Plethora: A Locality Enhancing Peer-to-Peer Network

Ronaldo A. Ferreira, Suresh Jagannathan and Ananth Grama
Department of Computer Sciences, Purdue University.
250 N. University Street
West Lafayette, IN, USA, 47907-1398
Emails: {rf, suresh, ayg}@cs.purdue.edu
Phone: +1 765 494-6010

Abstract. Distributed hash tables (DHTs), used in a number of structured peer-to-peer systems provide efficient mechanisms for resource location. A key distinguishing feature of current DHT systems like Chord [15], Pastry [12], and Tapestry [19] is the way they handle locality in the underlying network. Topology-based node identifier assignment, proximity routing, and proximity neighbor selection are examples of heuristics used to minimize message delays in the underlying network. While these heuristics are sometimes effective, they all rely on a single global overlay that may install the key of a popular object at a node far from most of the nodes accessing it. Furthermore, a response to a lookup message does not contain any locality information about the nodes holding a copy of the object. We address these issues by defining Plethora, a novel two-level overlay peer-to-peer network. A local overlay in Plethora acts as a locality-aware cache for the global overlay, grouping nodes close together in the underlying network. Local overlays are constructed by exploiting the structure of the Internet as Autonomous Systems. We present a detailed experimental study that demonstrates the practicality of the system, and shows performance gains in response time of upto 60% compared to a single global overlay. We also present efficient distributed algorithms for maintaining local overlays in the presence of node arrivals and departures.
Keywords: Peer-to-peer, DHT, locality, caching.

1 Introduction

The past few years have seen considerable activity in the area of peer-to-peer (P2P) systems and applications. The scale of P2P networks with large number of participating nodes requires these networks to be highly scalable in terms of aggregate resource requirement as well as end user performance. A number of researchers have addressed the problem of scalability in P2P networks [15, 12, 19]. Structured P2P systems such as Chord [15], Pastry [12], and Tapestry [19] provide a simple primitive for name resolution: given a file name, return the IP addresses of the nodes that currently have references to the file. To support this primitive, these systems rely on a *distributed hash table* (DHT) abstraction, and provide an upper bound of $O(\log n)$ on overlay hop-count, where n is the total number of nodes in the network. This upper bound is achieved using a small amount ($O(\log n)$) of routing information per node.

While recent work has focused on minimizing the number of overlay hops, the delay experienced by messages in the underlying network can also be a major performance bottleneck. Since nodes and objects are identified by random strings, lookup messages may travel around the world before reaching a destination node that is in the same LAN as the source node. To try to minimize the effects of randomization, several heuristics have been incorporated into these systems. These include *proximity routing*, *topology-based node ID assignment*, and *proximity neighbor selection* [11].

While these heuristics produce good results when compared to a standard implementation, they nevertheless rely exclusively on a single global overlay. Consequently, no guarantees can be provided that a popular object has its key installed close to the majority of nodes that access it. Moreover, a response to a lookup message does not contain any locality information about the nodes holding a copy of the object. Thus, a node receiving a query response has no information about which nodes are close or far from it. In this paper, we present the routing core of Plethora, a version-based wide-area read-write distributed file system currently under development at Purdue University [9].

The Plethora routing core is a two-level overlay: a global overlay, which shares the same properties as other structured P2P systems; and local overlays, which serve as locality-aware caches for the global overlay. The global overlay provides an information repository for operation and maintenance of local overlays, directing nodes to local overlays to which they belong. Plethora's two-level architecture uses additional information that is just a constant factor larger than the contemporary systems, but has the advantage of localizing queries for cached objects, reducing response time, and lookup message traffic in the underlying network. Moreover, responses to objects cached in the local overlay contain pointers to nodes that are close to the node issuing the query, thus reducing congestion on the underlying network.

The Plethora routing core's two-tier organization is motivated by studies which show that queries for multiple keys in P2P networks follow a Zipf-like distribution [14] – a few objects are very popular and are the targets of the majority of the queries. This suggests that a well designed cache mechanism can significantly improve the response time of the system. Another motivating factor is that for many wide-area distributed applications, nodes in the same region share common interests. For example, in a music sharing application, nodes in China will likely search for Chinese music.

To organize nodes into local overlays, we leverage the organization of the Internet as a collection of Autonomous Systems. An Autonomous System (AS) is a group of networks controlled by a single administrative authority, and in most cases restricted to a geographical region. An autonomous system is identified by a unique 16-bit number. When a new node joins the Plethora network, it first joins the global overlay and then uses its AS number to find other nodes of its AS that are already members of a local overlay. Each AS has a node in the global overlay, its *rendezvous point*, that is responsible for maintaining information about nodes in the AS that are present in the network.

In an ideal scenario, a local overlay contains nodes belonging to a single AS. Because the topology of the network may change frequently as peers join and leave, the number of participating associated with a particular AS participating in the overlay network at some point in time may be small; we therefore consider efficient distributed algorithms to merge local overlays. Two ASs are merged into the same local overlay if the distances among their nodes are smaller than a prescribed threshold. By allowing nodes in multiple ASs to form a local overlay, the arrival of new nodes to the network (which may occur with significant frequency) may increase the size of a local overlay beyond a point where performance improvements are not significant. To avoid this situation, we also present an efficient distributed algorithm for splitting local overlays. The algorithm for splitting a local overlay guarantees that nodes in the same AS stay in the same local overlay after a split operation.

Two parameters that impact the performance of our architecture are (i) the accuracy of topological information, and (ii) the optimality of sizes of the local overlays. There have been a number of other efforts aimed at accurately mapping the Internet topology [5, 8]. The results of these efforts can be used to derive topological information. Due to its reliance on the existing Internet infrastructure using ASs, our scheme is largely resilient to inaccuracies in topological information. With respect to determining optimal sizes for local overlays, the problem can be transformed into one of partitioning a weighted graph. A number of heuristics (multilevel and spectral methods) as well as distributed algorithms exist for this problem. We demonstrate here that a weak heuristic based on greedy node aggregation for merging ASs is sufficient for excellent performance.

1.1 Contributions

The main contributions of this paper are summarized below:

- We present a novel caching scheme that uses Internet topology information to group nodes that are physically proximate into local overlays. Our caching scheme can be used in conjunction with any DHT system, and does not rely on the deployment of well positioned landmarks.
- We present an efficient distributed algorithm for splitting local overlays that preserves locality in the two new local overlays created after a split operation. We present a rigorous analysis of the probabilistic properties of this algorithm.
- To ensure that local overlays do not become insignificantly small, we also develop an efficient distributed algorithm for merging local overlays.
- We present a detailed simulation study that demonstrates the effectiveness of our caching scheme. Our simulation study shows that our scheme can reduce response delays of the system by up to 60%, and can reduce the number of messages transmitted in the underlying network by 8-25% for different Zipf distributions compared to a corresponding Pastry network using the proximity neighbor selection heuristic. We also show that the overheads associated with splitting and merging local overlays is very low (approximately 5% and 15%, respectively).

2 Background

Current routing schemes in P2P networks such as Chord [15], Pastry [12], and Tapestry [19], work by correcting a certain number of bits at each routing step. In these systems, nodes and objects share the same address space. Nodes have their addresses assigned randomly and uniformly. This is generally achieved by computing a hash function on their IP addresses. Objects are identified by computing a hash function on their names. The uniform distribution of the identifiers (nodes and objects) is desirable in order to provide load balance in the system; that is, all nodes are expected to store roughly the same number of object keys. For the sake of brevity, we restrict our discussion here to Pastry and refer interested readers to the bibliography for other protocols.

In Pastry, objects and nodes are assigned random and uniformly distributed identifiers of length 128 bits. An object is stored in the node that is numerically closest to the object's identifier. Each node maintains routing information (overlay IDs and IP addresses) about a limited number of neighbors, with the size of the routing table varying depending on a configuration parameter (b) that indicates how many bits are resolved at each routing step. To route a message, each intermediate node, along the message path, tries to forward the message to a node whose node ID shares a prefix with the destination identifier that is at least b bits longer than the current node's shared prefix. If the routing table does not have any entries for such a node, the message is forwarded to a node that shares a prefix of the same length as the prefix shared by the current node, but is numerically closer to the destination. Each node also maintains a *leaf set*. The leaf set of a node n stores information about l nodes that are numerically closest to n, with $l/2$ nodes having smaller IDs than the current node, and $l/2$ having larger IDs. This information is stored to provide reliable message routing, and is normally used when the routing table does not have an entry for a node that shares a prefix longer than the current fnode. We refer the reader to [12] for a more detailed explanation of Pastry and its algorithms.

3 Plethora Routing Core

The Plethora routing core relies on a two-level overlay architecture [4, 18, 10, 16]. A global overlay serves as the main data repository, and several local overlays serve as caches to improve access time to data items. When a node n needs a data item, it first searches its local overlay. If the data item is cached in its local overlay, a copy is immediately returned. Otherwise, the data item is retrieved from the global overlay and automatically cached in n's local overlay.

Local overlays contain nodes that are close to each other in the Internet. To group nodes into local overlays, we rely on the organization of the Internet as a collection of autonomous systems (ASs). More specifically, nodes that are in the same AS should be in the same local overlay together with nodes of neighboring autonomous systems. The global overlay can be implemented using any DHT system (e.g., Pastry, Tapestry, Chord, etc.). Since we describe Pastry in section 2 and we adopt some of its underlying algorithms to support the local overlay, we use it in the global overlay as well. We introduce one small modification: in addition to the IP addresses of neighbor nodes, we also store their AS numbers as part of the state information stored at each node. A node can determine its AS number from its IP address using a number of possible alternatives. Whois servers are the simplest one, but since the data stored in the servers are not kept up-to-date, more accurate tools can be used. In [7] Mao et.al presented an accurate AS-level traceroute tool for mapping IP addresses to AS numbers. It is important to note that even if a whois server is used, the server does not constitute a bottleneck in our scheme since its content can be conveniently replicated. The server is accessed when a new node first joins the overlay network, and the new node's AS number can be locally cached for future reference.

A node must build its Internet neighborhood information (i.e., the list of ASs that can be present in its local overlay) over time. An initial list can be built from the routing information that the node receives when joining the global overlay. The node can measure the delay to each member of its routing table and leaf set. If the delay is less than a system parameter D, it includes the AS of the probed node in its neighborhood list. This is a nonintrusive way of acquiring neighborhood information. A more aggressive way for a node n to build the same list is by using `traceroute` to some of the nodes in n's state tables, using the IP addresses of the intermediate hops to find nearby ASs. Observe that we assume that the delay from a node in one AS to any other node in some other AS is an acceptable indication of proximity. This assumption may not be always true when we deal with ASs that span large geographical areas – we address this issue in section 3.6. To simplify the presentation of our algorithms, we initially assume that this assertion is valid.

In each local overlay there is a node, the *local overlay leader*, that controls the number of nodes in the local overlay. To avoid a complicated protocol for leader election, we assume the local overlay leader is the node with the smallest identifier in the local overlay. The local overlay leader must issue `split` messages if the number of nodes grows above the maximum allowed, and `merge` messages if the number of nodes drops below the minimum allowed. The maximum and minimum number of nodes in a local overlay are system parameters. To avoid having the leader maintain accurate information about the peers in its local overlay, thereby having to support a complicated protocol for node departures, we assume that the local overlay leader periodically circulates a message in the network to discover the current number of nodes. This is a reasonable approach, since the local overlay size has an impact only on performance and not in the correctness of the protocol. A node in a local overlay maintains a pointer to its current leader. This pointer is used to determine if the leader has departed or failed – in which case a new leader is selected.

Even though the maximum number of nodes in a local overlay is governed by a system parameter, we expect that this value can be arbitrarily large. We therefore require efficient algorithms to route messages within a local overlay. A simplistic scheme, such as one that stores complete information at each node, would not scale to a significant number of nodes. We use a modified version of Pastry to achieve scalability and to provide guarantees on the number of hops a message travels in the local overlay.

In the local overlay, nodes are also identified by random strings. One key difference in the identification of local overlay nodes is that the number of bits in a node's identifier can change over time. The reason for this will become clear when we present algorithms for merging and splitting local overlays. When a new node decides to join an existing local overlay, it must be informed of the current identifier length by the contact node. It then generates a random identifier of the informed length locally before asking the contact node to introduce it into the local overlay. A new node is introduced into a local overlay in the same way as in Pasty: a join message is sent by the contact node and routed to the new node identifier. Nodes along the path send their state information to the new node.

Since object identifiers are assigned random strings of fixed length in the global overlay, 128 bits in the case of Pastry, and we want the objects to have the same identifiers in both overlays, we only use the w least significant bits of the object identifiers to decide which node will store a particular object's key; w being the number of bits currently used to identify nodes in the local overlay. The node with the identifier numerically closest to the restricted object identifier is the one responsible for storing the object's key.

3.1 Routing in the Local Overlay

Routing in the local overlay proceeds as in the global overlay, with one significant exception. Instead of resolving multiple bits at each hop during the routing process, we resolve only one bit per hop. This is equivalent to having $b = 1$ in Pastry. The routing table of a local overlay node, therefore, has only one column. Row i of n's routing table has a pointer to a node whose identifier shares i bits with n's identifier and differs at bit $i+1$, with i varying from 0 to $w-1$, where w is the current identifier length. The reason we have $b = 1$ in the local overlay is to simplify the operations of merging and splitting as discussed in Sections 3.4 and 3.5. This is not a fundamental constraint – by suitably modifying merge and split operations to multi-way split and merge operations, higher values of b can be used. We also change the size of the leaf set to $2 \times k \log M$, where k is a constant greater than one, and M is the maximum number of nodes in a local overlay. This last choice is discussed in Section 3.4.

3.2 Node Arrivals

When a new node arrives, it needs to initialize its tables and also inform other nodes of its arrival. A new node first joins the global overlay, and initializes all its global overlay tables the same way as in Pastry. The AS number of the nodes is the only additional information introduced. After joining the global overlay, the new node must join a local overlay. To find which local overlay it should join, it computes a hash function on its AS number, and uses the resulting value to find a node in the global overlay that is responsible for keeping information about that particular AS (the *rendezvous point* of the AS).

Two situations may arise depending on the response of the rendezvous point. In the simplest case, the autonomous system of the new node already has some nodes in the network. In this case, the rendezvous point returns a list with the current nodes. The new node chooses one of the nodes as its contact in the local overlay. The contact node introduces the new node in the local overlay and sends to it a pointer to the local overlay leader. The rendezvous point also stores the new node information in its AS directory. The AS directory does not need to store all AS nodes present in the network; rather, it stores a constant number of nodes that will be used as contacts. The nodes in the local overlay refresh this information periodically. We assume that the global overlay replicates the contents of its nodes appropriately. If a rendezvous point leaves, one of its neighbors becomes the new rendezvous point and has the required information of the AS.

In the second case, the new node is the first node of its AS to join the network. In this case, the new node stores information at its AS's rendezvous point and uses its neighborhood information to find a contact node in a local overlay of some other AS. If the new node can find a contact node, it executes the procedure described above to join the contact node's local overlay. If the new node cannot find any contact nodes because its neighborhood list is empty or none of the ASs in the list have nodes currently in the network, the new node starts a new local overlay. In a well populated network, the latter case is unlikely, since the neighborhood list can be built with nodes already in the global overlay.

3.3 Node Departures

Nodes in the overlay networks may leave or fail without notifying their neighbors. Global overlay recovery is handled by the DHT system used. Node departures in Pastry are handled lazily, that is, a node departure is detected only when a node tries to contact the node that left to route a message or to access a data item. When a node detects that one of its neighbors has left, it must contact other nodes to restore its state information. The leaf set of a node n can be restored by contacting a live node y in n's leaf set. Node y sends its leaf set to n, and n fixes its information using the appropriate node. If n detects that a node x in its routing table is not responding, n can contact a live node y in n's routing table that is in the same row of x, and ask y to send n its routing table. Like in Pastry, node departures in Plethora are handled lazily.

The departure of a node also triggers updates to the state stored in the nodes other than the routing tables. When a node n detects that a node y has left the local overlay, n must notify y's rendezvous point in the global overlay of its departure. One other update that may happen is when a node n detects that the local overlay leader has left; a new local overlay leader must be chosen in this case. Once the leader's departure is detected, the node with the smallest identifier sends a broadcast message informing the other nodes that it is the new leader.

3.4 Splitting Local Overlays

When the local overlay leader detects that the number of nodes currently present in the local overlay exceeds the maximum allowed, it issues a split message to the local overlay members. The split message is a simple broadcast message that does not carry any information about the nodes in the local overlay, or how the nodes will be partitioned. A node receiving a split message must decide locally how the operation will be performed. One problem that may occur during a split operation is that nodes of the same AS may end up in different local overlays. This is a situation that we would like to avoid, since local overlays may eventually degenerate and lose their primary purpose to localize queries. Therefore, a split operation in a local overlay must preserve the invariant that *nodes of an autonomous system must always stay in the same local overlay after a split operation.* The implementation of this invariant is a challenging problem in itself. We want a distributed solution where the nodes should exchange as few messages as possible to restore their state tables. To implement the split operation efficiently, we extend the information stored in the routing tables of the nodes. For each entry in a node's routing table, we add a secondary neighbor. The secondary neighbors are chosen to preserve this invariant. To implement the invariant, we use a hash function $\mathcal{H}$ that maps an AS number to the set $\{0, 1\}$. This set identifies the two overlays that are created after a split operation. A node n applies $\mathcal{H}$ to its AS number and determines the local overlay it will belong to after a split operation. The secondary neighbors in n's routing table are nodes whose hash function $\mathcal{H}$ maps to the same value as node n's. Node n can determine locally if a given node qualifies as a secondary neighbor, since nodes exchange their AS numbers along with their IP addresses.

On receiving a split message, node n discards all pointers to nodes whose hash values differ from its hash value. This operation is performed in the routing table as well as in the leaf set. Figure 1 illustrates this operation. Observe that for correct operation of the network, it is sufficient that the leaf set of a node n contains at least one node in each direction whose hash function maps to the same local overlay as n.

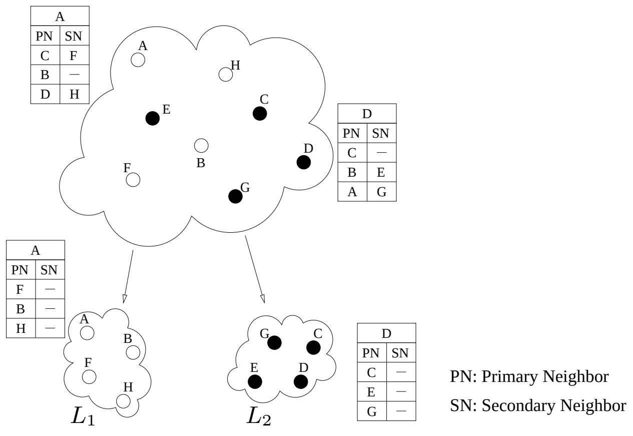


Fig. 1. Split operation. The two smaller clouds (L_1 and L_2) represent the two local overlays after the split operation. The circles of the same color represent nodes whose autonomous systems hash to the same value. The tables are the routing tables of nodes A and D before and after the split operation.

After a split operation, two overlays are created. One of the new local overlays will contain the leader of the original overlay; this leader remains the leader in the new overlay. In the other overlay, a new leader must be chosen. We use the same approach described above – the node with the smallest identifier is chosen as the new leader. This node must send a message to the nodes in its overlay informing them that it is the new leader, and collecting information about which ASs are present in the new local overlay.

A split operation may result in disconnected networks depending on the number of nodes in the leaf set. The reason why we set the leaf set size equal to $2 \times k \log M$ is to guarantee that the new local overlays will be connected with high probability (whp)[1]. The following lemma provides the connectivity guarantee.

Lemma 1 After a split operation, the two new local overlays are connected with high probability.

Proof: Let n be a node in the original overlay and h be the hash value, computed with the function $\mathcal{H}$, of its AS number. Without loss of generality, assume that h is equal to 0. The probability of n being in a disconnected overlay after a split operation is the probability that all nodes on one side of its leaf set (nodes in the clockwise direction, for example) have h equal to one. Assuming that the values 0 and 1 are equally possible, this probability is equal to:

$$\left(\frac{1}{2}\right)^{k \log m} = \frac{1}{m^k}$$

where m is the number of nodes in the original local overlay. The probability of n being in a connected overlay is, therefore, $1 - \frac{1}{m^k}$. □

The same analysis can be used to show that, before the split operation, the routing table of a node n has secondary pointers to nodes that share the same hash value of n, w.h.p. Consider the i^{th} row of n's routing table, $0 \leq i < w$, w is the number of bits in n's identifier. Let m be the number of nodes in the local overlay, and z be the number of nodes that can be chosen for row i in n's routing table. Since we assume that the node identifiers are uniformly distributed, the expected value of z is equal to $\frac{m}{2^{(i+1)}}$. Since we also assume that the values 0 and 1 are equally possible as results of the function $\mathcal{H}$, the probability that row i has a secondary neighbor is equal to $1 - \left(\frac{1}{2}\right)^z$.

3.5 Merging Local Overlays

If the number of nodes in a local overlay L_1 drops below the minimum allowed, L_1 must be merged with a nearby local overlay L_2. If the number of nodes in L_1, m_1, is much smaller than the number of nodes in L_2, m_2, simple insertions of the nodes in L_1 are performed in L_2. However, if m_1 differs from m_2 by at most a factor α, where α is a system parameter, we use an algorithm based on the mechanism for merging two hypercubes [6]. L_1 and L_2 can be viewed as two hypercubes of dimension d that can be merged to form a hypercube of dimension $d + 1$. First, we select bit values for the overlays; for example, 0 for L_1 and 1 for L_2. The bit values chosen are sent to the nodes as part of the merge message. The nodes increase the length of their identifiers by adding their overlay bit to their identifiers as the most significant bit. A new top row is also added to the routing tables of the nodes. A node in the local overlay L_1 must fill the new row with a pointer to a node in the local overlay L_2. The same operation must be applied by the nodes in the local overlay L_2 with

[1] Probability equal to $1 - \frac{1}{n^{\Omega(1)}}$.

new pointers pointing to nodes in L_1. Since the number of nodes in L_1 and L_2 is bounded, the merge message sent in L_1 can contain the nodes in L_2, and the merge message sent in L_2 can contain the nodes in L_1. Each node chooses the new neighbor independently. Figure 2 illustrates the merge operation. Observe that an efficient merge operation is necessary, even if the number of nodes in a local overlay is small. If a complete reorganization of the nodes were necessary in a merge operation, nodes would have to close all their current connections and open new ones.

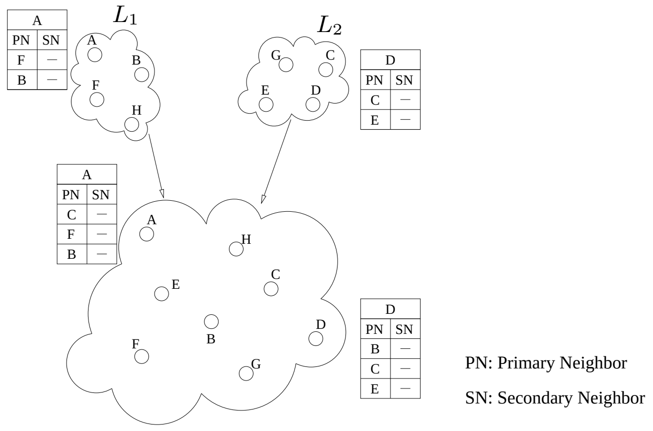


Fig. 2. Merge operation. L_1 and L_2 are the two local overlays that will be merged. The bigger cloud represents the local overlay after the merge operation. Observe that one row was added to A's routing table with a pointer to a node originally in L_2. The same thing happened in D's routing table, but here a pointer to a node originally in L_1 was added.

Once the new local overlay is established, some keys stored at a node n may have to move to a different node. Since we just add one bit to n's identifier, n just needs to republish the keys to objects whose identifiers differ in the new bit added.

3.6 Dealing with Large Autonomous Systems

As discussed in related work [18, 16], an autonomous system constitutes a reasonable unit of locality. Most of the current ASs in the Internet span a limited geographical area. However, there are also ASs that span a very large geographical area, such as big Internet service providers. To handle these systems, we can adapt our scheme in the following manner. When the first node n of an AS joins the network, it installs its information and declares itself a local landmark. When a node x from the same AS joins the network, it measures its delay to n, and to other landmarks of the AS. If there is a landmark y whose delay to x is less than D, the maximum distance allowed, x joins the local overlay of y. If there is no such y, x declares itself a new landmark, generates a virtual AS number (bigger than the maximum AS number allowed in the Internet), and either creates its own local overlay or joins the local overlay of an AS that is close to it. The virtual AS number is used, in conjunction with the IP address and the real AS number, to identify x in the local overlay, and to decide in which local overlay x will end up when a split operation is performed.

4 Experimental Results

In this section, we present simulation results for various performance aspects of the Plethora routing core. Specifically, we demonstrate considerable performance improvements (up to 60%) in response time, and show low overhead associated with control algorithms for joining, splitting, and merging local overlays, and storage. These results are in comparison to a Pastry network using the proximity neighbor selection heuristic.

Our simulation testbed implements the routing schemes of the global and local overlays, implements the algorithms for merging and splitting local overlays, and emulates the underlying network. The topology of the underlying network is generated using the Georgia Tech. transit-stub network topology model (GT-ITM) [17]. The underlying network topology used in our experiment contains 10 transit domains, with an average of 10 nodes per transit domain. Each transit node has an average of 10 stub domains. There are a total of 1,000 stub domains, with an average of 10 nodes per stub domain. A LAN with 10 hosts is connected to each stub node, resulting in a total of 100,000 hosts. The total number of nodes in the underlying network topology is, therefore, 110,100. Each stub domain is considered an AS. Link delays are random values within the following intervals: the delay of an edge between two transit domains is in the interval [20-80]ms; the delay of a transit-transit edge in the same transit domain is in the interval [3-23]ms; the delay of a transit-stub edge is in the interval [2-7]ms; the delay of a stub-stub edge is in the interval [1-4]ms; and the delay of an edge connecting a host to a stub node is fixed at 1ms. These figures are similar to those used by Xu et al. [16].

The evaluation of our routing scheme requires appropriate dimensioning of the storage available for caching at each node, and a realistic workload. Since there are no publicly available traces that contain file sizes for existing peer-to-peer systems, we use web proxy logs for the distribution of file sizes in the network. The same approach has been used to validate PAST [13]. We use a set of web proxy logs from NLANR[2] corresponding to eight consecutive days in February 2003. The trace contains references to 500,258 unique URLs, with mean file size 5,144 bytes, median file size 1,663 bytes, largest file size 15,002,466 bytes, and smallest file size 17 bytes. The total storage requirement of the files in the trace is 2.4GBytes.

As described in the previous section, a query that cannot be satisfied by the local overlay causes an access to the global overlay and the automatic caching of the object in the local overlay of the node issuing the query. As in PAST, we assume that objects are also cached in addition to keys. Due to the storage requirements of our workload, and to guarantee that the simulation would reach a steady state, the number of nodes is set to 10,000, with each node

[2] http://www.ircache.nlanr.net/

having a cache of size 5MB (approximately 0.2% of the trace requirement). While this cache size is small considering the current storage capacity of hard disks, it is dimensioned based on the worload used, composed mainly of small files. The overlay nodes are selected randomly from the 100,000 hosts. If the number of overlay nodes is increased without reducing the cache size per node to insignificant values, the total cache capacity available in the network would exceed the total storage requirement. This would cause the cache hit ratio to keep increasing with the number of queries. With the selected parameters, we are able to reach steady state after 90 million queries.

The main performance measurements that we investigate are the performance gains in response delay and number of packets in the underlying network for queries in the two-level overlay compared with a single Pastry overlay. Performance gain is defined as: $g = \frac{m_1 - m_2}{m_1}$, where m_1 is the measurement (average delay or lookup messages) in Pastry, and m_2 is the measurement in the Plethora routing core. The source nodes of the queries are chosen randomly and uniformly, and the objects are accessed according to a Zipf-like distribution, with the ranks of the objects being determined by their position in the original NLANR trace. The parameter α of the Zipf distribution is set to 0.70, 0.75, 0.80, 0.85, and 0.90. For the global overlay the Pastry parameters are: $b = 4$, and leaf set size $l = 32$. These parameters are also used in the single Pastry overlay. We measure the impact on performance gains of the cache hit ratio, the maximum delay parameter D used to construct local overlays, and the maximum number of nodes in a local overlay. The cache hit ratio is the fraction of all queries that are satisfied by a local overlay. We simulated both LRU (Least Recently Used) and GDS (Greedy Dual Size [1]) as the local cache replacement policies. The results for both policies were similar, we report the results for LRU.

The cache hit ratio is a function of the α value in the Zipf distribution and the maximum number of nodes in a local overlay. Figure 3 illustrates the cache hit ratios obtained with $\alpha = 0.70$ and $\alpha = 0.90$ and different maximum local overlay sizes. The values in the tables correspond to the minimum ($\alpha = 0.70$) and maximum ($\alpha = 0.90$) ratios obtained for the parameters, other values of α produce cache hit ratios in these intervals.

$\alpha = 0.70$					
Size \ Delay	30ms	40ms	50ms	100ms	200ms
200	51.70%	51.38%	51.24%	52.98%	56.64%
300	57.04%	58.40%	60.50%	61.36%	64.33%
400	59.31%	66.03%	66.17%	69.29%	68.88%
500	60.75%	68.26%	68.28%	72.65%	78.17%
1000	60.75%	72.15%	75.64%	86.03%	89.22%
2000	60.75%	72.15%	77.72%	90.01%	94.90%

$\alpha = 0.90$					
Size \ Delay	30ms	40ms	50ms	100ms	200ms
200	71.69%	71.55%	71.50%	72.53%	74.59%
300	74.72%	75.59%	76.82%	77.33%	78.89%
400	75.92%	79.65%	79.83%	81.59%	81.34%
500	76.68%	80.81%	80.94%	83.34%	86.20%
1000	76.68%	82.84%	84.78%	90.38%	92.18%
2000	76.68%	82.84%	86.00%	92.79%	95.88%

Fig. 3. Cache hit ratios for $\alpha = 0.70$ and $\alpha = 0.90$.

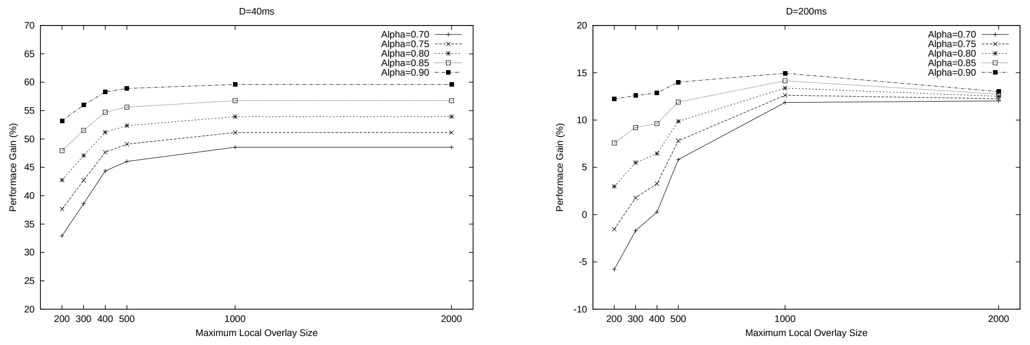


Fig. 4. Performance gains in response delay for $D = 40$ms, $D = 200$ms.

Figure 4 shows the performance improvements for the response delays. We use D equal to 30ms, 40ms, 50ms, 100ms, and 200ms in the simulation. The values in the figure correspond to D equal to 40ms, and 200ms. For D equal to 40ms we observe the best results – roughly 60% improvement. Figure 4 also shows an important result – for a large value of D ($D = 200$ms) the results are not only worse than the smaller values of D, but as the maximum number of nodes increase the performance also deteriorates. The diameter of the underlying network (maximum distance between any two nodes) for the topology generated is 298ms; this means that for D equal to 200ms the local overlays can include nodes spread almost over the entire network. This is clearly not a desirable situation, since the local overlays just add additional overhead and do not serve their intended purpose of localizing traffic. The other values of D produce results in the intervals covered by D equal to 40ms and 200ms.

We also quantify the number of packets sent in the underlying network when we have a single overlay and compare it to the two-level overlay. Due to space limitations, we show only the results for D equal to 30ms and 40ms in Figure 5. The performance improvements in all configurations are in the range 12.25% to 24.43%. The characteristics of the underlying topology is a limiting factor on the improvement that we observe, compared to improvements in response delays. With only 10 transit domains, the number of edges between two nodes in different transit domains is not significantly different from the number of edges between nodes in stub domains connected to the same transit domain.

The two-level overlay architecture of the Plethora routing core is useful only if the additional overhead of maintaining local overlays is sufficiently small. In Figures 6 and 7 we show the results of experiments that quantify this overhead. In the first experiment, we quantify the overhead of the split operation. In this case, all 10,000 nodes join the global overlay and the local overlays simultaneously. As new nodes are added to the local overlays, split operations are required. We measure the total number of messages exchanged by nodes in the global overlay and messages exchanged by nodes in the local overlays. The overhead associated with the split operation is very small – always less than 5% for all parameters used. To quantify the merge operation, we randomly select nodes to leave the network. As the number of nodes drops below the minimum threshold (in these experiments the minimum

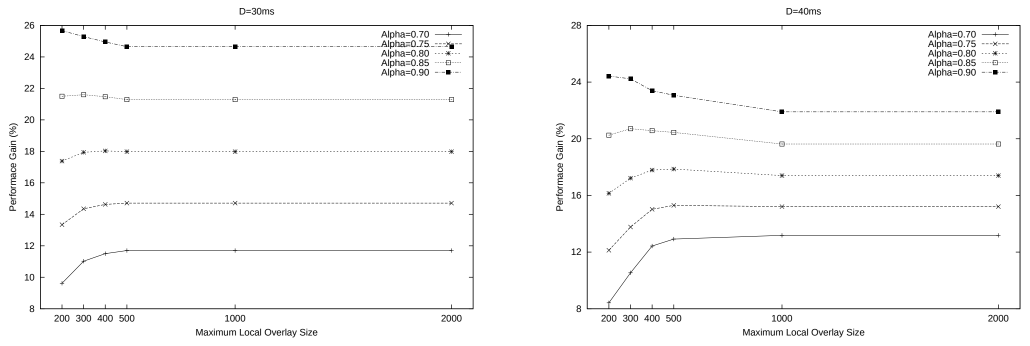


Fig. 5. Performance gain in the number of packets in the underlying network for D equal to 30ms and 40ms.

value is set to 20%, 30%, 40%, 50%, and 60% of the maximum value), merge operations are performed. Figure 7 shows the result for 20% and 60% of the maximum size. As we can observe from the figure, this operation is more expensive than the split operation. However, in all cases the additional overheads are smaller than 25%, with the highest value happening for a minimum size close (60%) to the maximum. This means that merge operations are executed very frequently when the nodes are leaving. Note that these fractional increases are with respect to the number of control messages in the corresponding Pastry network. In stable operation, the number of control messages can be expected to be dominated by data messages. Consequently, we expect the added overhead of our two-level overlay to be negligible.

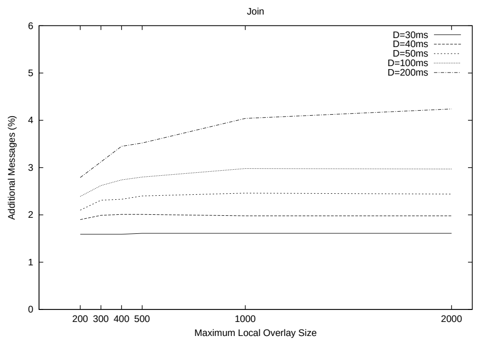
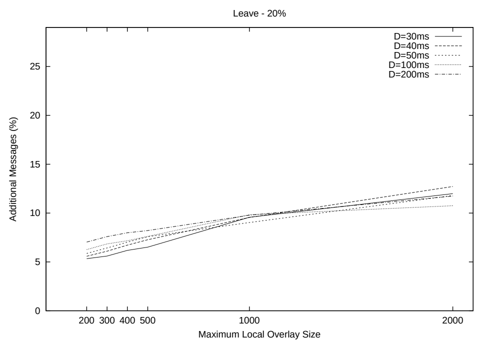


Fig. 6. Additional messages sent in the two-level overlay when nodes join the network, including messages exchanged in the split operation, when compared with a single overlay.



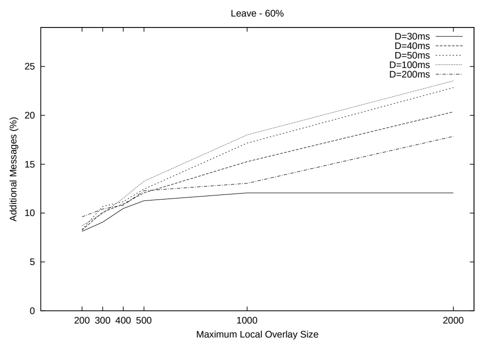


Fig. 7. Additional messages sent in the two-level overlay when nodes leave the network, including messages exchanged in the merge operation, when compared with a single overlay.

5 Related Work

Three major approaches have been proposed for topology-aware overlay construction in DHT networks [11]: *proximity routing* [3], *topology-based node ID assignment* [10], and *proximity neighbor selection* [2]. These heuristics improve the performance of the basic DHT systems, but they also introduce additional problems. Load imbalances, increase in the number of overlay hops, destruction of the uniform distribution of nodes in the ID space are among the problems these heuristics can cause in a DHT system.

The use of a two-level architecture to improve overlay performance is not new. Brocade [18] uses a secondary overlay network of supernodes. The supernodes are nodes with good capacity and network connectivity and are assumed to be close to network access points such as routers. Nodes inside an AS use the supernodes to access objects in the global overlay. Our approach differs from Brocade in several important respects. In Brocade, a normal node (not a supernode) participates in the overlay via supernodes. A normal node first needs to contact a supernode and asks it to route its messages. Supernodes have to keep information about all overlay nodes inside their ASs. The Brocade organization is basically an overlay of servers with several clients connected to them. There is no deterministic distributed way for a normal node to find a supernode. It is assumed that the supernodes are able to snoop the underlying network and detect overlay traffic, or that the supernodes have well known DNS names.

Xu et al. [16] also proposes a two level overlay, consisting of one auxiliary overlay, called *expressway*, composed of supernodes, as in Brocade, and a global overlay. The global overlay is used to find nodes in the same AS or nodes that are physically close, as in our scheme. Nodes in the expressway exchange routing information, much in the same way as routers exchange BGP reports in the Internet. Our scheme resembles this approach in the way a node finds information about other nodes that are physically close to it by using the global overlay as a rendezvous point. A key difference is that the auxiliary network in [16] is intended to speed up the communication of nodes far apart in the Internet. Our auxiliary network (local overlay) is intended to speed up the communication among nodes that are close to each other, taking advantage of possible common interests, and reducing traffic in the global network. Furthermore, the expressway requires significant work to be performed by the supernodes. The supernodes try to emulate, in the overlay network, the BGP routing protocol of the Internet.

6 Concluding Remarks

In this paper, we present Plethora, a two-level overlay architecture and show significant performance improvements with respect to state-of-the-art structured peer-to-peer systems. We also demonstrate low overheads associated with various control operations and resource requirements for Plethora.

References

1. P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of USENIX Symposium on Internet Technologies and Sysstems (USITS)*, Monterey, CA, December 1997.
2. M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting Network Proximity in Peer-to-Peer Overalay Networks, Technical Report MSR-TR-2002-82, 2002.
3. F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-Area Cooperative Storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Lake Louise, Canada, October 2001.
4. R. A. Ferreira, A. Grama, and S. Jagannathan. An IP Address Based Caching Scheme for Peer-to-Peer Networks. In *Proceedings of IEEE Globecom 2003, To appear*, San Francisco, CA, December 2003.
5. P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gryniewicz, and Y. Jin. An Architecture for a Global Internet Host Distance Estimation Service. In *Proceedings of IEEE INFOCOM 1999*, New York, NY, March 1999.
6. Frank Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1992.
7. Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an Accurate AS-Level Traceroute Tool. In *Proceedings of the 2003 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Karlsruhe, Germany, August 2003.
8. T. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordiantes-Based Approaches. In *Proceedings of IEEE INFOCOM 2002*, New York, NY, June 2002.
9. Plethora. http://www.cs.purdue.edu/homes/rf/plethora/.
10. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proceedings of IEEE INFOCOM 2002*, New York, NY, June 2002.
11. S. Ratnasamy, S. Shenker, and I. Stoica. Routing Algorithms for DHTs: Some Open Questions. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Boston, MA, March 2002.
12. A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 247–254, San Diego, CA, August 2001.

13. A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Lake Louise, Canada, October 2001.

14. K. Sripanidkulchia. The Popularity of Gnutella Queries and its Implication on Scalability, February 2001. http://www.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html.

15. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160, San Diego, CA, August 2001.

16. Z. Xu, M. Mahalingam, and M. Karlsson. Turning Heterogeneity into an Advantage in Overlay Routing. In *Proceedings of the IEEE INFOCOM 2003*, San Francisco, CA, April 2003.

17. E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE INFOCOM 1996*, March 1996.

18. B. Zhao, Y. Duan, L. Huang, A. Joseph, and J. Kubiatowicz. Brocade: Landmark Routing on Overlay Networks. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.

19. B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-0101141, UC Berkeley, Computer Science Division, April 2001.