Proximity neighbor selection in tree-based structured peer-to-peer overlays

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

[1]Microsoft Research, 7 J J Thomson Avenue, 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-2003-52

Structured peer-to-peer (p2p) overlay networks provide a useful substrate for building distributed applications. They assign object keys to overlay nodes and provide a primitive to route a message to the node responsible for a key. Proximity neighbor selection (PNS) can be used to achieve both low delay routes and low bandwidth usage but it introduces high overhead. This paper presents a detailed evaluation of PNS and heuristic approximations. We describe a new heuristic called constrained gossiping (PNS-CG) and show that it achieves performance similar to perfect PNS with low overhead. We also compare constrained gossiping with previous heuristics and show that it achieves better performance with lower overhead.

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

1 Introduction

Structured p2p overlay networks like CAN, Chord, Pastry, Tapestry, and others [1, 2, 3, 4, 5, 6] provide a useful substrate for building distributed applications [7, 8, 9, 10] because they are scalable, self-organizing, and reliable. They assign application-defined keys to overlay nodes and provide a primitive to route a message to the node responsible for a key. Structured overlays conform to a specific graph structure that allows them to route in $O(\log N)$ hops while maintaining at most $O(\log N)$ routing state where N is the number of nodes in the overlay.

It is important for overlay routing to exploit proximity in the underlying network. Otherwise, each overlay hop has an expected delay equal to the average delay between a pair of random overlay nodes, which stretches route delay by a factor equal to the number of overlay hops and increases the stress in the underlying network links. There are several techniques for proximity-aware routing proposed in the literature [11, 4, 1, 3, 10, 12]. Recent work [13, 14, 15] identifies proximity neighbor selection (PNS) as the most promising technique.

PNS can be used to achieve low delay routes and low bandwidth usage. It selects routing state entries for each node from among the closest nodes in the underlying topology that satisfy constraints required for overlay routing. PRR [11] used PNS first but it assumed knowledge of the delay between each node and all the nodes that could potentially be in that nodes routing state, which is expensive to obtain in large-scale dynamic systems. Tapestry [4, 10] and Pastry [3] proposed heuristics to approximate PNS requiring less delay measurements.

This paper presents a detailed evaluation of PNS and two heuristic approximations. We describe a new heuristic called *constrained gossiping* (PNS-CG) that achieves performance very close to perfect PNS with lower overhead than the heuristics proposed for Tapestry and Pastry. In particular, it reduces the number of messages and bandwidth required by the original Pastry heuristic [3] and eliminates Pastry's neighborhood set. The original Pastry heuristic assumed an oracle for finding a nearby seed node for joining. PNS-CG provides an efficient algorithm that takes a random overlay node and returns a nearby seed node for joining. This algorithm is interesting because it does not require any additional state beyond that which is already maintained for overlay routing. We describe constrained gossiping in the context of Pastry but it is applicable to other tree-based structured overlays like PRR and Tapestry.

We present a comparison between PNS and proximity-unaware routing based on results obtained via analysis and simulation with three realistic topology models. Our analysis differs from previous ones [11, 10] by providing precise delay stretch estimates instead of asymptotic bounds.

We also present a detailed comparison between PNS, PNS-CG, and PNS(K) [15] on Pastry. This comparison studies both the extent to which the two heuristics can approximate PNS and the overhead that they introduce. It is based on simulation results obtained with the three topology models, varying overlay sizes and varying router parameters (e.g., the number of bits fixed by each overlay routing hop). The results indicate that PNS-CG can achieve performance closer to PNS than PNS(K) and with lower overhead.

The rest of this paper is organized as follows. We begin with an overview of Pastry with perfect PNS in Section 2 then section 3 presents constrained gossiping. Section 4 presents an evaluation of PNS, constrained gossiping, and PNS(K). We conclude in Section 5.

2 Perfect PNS in Pastry

Each Pastry node has a unique *nodeId* that is selected randomly with uniform probability from a circular 128-bit identifier space. Keys are also selected from the same space and the root node for a key is the live node whose nodeId is numerically closest to the key. Pastry provides a primitive to send a message to a destination key. These messages are delivered to the key's root node.

2.1 Node state

The routing state maintained by each node consists of the *routing table* and the *leaf set*. Each entry in the routing state contains the nodeId and IP address of a node. NodeIds and keys are interpreted as unsigned integers in base 2^b (where b is a parameter with typical value 4).

The routing table is a matrix with $128/b$ rows and 2^b columns. The entry in row r and column c of the routing table contains a nodeId that shares the first r digits with the local node's nodeId, and has the $(r+1)$th digit equal to c. If there is no such nodeId, the entry is left empty. The uniform random distribution of nodeIds ensures that only $log_{2^b} N$ rows have non-empty entries on average. Figure 1 depicts a sample routing table. This routing table is similar to those used by PRR [11] and Tapestry [4].

The leaf set contains the $l/2$ closest nodeIds clockwise from the local nodeId and the $l/2$ closest nodeIds counterclockwise. The leaf set ensures reliable message delivery and is used to store replicas of application objects.

2.2 Message routing

At each routing step, the local node normally forwards the message to a node whose nodeId shares a prefix with the key that is at least one digit longer than the prefix that the key shares with the local node's nodeId. If no such node is known, the message is forwarded to a node whose nodeId is numerically closer to the key and shares a prefix with the key at least as long. If there is no such node, the message is delivered to the local node. Figure 2 shows the path of a message and Figure 3 shows the pseudo code for the routing algorithm.

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.

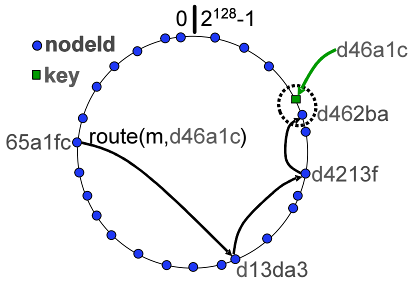

Figure 2: Routing a message from node $65a1fc$ with key $d46a1c$. The dots depict live nodes in Pastry's circular namespace.

if ($k.isBetween(L_{-l/2}, L_{l/2})$)
  // use the leaf set
  forward to L_i such that $|k - L_i|$ is minimal
else
  // use the routing table
  let $r = shl(k, n)$
  if ($R_r^{k_r}$ exists and is live)
    forward to $R_r^{k_r}$
  else
    // rare case
    forward to $t \in L \cup R$ such that
      $shl(k, t) \geq r \wedge |t - k| < |n - k|$

Figure 3: Pastry routing procedure, executed when a message with key k arrives at a node with nodeId n. R_r^i is the entry in the routing table R at column i and row r. 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 nodeIds at the edges of the local leaf set. k_r represents the rth digit in the key k. $shl(k, n)$ is the length of the prefix shared between k and n in digits.

The algorithm guarantees that the message is delivered to the key's root unless all $l/2$ nodes in one half of the leaf set have failed simultaneously (regardless of the state of routing tables). Therefore, good fault tolerance requires a leaf set that is large enough to make the probability of such simultaneous failures small. For example, a size of 16 to 24 provides good fault tolerance up to large values of N, which can be further improved by the leaf set repair mechanism in [16].

2.3 Proximity neighbor selection

The flexibility in the choice of nodeIds to fill routing table slots can be exploited to implement PNS effectively. Since any nodeId with the required prefix can be used to fill a slot, proximity neighbor selection picks the closest node in the underlying network from among those whose nodeIds have the required prefix. The proximity metric that is typically used in the definition of closest is round trip delay.

This technique was first proposed in PRR [11]. It is expensive to implement perfect PNS in a large dynamic system but we describe an heuristic that can achieve similar performance with low overhead in the next section.

With perfect PNS, the expected distance traveled in the initial routing hop is small and it tends to increase exponentially at each consecutive routing step. This happens because the density of nodes tends to decrease exponentially with the length of the prefix match between their nodeId and the destination key. From this observation, one can derive two important properties:

Low 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 value that is mostly independent of the overlay size.

Local route convergence: The routes 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. This happens because the messages travel exponentially larger distances at each consecutive routing hop towards an exponentially shrinking set of nodes. Thus the probability of route convergence increases in each step even when earlier (shorter) routing steps moved the messages farther apart. This result has significance for caching applications layered on Pastry. A copy of a popular object requested by a node n and cached by all nodes along the Pastry route is likely to be retrieved from a nearby node when requested by a node close to n. This property is also exploited in Scribe [8] to achieve low link stress in application level multicast, and to implement nearest-member anycast [17].

We present an analysis of delay stretch of PNS in Pastry in the Appendix. This analysis differs from the one in [11] because it provides a precise estimate instead of asymptotic bounds and it works for arbitrary topologies that may fail to satisfy the conditions assumed in [11].

We derive a closed-form expression for the average delay stretch when messages are sent from nodes chosen randomly with uniform probability from the overlay to keys chosen randomly with uniform probability from the id space. To compute the average delay stretch, we characterize topologies using the function $D(k)$ — the average over all nodes p of $D(p, k)$, which returns the average delay from p to its k closest nodes in the underlying network.

This analysis is useful to provide an insight into the characteristics of topologies that affect the performance of PNS. Additionally, we may use approximations of $D(k)$ to predict performance ahead of deployment.

3 Constrained gossiping

Constrained gossiping (PNS-CG) is a new heuristic that can approximate proximity neighbor selection with low overhead. It consists of new node join and overlay maintenance protocols that reduce the overhead relative to the original Pastry protocols [3] and a new algorithm that uses the routing state already maintained by Pastry to locate nearby seed nodes for joining.

3.1 Node join

When joining the overlay, a new node x with nodeId X must contact an existing overlay 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 argue that x's resulting routing table is nearly perfect provided node a is the closest overlay node to x according to the proximity metric. The closest node can be found using expanding ring IP multicast in some applications or the algorithm described later.

First, consider the top row of x's routing table, which is obtained from node a. Assuming that the triangle inequality holds in the proximity space, the entries in the top row of a's routing table should also be 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 in the first $n-1$ digits. Therefore, if the triangle inequality holds, the entries in the nth row of a_n's routing table should also be close to x.

It is also important to update other node's routing tables to ensure that they remain near perfect after new nodes join the overlay. 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 gossip 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.

This procedure provides a very restricted form of gossiping. It ensures that routing tables remain near perfect because 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. These nodes should learn about x's arrival because x may displace a more distant node in their routing tables. Conversely, a node 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.

3.2 Node failure

Failed routing table 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.

This procedure also preserves near perfect routing tables. The downstream node is usually 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 low because all three nodes were part of the same group of nearby nodes with identical nodeId prefix.

If no replacement node is supplied by the downstream node, a replacement is found by triggering the routing table maintenance task, which is described next.

3.3 Routing table maintenance

We also define a periodic routing table maintenance protocol that is another form of restricted gossiping designed both to repair failed entries and to ensure that routing table entries remain near perfect to prevent a slow deterioration of the locality properties over time. 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 idea behind this maintenance procedure is to gossip routing information among groups of nearby nodes with identical nodeId prefix. If a nearby node with the appropriate prefix is known to at least one member of the group, the procedure ensures that the entire group will eventually learn about the node and will adjust their routing tables accordingly.

3.4 Locating a nearby seed node

Recall that for the node join algorithm to achieve near-perfect routing tables, the starting node a should be the closest over-

```
discover(seed)
    nodes = getLeafSet(seed)
    nearNode = pickClosest(nodes)
    depth = getMaxRoutingTableLevel(nearNode)
    closest = nil
    while (closest != nearNode)
      closest = nearNode
      nodes = getRoutingTable(nearNode,depth)
      nearNode = pickClosest(nodes)
      if (depth > 0) depth = depth-1
    end
    return closest
```

Figure 4: Algorithm to locate closest overlay node. *seed* is the overlay node initially known to the joining node.

lay node to the new node x. The original Pastry [3] heuristic assumed an oracle that returned the closest overlay node to x. This oracle could be implemented using, for example, the algorithm in [18] but this would require maintaining additional state.

In Figure 4, we present a new algorithm to find an approximation to the closest overlay node to x given any seed node in the overlay. This algorithm is interesting because it does not require any additional state beyond the routing table and leaf set that are already maintained by Pastry nodes.

The algorithm exploits the property that location of the nodes in the seed's leaf set is 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. This performs a constant number of probes at each level but the probed nodes get exponentially closer at each step. The last phase repeats the process for the top level until no more progress is made.

To avoid falling into local minima, the process is reseeded from a new random node until the distance to the closest node found is below a threshold (currently the average distance between the nodes contacted and their closest neighbors) or up to a maximum number of times (currently 5).

Our experimental evaluation shows that this algorithm is efficient and returns a node whose distance to x is almost as small as the distance to the closest node.

4 Experimental results

In this section, we present experimental results quantifying the performance of PNS, PNS-CG, and PNS(K) in Pastry. The results were obtained using a Pastry implementation running on top of a network simulator.

4.1 Experimental setup

We used three network topology models. Each topology has a *core* set of routers and we ran Pastry on end nodes that were randomly assigned to routers in the core with uniform probability. Each end node was directly attached by a LAN link with a delay of 1ms to its assigned router.

GATech is a transit-stub topology generated with the Georgia Tech [19] topology generator. This topology has 5050 routers arranged hierarchically. There are 10 transit domains at the top level with an average of 5 routers in each. Each transit router has an average of 10 stub domains attached, with an average of 10 routers each. We did not assign overlay nodes to transit routers. The delay between core routers is computed by the topology generator and routing is performed using the routing policy weights of the graph generator. As in the real Internet, the triangle inequality does not hold for a signifcant fraction of triples of nodes in this topology. Pastry uses the round-trip delay (RTT) between two nodes as its proximity metric.

Mercator is a topology with 102,639 routers. It was obtained from real measurements of the Internet using the Mercator system [20] and it uses hierarchical routing as in the Internet. Since the Mercator topology is not annotated with delay information, Pastry uses the number of network-level (IP) routing hops between two nodes as a proxy for delay.

CorpNet is a topology with 298 routers and is generated using real measurements of the world-wide Microsoft corporate network. The network distance in this topology is the minimum round-trip delay.

We compared four versions of Pastry: *no locality* builds routing tables without taking into account network distance, PNS uses global knowledge in the simulator to implement perfect PNS, PNS-CG uses constrained gossiping to approximate PNS, and PNS(16) [15] approximates PNS by probing at most 16 random nodes for each routing table slot. The comparison between the first two provides an upper bound on the benefit of using PNS. The comparison between the last three evaluates the extent to which the two heuristics can approximate the performance of PNS and their overhead. We also compare the performance of PNS with *predicted*, which is the value predicted by our analysis. There are no failures in our experiments, and routing table maintenance was disabled. We evaluate routing table maintenance and failures in [21, 16].

4.2 Delay stretch with varying N

The first experiment routed 200,000 lookup messages from randomly chosen nodes to randomly chosen keys using the four different Pastry versions. We ran this experiment in the three network topologies with $b = 4$, $l = 16$, and a varying number of Pastry nodes.

PNS: Figure 5 compares the delay stretch without locality and with perfect PNS. It provides an upper bound on the benefit of PNS in the three topologies. The delay stretch achieved

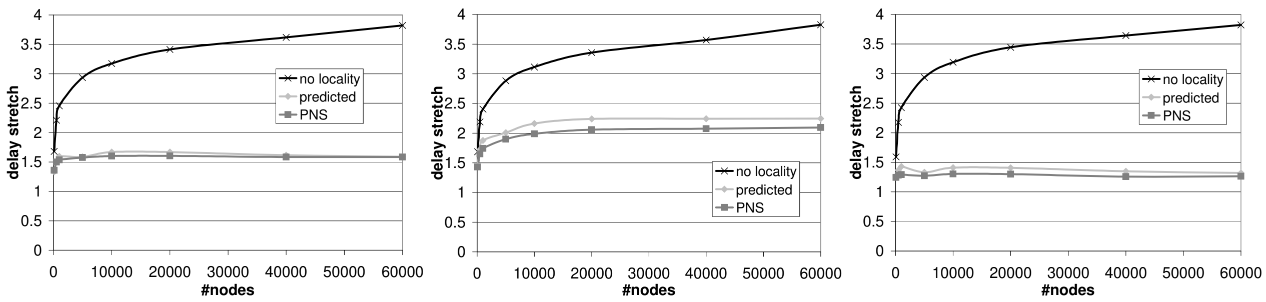

Figure 5: Delay stretch with $b = 4$, $l = 16$, and varying N for GATech, Mercator and CorpNet.

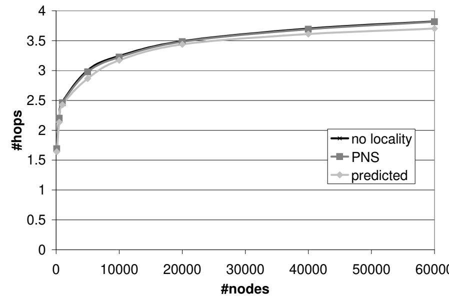

Figure 6: Number of Pastry hops with $b = 4$, $l = 16$, and varying N.

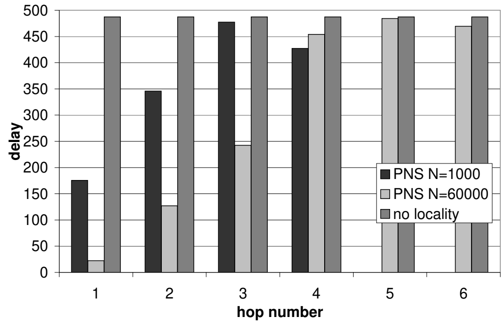

Figure 7: Delay in the i-th hop in a Pastry route in GATech with $b = 4$, $l = 16$, and varying N.

without locality is up to 3 times worse than with the PNS in Figure 5. Furthermore, the difference grows with the number of nodes in the overlay. This speaks to the effectiveness of proximity neighbor selection.

Figures 6 and 7 explain the difference in the delay stretch achieved with and without proximity-aware routing. Figure 6 shows the number of Pastry routing hops as a function of N. Figure 7 shows the average delay of the i-th hop in a Pastry route for regular Pastry with different values of N, and for the version of Pastry with no locality.

The expected network delay in each hop without proximity-aware routing is constant and equal to the average delay between two random nodes in the network. Therefore, the average delay stretch without proximity-aware routing is equal to the number of Pastry hops and it grows logarithmically with N as predicted by the analysis. This can be observed by comparing Figures 6 and 5.

The number of hops is virtually identical with and without proximity-aware routing. Yet, Figure 5 shows that the delay stretch achieved by PNS is largely independent of the number of nodes in the overlay. This is because the increase in the number of hops as N increases is offset by a decrease in the delay of the first hops along the Pastry route. Figure 7 illustrates this effect for $N = 1000$ and $N = 60000$. The increased number of end nodes attached to the core results in additional nearby nodes to choose from when filling routing table slots at the top levels of routing tables.

Our analysis predicts the delay stretch quite accurately. The average prediction error is 1.7% for GATech, 6.4% for Mercator, and 6.5% for CorpNet. The analysis tends to predict a larger delay stretch than the one measured because of our assumption that all nodes are equally likely to be used during routing. Our simulations show that the distribution of the number of routing table entries pointing to each node is skewed towards nodes that are central in the network, i.e., nodes that have a lower average delay to other nodes. This effect is more significant in Mercator and Corpnet and it results in lower per-hop delays.

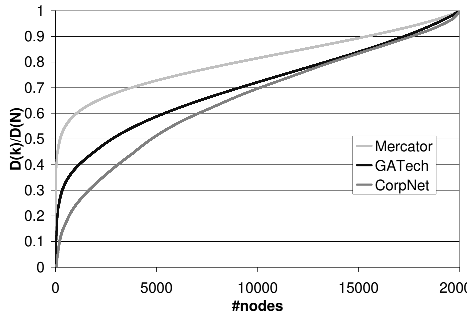

Figure 8: Average delay to the closest k nodes (D(k)) normalized by D(N) with $N = 20000$ for all topologies.

The delay stretch achieved by PNS depends on the topology: it tends to 1.58 in GATech, to 2.09 in Mercator, and to 1.26 in CorpNet. Recall that the analysis summarizes each topology using a function $D(k)$ that returns the average delay to the closest k nodes in the network. Figure 8 plots $D(k)/D(N)$

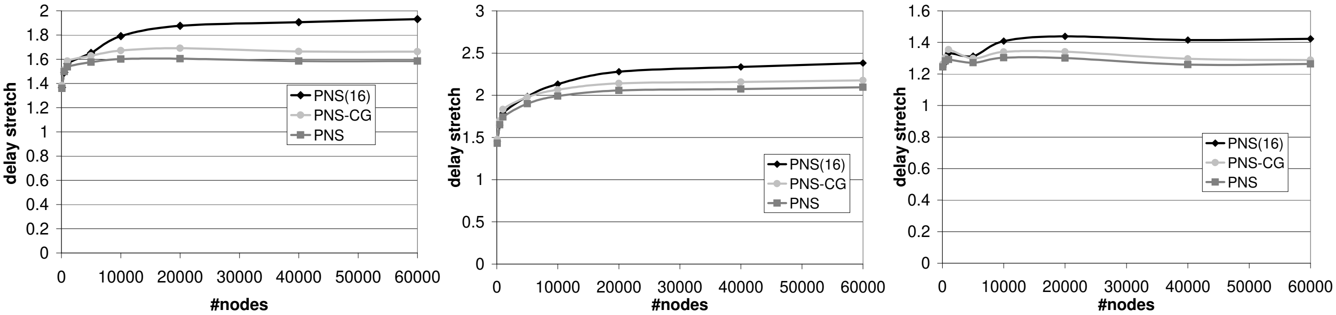

Figure 9: Delay stretch of heuristics with $b = 4$, $l = 16$, and varying N for GATech, Mercator and CorpNet.

for each topology with $N = 20000$ to provide some intuition on the difference between the topologies.

The expected delay in the last hop of a Pastry route is equal to the average delay between two points in the network $D(N)$. Pastry achieves low delay stretch when the delay in a route is dominated by the delay of the last hop. This is not the case in Mercator because $D(k)$ grows very fast reaching 60% of $D(N)$ for only 5% of the nodes. Therefore, the delay of the initial hops in Mercator is relatively large when compared with the delay of the last hop. We believe that this is due to the use of network hops as a proxy for delay in Mercator. Pastry achieves much better performance in CorpNet because $D(k)$ grows slowly. The initial hops in CorpNet account for less than 30% of the overall route delay.

Heuristics: Figure 9 shows that both heuristics achieve a delay stretch close to PNS. The delay stretch with PNS-CG is at most 5% worse than with PNS for all topologies and the delay stretch with PNS(16) varies between 13% worse in CorpNet to 22% worse in GATech. PNS(16) provides lower delay stretch than PNS-CG for small overlays but its performance degrades as the overlay size increases. The performance of PNS-CG is mostly independent of the overlay size. It is interesting to note that PNS-CG can achieve good performance even though the triangle inequality does not hold in either GATech or Mercator. In fact, PNS-CG is able to achieve a delay stretch below 1 for 5% of the messages in GATech and 0.3% in Mercator.

4.3 Delay stretch with varying b and l

We also compared the delay stretch with different values of b and l for $N = 20000$.

PNS: Figure 10 shows the delay stretch with $l = 16$ and varying b for GATech. Decreasing b increases the number of hops in a Pastry route and consequently increases the delay stretch. The delay stretch without locality increases by 100% with $b = 1$, but it increases only by 20% with PNS. The increased hop count is offset by decreased delay in the first hops of a Pastry route. The delays decrease because smaller values of b impose weaker constraints on the nodeIds that can fill slots at the top levels of routing tables. The analysis is still quite accurate with an average prediction error of 7%.

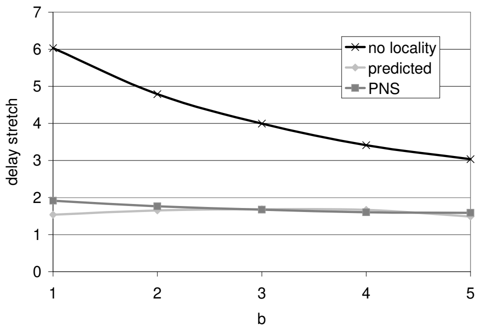

Figure 10: Delay stretch with $l = 16$, $N = 20000$, and varying b for GATech.

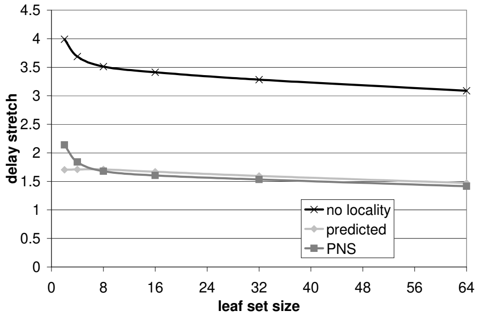

Figure 11: Delay stretch with $b = 4$, $N = 20000$, and varying l for GATech.

Figure 11 shows delay stretch with $b = 4$ and varying l for GATech. Increasing l decreases the number of hops and the delay stretch. The results show that using a leaf set with a reasonble size is important not only for fault tolerance but also to reduce delay. The delay of PNS with $l = 2$ is 51% worse than with $l = 64$. The analysis predicts the delay stretch with perfect routing tables accurately except for $l \leq 4$ (this is expected as explained in the Appendix).

Heuristics: Figure 12 evaluates the performance of the heuristics with varying b. The performance of both heuristics relative to PNS degrades in all topologies when b decreases. The heuristics are unable to offset the increase in hop count as well as PNS. The number of candidates for routing table slots increases when b decreases but PNS(16) always performs the same number of probes for each routing table slot. Similarly, the algorithm to locate nearby seed nodes in PNS-CG has a very small number of samples at each routing table level when b is small. This results in a significant performance degradation. Additionally, the amount of gossiping in PNS-CG is proportional to the routing table size, which increases with b.

We also evaluated the performance of the heuristics with varying l. Our results show that the performance of the two heuristics relative to PNS is mostly independent of the value of l across all topologies.

4.4 Local route convergence

The next experiment evaluates the local route convergence properties of PNS, PNS-CG and PNS(16). The experiment ran with $l = 16$, $b = 4$ and $N = 20,000$. Each of the 20,000 nodes sent a message to each of 10 randomly selected destination keys and we recorded the path from each node to the root node of each key. We used this information to compute a convergence metric for all pairs of paths to the same key. The convergence metric was $(\frac{c_r}{c_r+s_c^1} + \frac{c_r}{c_r+s_c^2})/2$, where c_r is the distance (through the overlay) from the node where the two paths converge to the root node, and s_c^1 and s_c^2 are the distances (through the overlay) from each source node to the node where the paths converge. This convergence metric captures the average fraction of the path that was shared between two messages sent to the same key. When the convergence metric is zero, the paths converge at the root node.

Figure 13 shows the average convergence metric value versus the distance between the two source nodes for the three network topologies. Perfect PNS achieves the best convergence. PNS-CG achieves almost as good convergence as PNS but has worse performance when the source nodes are very close in the underlying network. This is due primarily to the inaccuracy of the algorithm to locate a nearby seed node. The performance of PNS-CG using an oracle that returns the closest node in the overlay to seed joins is almost identical to perfect PNS. As observed in [15], PNS(16) provides poor convergence. It performs worse than PNS and PNS-CG across all topologies.

4.5 Joining overhead

We also ran experiments to evaluate the overhead of the heuristics when building the overlay. We quantified this overhead by measuring the average number of *distance probes* performed by each node when the overlay grows from empty up to a final size N. Each probe corresponds to the communication required to measure the distance between two nodes and involves at least two messages.

We compare PNS(16) with PNS-CG and with a variant of PNS-CG that obtains a nearby seed node from an oracle instead of running the algorithm in Section 3.4

The number of distance probes required by PNS(16) is $\sum_{i=1}^{128/b-1} min(1/2^{ib}, 16)$. With PNS-CG, a node that joins the overlay will eventually probe all the nodes in its routing table and in its leaf set. The routing table size can be approximated by $(2^b - 1)log_{2^b}N$ and the number of nodes in the leaf set is l. Additionally, a joining node sends each row of its routing table to all the nodes pointed to by the row and it sends its leaf set to all the nodes in its leaf set. This results in an additional $c_1(2^b - 1)^2 log_{2^b}N + c_2 l^2$ probes per join. The constants c_1 and c_2 are smaller than one because the current implementation remembers the distances to nodes that were probed in the past to avoid probing a node twice. The constants are difficult to estimate analytically. Putting these formulas together we can estimate the average number of distance probes per node for PNS-CG with the oracle $(1 + c_1(2^b - 1))(2^b - 1)log_{2^b}N + (1 + c_2 l)l$. The algorithm to locate a nearby seed node requires additional distance probes. The precise value is hard to estimate but it is $O(logn)$.

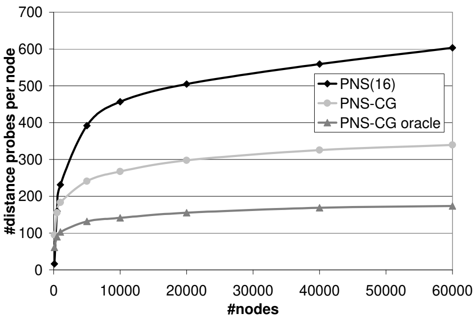

Figure 14: Number of distance probes per node in GATech with $b = 4$, $l = 16$, and varying N.

The joining protocol requires additional messages that are not distance probes. There are approximately $\frac{2^b-1}{2^b}log_{2^b}N$ messages to reach the root of the nodeId of the joining node. This cost is incurred by both PNS-CG and PNS(16). PNS(16) requires additional messages to locate the nodes to probe for each slot and PNS-CG requires messages to locate a nearby seed node and $(2^b - 1)log_{2^b}N$ messages to send the routing table rows to nodes in the new routing table of the joining node. The last set of messages can be piggybacked on the probing of the same nodes.

Figure 14 shows the number of distance probes per node in GATech with $b = 4$, $l = 16$, and varying N. The overheads were similar with other topologies. The overhead of all heuristics grows logarithmically with N as predicted by our analysis. The overhead of PNS(16) is initially lower than the overhead of PNS-CG but it grows faster with the overlay size. PNS-CG sends 78% less distance probes than PNS(16) when $N = 60,000$ and approximately half the probes are due to the algorithm to locate a nearby seed node at this point. This suggests that PNS-CG should attempt to locate a nearby node using IP multicast in the local network and use the more expensive algorithm only if this fails. The performance of this optimized algorithm should approach the performance of PNS-CG with oracle in many practical settings.

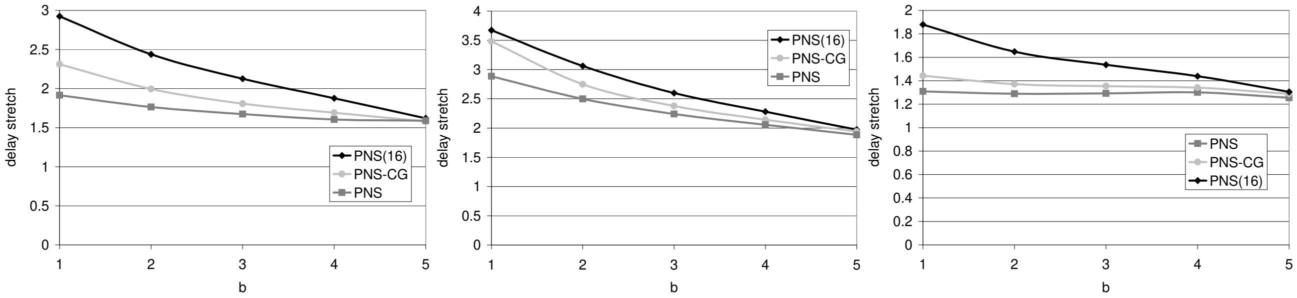

Figure 12: Delay stretch of heuristics with $l = 16$, $N = 20000$, and varying b for GATech, Mercator and CorpNet.

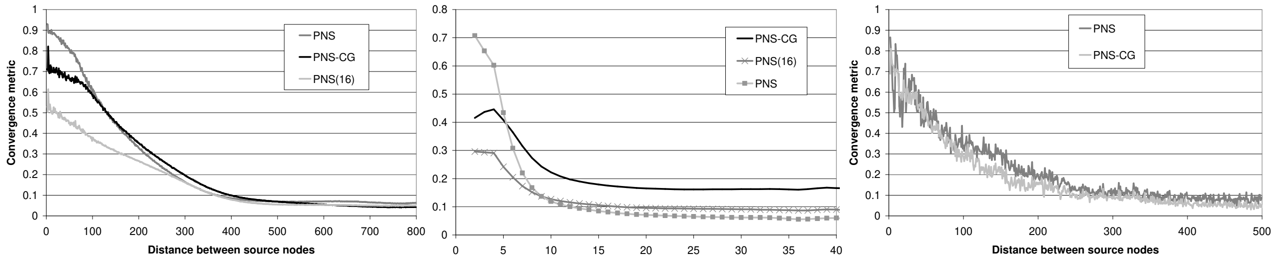

Figure 13: Convergence metric versus distance between the source nodes with $l = 16$, $N = 20000$, and $b = 4$ for GATech, Mercator and CorpNet.

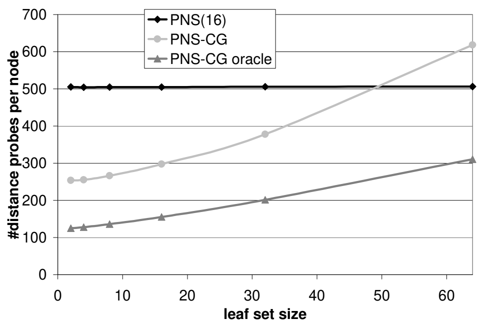

Figure 15: Number of distance probes per node in GATech with $b = 4$, $N = 20000$, and varying l.

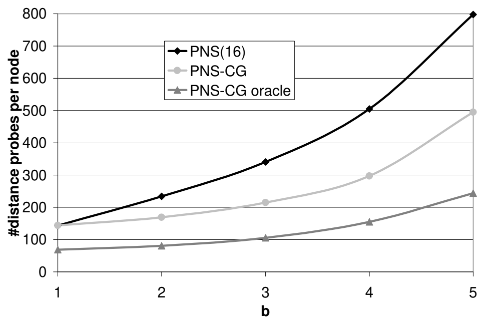

Figure 16: Number of distance probes per node in GATech with $l = 16$, $N = 20000$, and varying b.

We also ran experiments to evaluate the impact of varying the leaf set size on the overhead. Figure 15 shows the results. The overhead of PNS(16) is independent of the leaf set size because PNS(16) never probes the distance to leaf set members. The overhead of PNS-CG grows with l because leaf set members are probed both to locate a nearby seed node and to select entries to fill the lower layers of routing tables. Many of this probes are unlikely to generate good candidates and should be avoided but this is not a problem with the typical leaf set size between 8 and 16 (which achieves low overhead and can achieve very good fault tolerance with leaf set repair [16]).

Figure 16 shows the number of distance probes per node in GATech with $l = 16$, $N = 20000$, and varying b. The overhead increases exponentially with b as predicted by the analysis. PNS(16) and PNS-CG have similar overhead for small values of b but the overhead of PNS(16) increases faster with b. Choosing a value of 3 or 4 for b appears to be a good choice considering the number of routing hops, delay stretch, and overhead.

5 Conclusion

The paper presented a detailed study of proximity neighbor selection and two heuristic approximations in tree-based structured p2p overlays. The study was based on simulation results using three different network topologies, different overlay sizes, and different routing parameters. The results show that PNS provides a significant performance improvement relative to proximity unaware routing.

We introduced a new heuristic called constrained gossiping (PNS-CG). The study also compared the overhead of PNS-CG and PNS(16) [15] and their performance relative to PNS.

The results show that PNS-CG achieves performance similar to PNS with low overhead and that it achieves lower delay stretch and better route convergence than PNS(16) while incurring the same or lower overhead.

References

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

[2] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari 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.

[3] Antony Rowstron and Peter Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Middleware'01*, Nov. 2001.

[4] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph, "Tapestry: An infrastructure for fault-resilient wide-area location and routing," Tech. Rep. UCB//CSD-01-1141, U. C. Berkeley, April 2001.

[5] Petar Maymounkov and David Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS'02*, Boston, MA, Mar. 2002.

[6] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *PODC'02*, Aug. 2002.

[7] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, "Wide-area cooperative storage with CFS," in *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.

[8] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel, "Scribe: The design of a large-scale event notification infrastructure," in *Third International Workshop on Networked Group Communications*, Nov. 2001.

[9] Shelly Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination," in *NOSSDAV 2001*, June 2001.

[10] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao, "Distribted data location in a dynamic network," in *SPAA'02*, Aug. 2002, Winnipeg, Manitoba, Canada.

[11] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *Proc. 9th ACM Symp. on Parallel Algorithms and Architectures*, June 1997, pp. 311–320, Newport, Rhode Island, USA.

[12] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker, "Topologically-aware overlay construction and server selection," in *Proc. 21st IEEE INFOCOM*, New York, NY, June 2002.

[13] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron, "Exploiting network proximity in distributed hash tables," in *FuDiCo 2002: International Workshop on Future Directions in Distributed Computing*, June 2002.

[14] Miguel Castro, Mike Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlay networks," in *Proc. 22nt IEEE INFOCOM*, Mar. 2003.

[15] Krishna P. Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica, "The impact of dht routing geometry on resilience and proximity," in *ACM SIGCOMM 2003*, 2003.

[16] Ratul Mahajan, Miguel Castro, and Antony Rowstron, "Controlling the cost of reliability in peer-to-peer overlays," in *IPTPS'03*, Feb. 2003.

[17] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron, "Scalable application-level anycast for highly dynamic groups," in *NGC'2003*, 2003.

[18] David R. Karger and Matthias Ruhl, "Finding nearest neighbors in growth-restricted metrics," in *STOC'02*, July 2002.

[19] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *INFOCOM96*, 1996.

[20] 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.

[21] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron, "Exploiting network proximity in peer-to-peer overlay networks," Tech. Rep. MSR-TR-2002-82, Microsoft Research, 2002.

Appendix A

In this appendix, we derive a closed-form expression to compute the average delay stretch when messages are sent from random nodes to random keys. We start by developing a formula for the expected number and type of hops that will be required to predict delay stretch.

.1 Number of hops

Pastry routes a message towards a destination key by using the routing table to forward the message to a node that matches an additional digit of the key's prefix (line (8) in Figure 3). This is repeated until the key is between the nodeIds at the two extremes of the leaf set of the current node (line (3) in Figure 3). Routing completes in at most one hop after this happens.

To keep the analysis simple, we neglect the probability of being unable to forward the message to a node with a longer prefix match before reaching line (3). This case (line (11) in Figure 3) is important in the presence of faults and because our joining algorithm may temporarily leave routing tables incomplete. But with a reasonably large leaf set and perfect routing tables this probability is negligible. Our experimental results indicate that this case only becomes important for $l \leq 4$.

Let $P_{ls}(d)$ be the probability of using the leaf set of the current node to route given that the current node matches the first d digits of the key. We define

$$P_{rt}(d) = P_{ls}(d) \prod_{i=0}^{d-1} 1 - P_{ls}(i)$$

$P_{rt}(d)$ is the probability of using only routing table entries to reach a node that matches the first d digits of the key and then using this node's leaf set to reach the key's root. We also define $P_{me}(d)$ to be the probability that this node is the key's root.

We compute the expected overlap between sets in the id space to compute an approximate value for $P_{ls}(d)$ and $P_{me}(d)$. Let $I_{me} = \frac{2^{128}}{N}$ be the expected number of id space values that are assigned to a node, $I_{ls} = l \times I_{me}$ be the expected the number of id space values that are between the two extreme nodeIds in a leaf set, and $I(d) = \frac{2^{128}}{2^{bd}}$ be the number of id space values that match d digits of a key. If we approximate the distribution of interval sizes by the expected sizes defined above, we have:

$$P_{ls}(d) \approx P_o(I_{ls}, I(d))$$

$$P_{me}(d) \approx P_o(I_{me}, I(d))$$

where $P_o(A, B)$ is the probability that an interval with width A covers a point selected with uniform probability from an interval with width B, when the midpoint of the first interval is selected independently with uniform probability from the second. This probability can be approximated by

$$P_o(A, B) \approx \frac{1}{B^2} \int_0^B min(x, \frac{A}{2}) + min(B - x, \frac{A}{2}) dx$$

The integral is a good approximation because $I(d)$, I_{ls} and I_{me} are very large for the values of N and d that we use. We derived a closed-form expression for this integral.

We can use the above formulas to compute the expected number of hops, h, in a Pastry route as

$$h \approx \sum_{d=0}^{128/b} P_{rt}(d)[\frac{2^b - 1}{2^b} d + \frac{P_{ls}(d) - P_{me}(d)}{P_{ls}(d)}]$$

This formula simply multiples the probabilities $P_{rt}(d)$ for all possible vaues of d by their cost in hops. The cost of reaching a node that matches the first d digits of the key is smaller than d because whenever a node matches the first d digits of the key it will match $d+1$ with probability $\frac{2^b-1}{2^b}$. The second component of the cost accounts for the case where the destination is a leaf set member and not the current node, thus requiring an additional final hop.

The expected number of Pastry hops tends to $\frac{2^b-1}{2^b} log_{2^b} N$ when N grows because the effect of the leaf set becomes negligible. The expected number of nodes that match the first d digits of a key is $N/2^{bd}$. This number drops to one when $d = log_{2^b} N$. The number of hops is smaller than $log_{2^b} N$ for the same reason as before.

.2 Delay stretch

To analyze the delay stretch introduced by Pastry, we need to predict the expected delay of each Pastry hop. When a message is forwarded to a leaf set member of the current node, the delay is equal to the expected delay in the direct route between two random nodes in the underlying network. This is because nodeIds and keys are selected randomly with uniform probability from the id space and independently from a node's network location.

When a message is forwarded to a node in the routing table of the current node, the delay depends on the level of the routing table. We can predict the expected delay to nodes at each level of the routing table of a node p using the function $D(p, k)$, which returns the average delay from p to its k closest nodes in the underlying network, for $k \leq N$ (with ties broken arbitrarily). If $k > N$ then $D(p, k) = D(p, N)$. Additionally, let $D(p, k_1 : k_2) = \frac{k_2 D(p,k_2) - k_1 D(p,k_1)}{k_2 - k_1}$ denote the average distance from p to the set of nodes above the k_1-th closest up to the k_2-th closest for $k_2 > k_1$.

There is always one slot at each level d $(d \geq 1)$ of p's routing table that points to p. With perfect routing tables, the nodes in the other slots are selected by traversing the list of all nodes in order of increasing distance until nodes with the desired nodeId prefix are found. We expect to find a node that can fill another slot at level d after inspecting the first $N_1 = \frac{2^{bd}}{2^b - 1}$ nodes in the list. So the expected delay to this node is $D(p, 1 : N_1)$. Similarly, we expect to find a node to fill the next slot after inspecting an additional $\frac{2^{bd}}{2^b - 2}$ nodes in the list.

The expected delay to this node is $D(p, N_1 : N_2)$ where $N_2 = N_1 + \frac{2^{bd}}{2^b-2}$. Therefore, the expected delay to the i-th filled slot ($0 < i < 2^b$) at level d of p's routing table is approximately equal to

$$D(p, N_{i-1} : N_i)$$

with $N_i = N_{i-1} + \frac{2^{bd}}{2^b-i}$ for $i > 1$ and $N_0 = 1$. This is an approximation because we are approximating the distribution of the number of nodes required to fill a slot by its expected value.

The expected delay to the slots at level d of the routing table of node p, $D_{rt}(p, d)$, can be computed by averaging the expected distance to each slot given by the above formula:

$$D_{rt}(p, d) \approx \frac{1}{2^b - 1} \sum_{i=1}^{2^b-1} D(p, N_{i-1} : N_i)$$

The average of this function over all nodes in the overlay, $D_{rt}(d)$, can be used to compute the average delay in Pastry routes assuming that all nodes are equally likely to be used in routing. This average can be computed using a more compact characterization of the underlying network topology, which averages the function $D(p, k)$ over all overlay nodes

$$D(k) = \frac{1}{N} \sum_{p \in \mathcal{N}} D(p, k)$$

Given this compact description of the topology we can compute $D_{rt}(d)$ for all desired values of b:

$$D_{rt}(d) \approx \frac{1}{2^b - 1} \sum_{i=1}^{2^b-1} D(N_{i-1} : N_i)$$

This function can be used to compute the expected delay stretch, S, by replacing the unit cost for each hop by the expected delay of each hop in the equation to compute the expected numbed of hops h.

$$S \approx \frac{1}{D(N)} \sum_{d=0}^{128/b} P_{rt}(d)[\delta(d) + \frac{P_{ls}(d) - P_{me}(d)}{P_{ls}(d)} D(N)]$$

where $\delta(d) = \sum_{i=0}^{d} \frac{2^b-1}{2^b} D_{rt}(i)$, $D_{rt}(0) = 0$, and $D(N)$ is the average delay in the direct route between two random nodes in the underlying network. This analysis can be applied to arbitrary network topologies by providing a function $D(k)$ that characterizes the network.