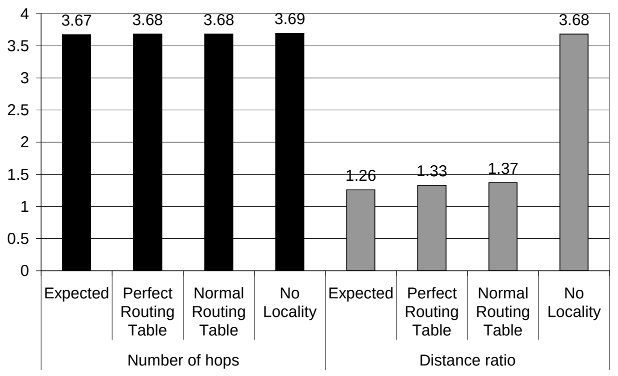


Figure 8: Number of routing hops and distance ratio, sphere topology.

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 proximity based routing, and the effectiveness of Pastry's heuristics.

6.3 Routing distance

Figure 9 shows the distance messages travel in each consecutive routing hops. 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 aver-

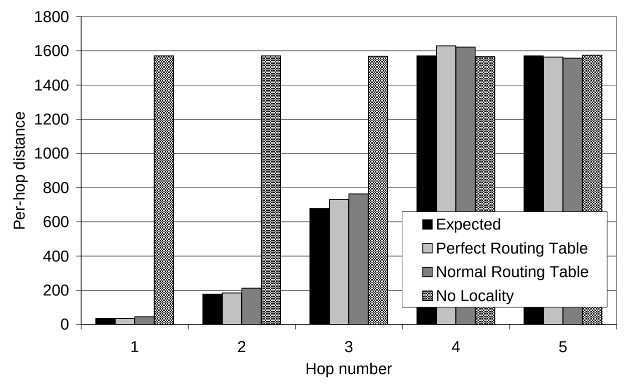


Figure 9: Distance traversed per hop, sphere topology.

age 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).

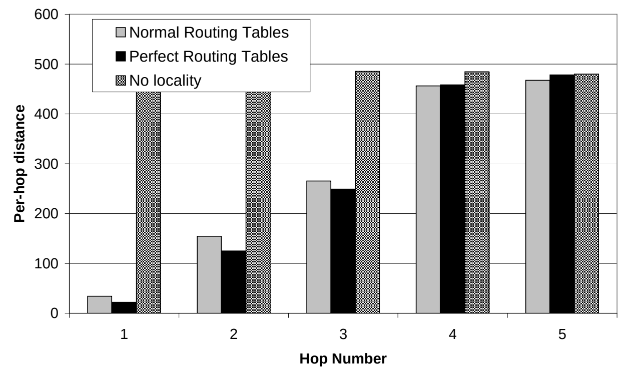


Figure 10: Distance traversed per hop, GATech topology.

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.

The proximity metric used with the Mercator topology makes Pastry's locality properties 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, we chose to include these results to demonstrate that Pastry's locality properties are good even

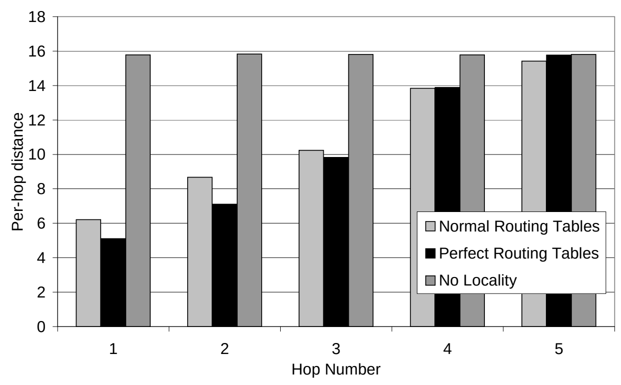


Figure 11: Distance traversed per hop, Mercator topology.

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 acceptable results, with a mean distance ration of 2.2 and a maximum of about 6.5.

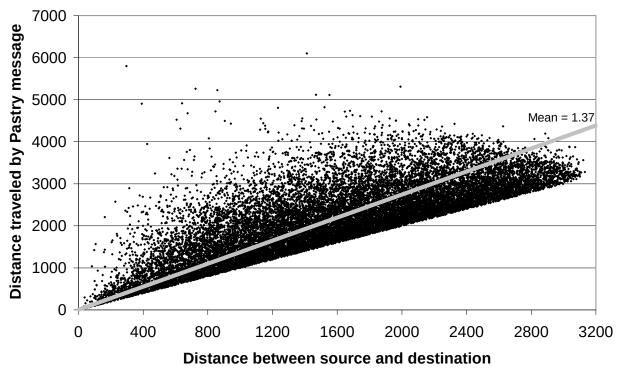


Figure 12: Distance traversed versus distance between source and destination, sphere topology.

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

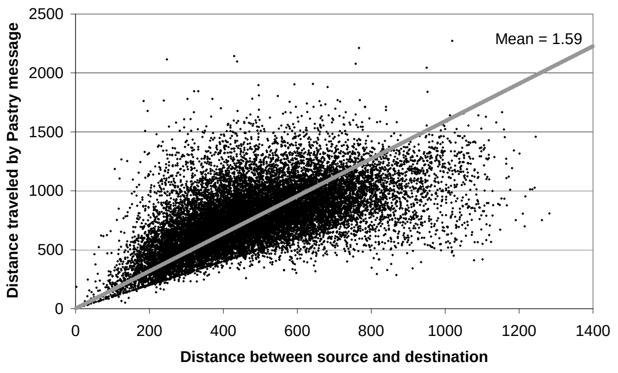


Figure 13: Distance traversed versus distance between source and destination, GATech topology.

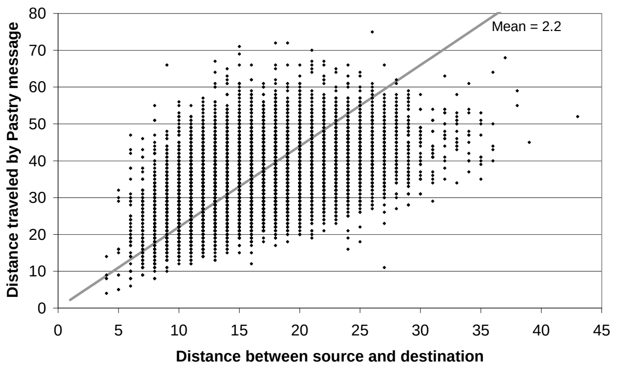


Figure 14: Distance traversed versus distance between source and destination, Mercator topology.

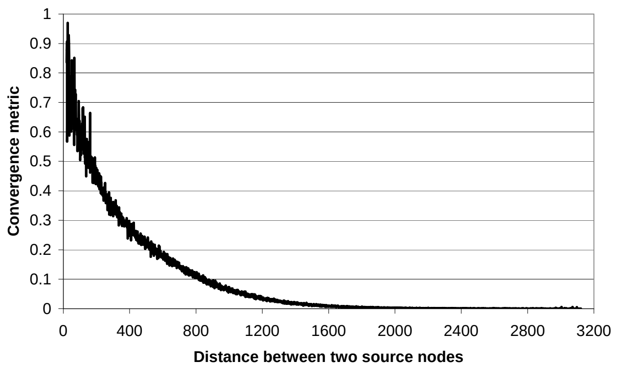


Figure 15: Convergence metric versus the distance between the source nodes, sphere topology.

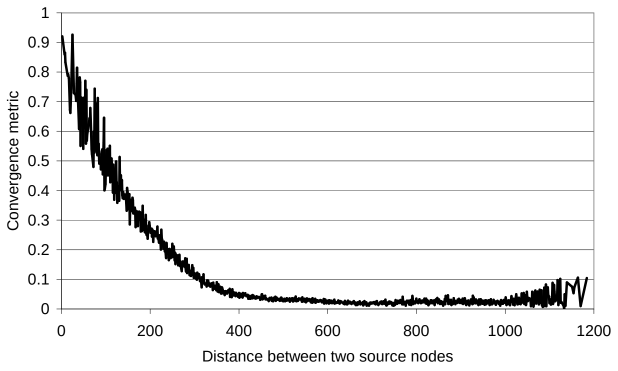


Figure 16: Convergence metric versus distance between the source nodes, GATech topology.

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.

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 [8, 9].

Figure 18 shows the average distance traveled from the source nodes to the node where the paths converge, as a function of the distance between the sources nodes. Note that the convergence node could be the destination. We included this graph for the sphere topology, as this allows a direct comparison with the results of the analysis (Figure 7). The results match well.

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 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.

Figure 19 shows the maximum, mean and minimum number of probes performed by a node joining the Pastry network. The results were generated for Pastry networks of between 1,000 and 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.

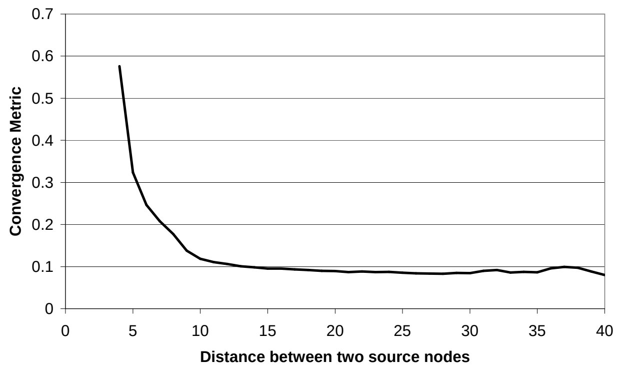


Figure 17: Convergence metric versus distance between the source nodes, Mercator topology.

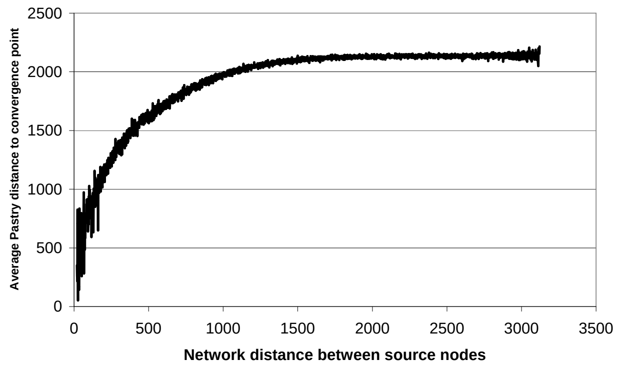


Figure 18: Distance to node where message paths converge, versus the distance between the two source nodes, sphere topology.

Figure 20 shows the corresponding number of probes performed by nodes other than the joining node during a join.

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 4.2). Recall that leaf set repair is instantaneous,

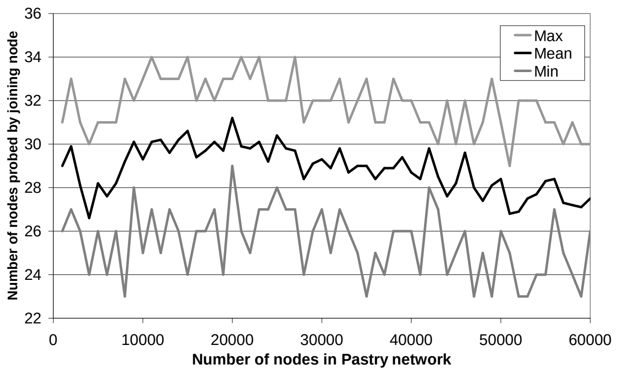


Figure 19: Number of probes performed by a newly joining node for Pastry networks between 1,000 and 60,000 nodes, sphere topology.

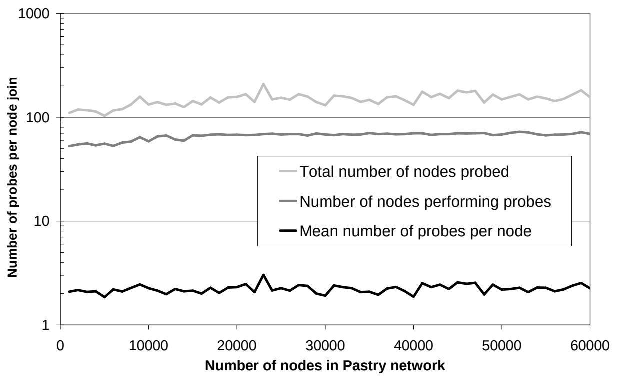


Figure 20: Number of probes per-join performed by nodes other than the joining node for Pastry networks between 1,000 and 60,000 nodes, sphere topology.

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 21 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.

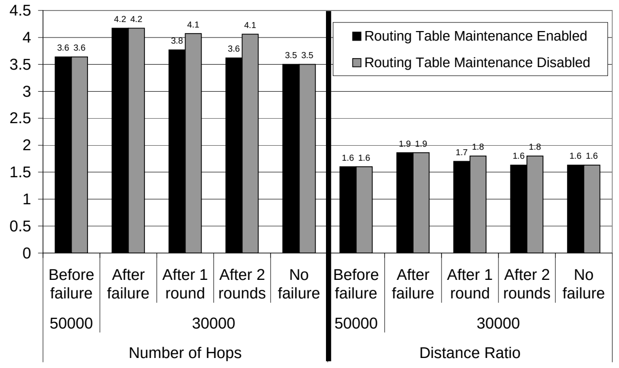


Figure 21: Routing hops and distance ratio for a 50,000 node Pastry overlay when 20,000 nodes simultaneously fail, GATech topology.

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 numerically closer node (third branch of the routing algorithm). 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 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.

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. However, when many large overlay networks perform probing in the Internet, there can be a significant burden on the network. For this reason, several project are currently working on a generic measurement infrastructure for the Internet. We expect that this work will provide a solution to this problem in the long term.

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 based routing 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 22 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 delay penalty 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.

6.8 Discovering a nearby seed node

Finally, we evaluate the discovery algorithm used to find a nearby Pastry node, presented in Section 4.3. 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 the joining node that wishes to locate a nearby Pastry node, the other is treated as the seed Pastry node known to the joining

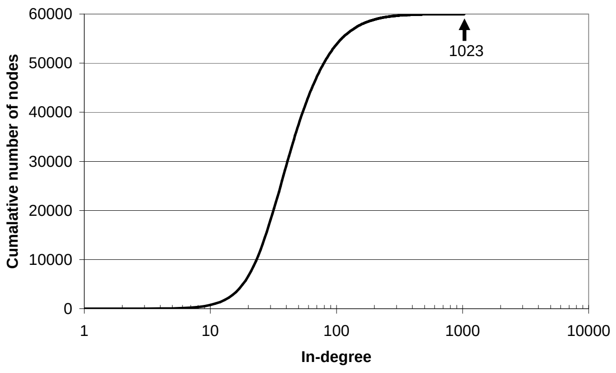


Figure 22: Indegree distribution of 60,000 Pastry nodes, GATech topology.

	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.

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.

7 Conclusion

This paper presents an analysis and an experimental evaluation of the network locality properties of a p2p overlay network, Pastry. Analysis shows that good network locality properties can be achieved with very low overhead in synthetic network topologies. A refined protocol for node joining and node failure recovery significantly reduces the overhead of maintaining a topology-aware overlay. Simulations on two different Internet topology models then show that these properties hold also in more realistic network topologies. We conclude that exploiting network proximity can be accomplished effectively and with low overhead in a self-organizing peer-to-peer overlay network.

References

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

[2] 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.

[3] J. K. et al. Oceanstore: An architecture for global-scale persistent store. In *Proc. ASPLOS'2000*, November 2000.

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

[5] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32:241–280, 1999.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *SIGCOMM'88*, Stanford, CA, 1988.

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

[14] 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.