# Exploiting Host Name Locality for Reduced Stretch P2P Routing

Gert Pfeifer, Christof Fetzer and Thomas Hohnstein
Dresden University of Technology
gert.pfeifer, christof.fetzer, thomas.hohnstein@inf.tu-dresden.de

## Abstract

*Structured P2P networks are a promising alternative for engineering new distributed services and for replacing existing distributed services like DNS. Providing competitive performance with traditional distributed services is however very difficult because existing services like DNS are highly tuned using a combination of caching and localized communication. Typically, P2P systems use randomized host IDs which destroys any locality that might have been inherent in the IP addresses or the names of the hosts. In this way, P2P communication can result in a high stretch. We propose a locality preserving structured P2P system that supports efficient local communication and low stretch. While this system was optimized for resolving domain names, it will also provide a low stretch to other applications and it can be combined with existing replication schemes to optimize the response times even further.*

*Keywords: Distributed Systems, DNS, Peer-to-Peer Networks, Pastry, Low Stretch, Overlay Simulation*

## 1 Introduction

The central goal of this paper is to describe a design of a low-latency P2P lookup service for global scale deployment. Today, there is one global-scale distributed lookup service that is deployed and used in the Internet: DNS. It is a client-server solution that uses caching and replication to achieve mostly good response times. DNS was developed in the early days of the Internet and has already been considered unreliable and insecure for many years. Jung et al. [11] measured that less than 65% of the DNS queries are answered successfully. Some of the unreliability of the DNS can be attributed to operator errors: Danzig et al. [6] found that most of the DNS problems are caused by configuration issues. However, there are also some inherent protocol properties that make DNS unreliable.

In the recent years, the missing security features of DNS have been a major point of concern. DNSSEC can help to address the security issues. It allows a separation of server management and authority. DNS data can be signed by the owner of the data and can then reside wherever it is useful. DNSSEC has however been in development for many years and it is not clear if or when it gets more widely deployed. Hence, several research groups have started to focus on P2P systems for replacing or augmenting DNS [3, 17, 15] .

While providing simplified management, P2P networks lack routing efficiency leading to high delays. Our work focuses on reducing the stretch of P2P networks, i.e., the ratio between P2P transmission delay and IP transmission delay. A lower stretch would result in better query times and reduce unnecessary bandwidth consumption. Our approach is orthogonal to approaches that optimize the replication of objects in the P2P network like [17]. It is based on the observation that we are often interested in accessing hosts in the same domain of the DNS namespace, e.g., in the same country. These hosts are often also geographically closer than hosts in other domains or hosts in other countries. We show how one can use this information in the creation of host IDs to reduce the stretch. Our evaluation shows that we are better than Pastry (from which our system is derived) even for random traffic patterns. For localized queries, we expect an even better performance.

The rest of this paper is structured as follows. Section 2 describes related works that optimize peer-to-peer routing. The basic ideas of our routing scheme are presented in Section 3. Sections 4 and 5 describe our overlay organization in detail. Performance evaluation is presented in Section 6. Section 7 concludes the paper.

## 2 Related Work

The time to locate objects in a DHT can be reduced in several ways, e.g.: (A) The access locality is exploited: each object is stored near the location where it is most likely to be used. This includes replication and caching strategies. (B) Proximity Neighbor Selection (PNS): the routing steps towards the desired object are optimized with respect to network delay. To do this, one can (B.1) measure the round trip time to nodes that are candidates for entries in a routing table, or (B.2) use existing knowledge about the underlying network structure to select the closest candidate.

## 2.1 Locality

Some applications would be able to provide locality hints for helping to place objects in a peer-to-peer network. For example, consider that a P2P network is used to locate text documents. One could use the language of a document as a hint for the most likely countries in which the users of a document might be located. Other applications might use URLs that can give us similar hints. Since the hash function of a DHT usually destroys these locality hints, caching and replication is needed to exploit access locality. Data caching and routing information caching are often used in P2P networks to reduce response times.

Pastry [20] uses Leaf-Set-Replication[1] of data to optimize the distribution in the network. Since the peer-IDs are randomly created, replicas are evenly distributed in the topology of the underlying network. This means that each user should have a nearby replica. Applications based on Pastry can also exploit local route convergence to find good locations for cached objects.

Tapestry [25] allows the selection of the replication level which defines how many copies of an object are stored in the network. This level is implemented by defining a common prefix length that a replication host and the object ID must share. The shorter the prefix length, the higher is the number of replicas. If node IDs are generated randomly, the replicas are evenly distributed in the network topology. This allows to adjust the replication level to the desired mean time for fetching an object.

Beehive [16] uses this technique to build a pro-active caching framework that adjusts the replication level of an object based on the request rate for this object. Beehive is used in CoDoNS [17] to build a P2P-based domain name system. CoDoNS reaches O(1) lookup performance. The number of hops that is allowed for a lookup can be adjusted. To achieve a low number of hops, e.g., 0.5 hops average latency, the local cache hit rate must be very high, i.e., 50% in this case.

## 2.2 PNS

In different P2P solutions the stretch is reduced by optimizing the routing tables of peers. When a node has to forward a message, it should pick a nearby peer. This is often reached by measuring the round trip time to peers that are available for a certain position in the routing table. Sometimes this generates a substantial amount of traffic due to the high number of peers. Fortunately, the traffic can be reduced. In such situations it is sufficient to probe a small number of them [8]. To reduce the PNS effort even more, we plan to use Vivaldi [4] in the future.

[1]The leaf set contains the direct successors and predecessors of the node in the ID space. Details can be found in [20].

Another solution to the same problem is landmark routing or clustering [24, 19]: peers cluster around distinguished peers called landmarks. The proximity to and between landmarks is assumed to be known by the use of network coordinates or RTT measurement data. This information can be used as an approximation of the proximity of nodes that have similar distance vectors to the landmark nodes.

## 2.3 Intrinsic problems of DHTs

Despite the availability of these technologies to optimize routing table information, P2P routing is still not widely accepted and used for low latency services. Objects are stored too far away from their users and replication schemes like Beehive (used in CoDoNS) cannot exploit the access patterns of local users or user groups in a global system. Instead, such solutions replicate a high number of the most popular objects on hosts without a guaranteed benefit for the local user.

DNS has good locality properties. Computers in a LAN are often served by the same DNS server and have a long common suffix in their DNS names. Also Internet users usually visit web sites that have a cultural proximity, i.e., they are written in the same language or located in the same country. This proximity can be exploited by DNS and improves, together with caching, the performance. DHTs often destroy such proximity information, contributing to a rather slow lookup and poor proximity between the target object itself and the user.

While other approaches try to exploit common keywords to express cultural proximity like Lu et al. [13] to speed up P2P lookups, we exploit the hierarchical structure of DNS names. If some user looks up *harrypotter.warnerbros.co.uk*, it is more probable that user will also lookup *www.amazon.co.uk* instead of *harrypotter.warnerbros.fr*. In general, a country code of a URL may tell us where the object may be used frequently. Also, typically it is more likely to lookup the IP addresses of hosts within its own domain than of hosts in other domains. The postfixes of host names indicate their location in the DNS name space and in this way provide some locality information: (A) not only is it more likely to resolve names in its own proximity of the DNS hierarchy but (B) there is also a correlation between the geographical proximity of hosts and their proximity in the DNS hierarchy. To preserve these two correlations, our hash scheme preserves the DNS hierarchy in the generated host IDs. For locating objects, we are using a prefix routing scheme like the one of Pastry [20] or Tapestry [25]. Our evaluation shows that even for random access patterns (i.e., without using locality factor (A)), our scheme harnesses (B) provides a better stretch than Pastry (from which our system is derived). Our goals are also similar to those of SkipNet [10]. However, optimizing routing tables using PNS is much easier in our system than in SkipNet, where it is necessary to maintain additional routing state for this purpose.

## 3 DNS-based Clustering

In prefix-based P2P schemes, peers cluster around common prefixes. In DNS, names are contained in zones. A zone is an organizational unit. Starting from the root zone, authority can be delegated to sub-zones. A zone manifests itself in a name as a suffix.

The King approach [9] assumes that Internet hosts often cluster around their authoritative DNS servers. These hosts share a common suffix, i. e., the name of their DNS zone. Since the KING tool achieved good results, we exploit this knowledge in the design of our hash scheme. In contrast to that, DNS gives us a poor indication for the inter-domain proximities. Country code domains give us some benefit which we exploit, but generic top level domains (.org, .com and .net) do not. However, we assume that most communication is done within a domain [2].

When calculating an ID for a name, we compute for each label of the node's DNS name a hash value. These values are then concatenated to build the ID. This construction itself does not cause collisions because names in the hierarchical DNS name space are unique. A novelty in this approach is that IDs can have different length. This makes the comparison of IDs slightly more difficult: the identifier space is not a ring with a *modulo* operation and is infinitely large. At the same time it gains direct support for domains with different granularity in their organizational structure.

Our performance evaluations are based on transit-stub topologies. These topologies are usually interpreted as router-level models because they explicitly group vertices into domains and reflect that grouping in the connectivity between vertices. In our case, they reflect the DNS hierarchy. We assume that links within the same DNS domain have a lower delay than inter-domain links. As mentioned before, this assumption is based on the observations of Gummadi et al. [9].

In our overlay (DNS-Pastry), nodes within the same DNS domain are neighbors in the ID space. Hence, the distances between leaf set nodes in a DNS-Pastry overlay are much smaller than in a Pastry or Chord overlay.

[2]For connections to web sites this is not true but usually, there are many applications that satisfy this assumption, like e-mail clients that fetch new e-mails from the local mail servers in short intervals, connections to domain controllers for user authentication, group communication, connections to the local NTP server, etc.

### 3.1 Comparing IDs

To make routing decision, IDs have to be comparable. For computing the ID of a peer, a hash function $\pi^l_{name}$ is used: $\pi^l_{name}$ returns a hash value for the $l$-th label of the peer name, where $l = 0$ selects the highest label, i.e., the top level domain. The ID of a peer which *name* consists of $k+1$ labels is defined as:

$$\pi^0_{name} \cdot \pi^1_{name} \cdot \ \dots \ \cdot \pi^k_{name},$$

where "·" is the concatenation operator.

The order we use on IDs is lexicographical order, i.e., we interpret the digits as letters of an alphabet (and not numbers) and use the alphabetical order to compare two IDs.

## 4 Routing

The routing algorithm is very similar to Pastry, but it provides path locality, like defined by Harvey et al. in [10]. Path locality, which is an additional security feature, prevents packets from being routed through external hosts in the case that source and destination of the packet are in the same domain. This property is actually provided by the combination of the original routing algorithm of Pastry and our approach to generate IDs: because the routing ensures that the common prefix never gets shorter and our hash algorithm ensures that a common domain is represented by a common ID-prefix. The routing information consists of a prefix-routing table and a leaf set. The leaf set covers a certain part of the ID space. It contains up to $N$ node handles: the $(N-1)/2$ nodes with the closest smaller IDs, the current (local) node, and the $(N-1)/2$ nodes with the closest greater IDs. If there are not sufficiently many nodes with smaller (larger) IDs, the leaf set is padded with nodes that have the largest (smallest) IDs in the system. The worst case routing complexity depends on the number of digits in the destination's ID since the routing algorithm assures to add at least one digit with each hop to the possibly non-empty prefix match that source ID and destination ID already have. During our measurements we have seen an *O(log N)* routing complexity.

Say, we want to locate an object with ID $id_d$. To do so, the routing algorithm sends a message $m$ to locate $id_d$. A peer $L$ that receives (or sends) $m$ performs the following steps:

1. IF there are peers in the routing table or the leaf set of $L$ with a longer prefix match with $id_d$ than the local node $L$, pick the one with the longest prefix match. If there exists multiple nodes with a longest prefix, forward to the closest one of them in terms of the round trip time. (Select one at random if there is more than one node with the longest prefix and the shortest round trip time.)

2. ELSE IF there is a node $N$ in the leaf set or the routing table of $L$ that has the same prefix match with $id_d$ as $L$ but is lexicographically closer to $id_d$ than the local node $L$ (i.e., $L < N < id_d$), then $L$ forwards the message to $N$.

3. ELSE deliver it to the application on the local node.

## 4.1 Routing table

The routing table is very similar to the one used by Pastry. The number of columns is set to the base $2^b$ which is used to encode IDs, e.g., for base 16 (i.e., $b = 4$) there are 16 columns. We use function $shPfD(id_1, id_2)$ to denote the number of digits in the shared prefix of IDs $id_1$ and $id_2$, i.e., they match in the first $shPfD(id_1, id_2)$ digits. The term $id_1[i]$ refers to the $i^{th}$ digit in the ID $id_1$.

Consider that we want to store ID *t* in the routing table. The column of *t* is determined by $t[shPfD(c, t)+1]$ which is the first digit after the common prefix $shPfD(c, t)$ if the length of *t* is at least $shPfD(c, t) + 1$. Otherwise, *t* is a prefix of $c$ as discussed in Section 4.2. $t[shPfD(c, t)]$. The line is determined by the length of the common prefix. The first line contains node handles whose IDs do not have a prefix match (except for the the entry in column $c[1]$). In the second line $shPfD(id_1, id_2)$ equals one (except for the column $c[2]$) and so on. The number of lines is equal to the length of the ID of the current node plus 1 because we have to gain one prefix digit in each routing step.

Figure 1 shows a routing table for the node 02312 and $b = 2$. The table has six lines for the reasons mentioned before. The number of columns is four since $b = 2$, which means that possible digits are $\in \{0, 1, 2, 3\}$. The leaf set size is eight, which means that at most four nodes nodes are stored in each direction. For simplicity, Figure 1 shows only one entry for each field in the prefix routing table. We can store more than one for better fault tolerance. One of them will be chosen as next hop depending on the round trip time.

The variable length of IDs leads to a little more complex indexing in the routing table (see below) but at the same time our approach offers a novel ability to the overlay: like in DNS, an optimization of the communication between hosts within the same organizational unit is possible. The organizational structure can be as fine grained as desired. The structure is, like in DNS, reflected by the names of the nodes. Since these names are directly mapped to node IDs, the hierarchical structure of the overlay is represented by the prefix-routing table.

## 4.2 Routing table indexing

If peer IDs with a fixed length are used, the indexing of the routing table is very straightforward. A message with destination $id_2$ is forwarded by peer $id_1$ to the entry

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | | **1**2 | **2**11211 | **3**1131 |
| 1 | 0**0**1 | 0**1**121 | 0 | 0**3**311 |
| 2 | 02**0**12 | 02**1**3031 | 02**2**10 | 02 |
| 3 | 023**0**0 | 023 | 023**2**01 | 023**3**32 |
| 4 | 0231**0** | 0231**1** | 0231 | 0231**3**1 |
| 5 | 02312**0**1 | 02312**1** | 02312**2**3 | 02312**3** |

| | | | | |
|---|---|---|---|---|
| b = 2 | 0202 | 0203 | 02013 | 023113 |
| ID:02312 | 0231202 | 02312022 | 0231211 | 0231213 |

**Figure 1. Routing information for node 02312**

$$(column, line) = (id_2[m + 1], m)$$
where $m = shPfD(id_1, id_2)$ and $m < length(id_1)$ and $m < length(id_2)$

in its prefix-routing table.

Because we are using a variable ID length, we need to extend this rule. As noted, we only use the above rule if the prefix match of $id_1$ and $id_2$ is smaller than the length of any of the two IDs. If $id_2$ is a strict prefix of $id_1$, we use position

$$(column, line) = (id_1[m], m)$$
where $m = shPfD(id_1, id_2)$ and $m = length(id_2)$ and $m < length(id_1)$

In turn, if $id_1$ is a strict prefix of $id_2$, we use position

$$(column, line) = (id_2[m + 1], m + 1)$$
where $m = shPfD(id_1, id_2)$ and $m = length(id_1)$ and $m < length(id_2)$

Note that the last line in the routing table is needed to ensure logarithmic routing performance.

## 4.3 Algorithm for Weighted Proximity-Neighbor-Routing

Depending on the density of nodes in the identifier space, there could be more than one node possible for a certain position in the routing table. In this case, a routing decision is made based on physical proximity, i.e., the measured round trip time, and logical proximity, i.e., the distance in the ID space. For this purpose, a weighted ranking is needed, which is defined by the following function.

$$min(\alpha * |destId[l] - nextHopId[l]| + \beta * |nextHopRTT|)$$
where $l = shPfD(destId, nextHopId) + 1$ and $length(destId) \geq l$ and $length(nextHopId) \geq l$

Choosing a stronger $\beta$ (i.e., $\beta > \alpha$) would increase the work load on overlay nodes due to the higher number of hops for the sakes of better response time. Choosing a stronger $\alpha$ (i.e., $\alpha > \beta$) would reduce the number of hops but increase response time.

Having multiple options for each routing decision does not just improve performance but node failures can be tolerated much better too.

## 5 Bootstrapping

To enter the overlay network, a new node must know at least one overlay peer. This peer is used to send a message to the new node's ID. The node, that is responsible for this ID is asked to send its routing table and leaf set information.

We extended the bootstrapping mechanism for Proximity Neighbor Selection. After retrieving the routing information, the node can iterate over all nodes it knows and ask them for their routing information. Then candidate nodes for a certain position in the routing table are compared with respect to proximity, i.e., RTT.

### 5.1 Updating routing information

To keep routing information up-to-date, each message that is forwarded is analyzed. In the case that the sender of the message could fill a field in the routing table, a PING message is sent to get RTT information. In the future we want to use Vivaldi [4] to reduce the number of ping messages. A periodic maintenance of the routing table is not used, instead we check and correct the routing table on use. This is similar to Pastry.

## 6 Performance Analysis

Our main requirements for selecting a simulator were to evaluate the scalability of our solution and to compare the routing performance with IP-routing to compute the stretch. Furthermore, we wanted to compare it to other P2P solutions. Hence, we looked for a simulator that already included several common P2P protocols and allowed us to simulate at least 1000 nodes. To be able to use the algorithms applied in the simulation runs also in a real application, we payed attention to the fact that the simulator sticks to the common API for structured P2P overlays [5]. Another advantage of maintaining this API is that application developers can easily use our simulator and the overlay to simulate their application on it. We also preferred Open-Source-Software to be able to perform some extensions. Since our overlay is based on a mapping of the underlying hierarchical structure into the ID space, the simulator had to be able to use topological network information. We examined the several simulators [7, 14, 1, 2, 12, 23, 21] and decided to use and extend PlanetSim.

PlanetSim [2, 12] is very well documented and comes with implementations of Chord and Symphony. It is a implemented as single threaded application and does not support topological information. However, it was able to simulate a Chord-overlay with 100,000 peers on a Pentium 4 with 1GB RAM. The stabilization of the ring took 46 hours. PlanetSim provides a very clear separation of the network layer, overlay layer, and application layer. Hence, it is easy to try new applications on top of existing overlay networks or vice versa.

We added three extensions: (1) a topology front-end that parses and imports topology files, (2) a statistics back-end that collects statistics like delay times and hop counts during simulation, and (3) a graphics backend that is able to display the network topology and the overlay topology on top of it. Of course, the latter only makes sense for small networks. The output format is GML. The network layer had to be extended to use the given topology and report delays to the statistics component.

### 6.1 Evaluation

The main focus of the simulation is to figure out, how efficient the overlay routing of our clustering-based overlay (DNS-Pastry) behaves in comparison to other overlays and to IP-routing. The important characteristics are (1) the average number of hops, (2) the round trip time (RTT), and (3) the stretch.

We created Transit-Stub-topologies with a three-level hierarchy, which are very common in DNS names, e.g., *www.tu-dresden.de*. For the link latencies we use a Gaussian distribution[3] with a mean of 100[4] for intra-transit domain links, 10 for stub-transit links, and 1 for intra-stub links and a deviation of one fifth of the mean. These values are derived from the CAN simulation [18] for better comparison.

Since our implementation is very similar to Pastry, we also added a Pastry implementation to the PlanetSim project. Additionally, we tried to use the existing Chord implementation, which was however quite buggy, so that we had to revise it. The comparison to Chord might be slightly unfair because it does not use PNS.

We were not able to simulate 100,000 nodes like in [12] since our enhanced simulator needs much more memory for statistics and topology management. So we created overlay networks from 500 to 4000 peers, in steps of 500. The nodes of the peer-to-peer overlay were chosen randomly for each size from a topology of 4420 nodes. The simulation was done on a Intel Pentium D 820 with 2048 MB RAM running

[3]For simplicity: all timings of topologies mentioned in the following parts of this paper refer to the mean.

[4]We use abstract time units instead of ms, just to show the ratio of delays between different parts of the network.

Ubuntu Linux and Sun Java-VM 1.5.

We compared Pastry, Chord and our DNS-Pastry with respect to the average number of hops, the round trip time and the stretch. Our benchmark application generates 100,000 ping messages between randomly chosen peers. Obviously, this kind of application has no locality properties. Hence, the results of the benchmark are interesting for a wide range of applications. However, we think that the benefits of our overlay are even higher for applications which can exploit data and request locality. Of course, all of the measured values depend on size and structure of the underlying network topology and the parameters of the overlay networks, so they can be compared to each other, but not to values of other benchmarks. Pastry and DNS-Pastry use PNS to optimize routing information. In our case, up to three entries are contained in one field in the routing table to choose the best proximity and allow some resilience to node failures. The leaf set contains 32 entries, 16 in each direction. The Chord implementation does not use PNS. The successor list contains 16 entries.

## 6.2 Results

Figure 2(a) depicts the average number of hops for 100,000 ping messages in four different simulations. The number of hops is often a good performance indicator when configuring an overlay network. All overlays we used reach *O(log N)* complexity where N is the number of overlay nodes. Nevertheless, some fine-tuning is always possible. The trade-off is the size of the routing information. If, e.g., the base *b* of a digit in Pastry's prefix routing is increased, the number of columns in the routing table increases to $2^b$. This would help to reduce the average number of hops because Pastry needs $O(log_{2^b} N)$ hops to reach a destination. In the case of Chord we use a successor list of 16 nodes, i.e. $2^4$. For a network size, e.g., 4000 ($\approx 2^{12}$) nodes, we can reach a destination within 8 hops.

Another measure that reduces the number of hops is to increase the leaf set size. We ran DNS-Pastry with 4, 8, and 16 nodes in the leaf set, reducing the average hop count by 0.2 to 0.3 for each doubling of the leaf set size.

In Figure 2(a) we can see that Pastry needs less overlay hops than Chord and our DNS-Pastry. We also simulated an application that only sends ping messages within on Top Level Domain (TLD) to show the performance gain for local communication. We found, that DNS-Pastry is, as expected, much better in this case, since the domain structure is reflected by the routing information. The graph *DNS-Pastry TLD* in Figure 2(a) shows this result.

To find out, whether a low hop count guarantees a good response time, we also measured the round trip time for this experiment. Figure 2(b) shows the results. We observed that DNS-Pastry has a much better RTT for random ping messages. The reason is that it efficiently accelerates the last steps of the routing that go through the leaf set. The PNS of Pastry ensures that the first few hops are done to nearby nodes. The reason is that there are many options as long as the common prefix is short, but the number of choices drops exponentially with a growing prefix. The last hops of the Pastry routing are usually very long. This is where DNS-Pastry does much better and gains the RTT advantage.

Of course, the RTT of the intra-TLD simulation of DNS-Pastry has much lower values. We can see that DNS-Pastry needs less RTT than Pastry but more hops. This phenomenon has already been observed by Xu et al. [22] when constructing a hierarchical P2P routing algorithm. The number of routing hops is getting larger as the hierarchy depth increases. However, the user perceived performance depends more on delay and not on the number of hops. Therefore, we measured the stretch of DNS-Pastry. Figure 2(c) shows the comparison of stretch for the different overlay networks. Stretch is defined as the ratio $\frac{\text{overlay distance}}{\text{IP distance}}$.

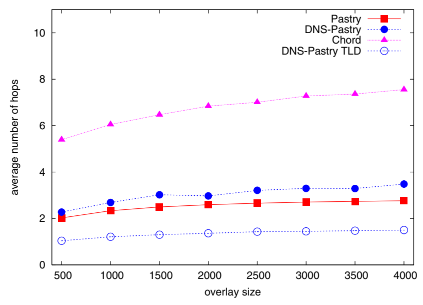
We can see that DNS-Pastry does not incur as much delay penalty as Pastry or Chord. The difference between DNS-Pastry with arbitrary ping destinations and intra-TLD destinations is almost negligible. This indicates that DNS-Pastry routing is quite efficient even for applications without good locality properties.

Since the good stretch of DNS-Pastry is due to the fact that long distance hops are efficiently avoided, we also examined the influence of different timings in the transit-stub topology.

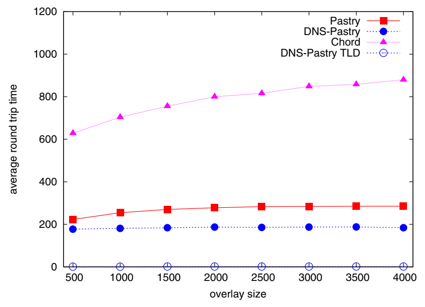
Therefore, we created two more topologies. For better comparison, we used timings that were also used in other simulations, like done by Ratnasamy et al. [18]. Our first topology had link latencies of 100 for intra-transit domain links, 10 for stub-transit links, and 1 for intra-stub links. Now we add a 20-5-2 and a 10-10-10 topology. These timings can help to examine the case that a domain does not cluster advantageous. The 10-10-10 timing would also be a case where the King tool [9] would not work properly since it depends on the assumption of clustering around DNS names. According to the authors of [9] this does not seem to be a common case. Our overlay (Figure 3(a)) can deal with this situation but loses some performance. This might be the case for communication between different domains within generic TLDs. However, if we compare it to the results of Pastry and Chord in Figure 2(c) we still see better performance.

Our transit stub topology does still not reflect the real characteristics of DNS zones in the Internet. There are two enhancements that we apply therefor: (1) slow intra-stub links and (2) fast intra-transit links.

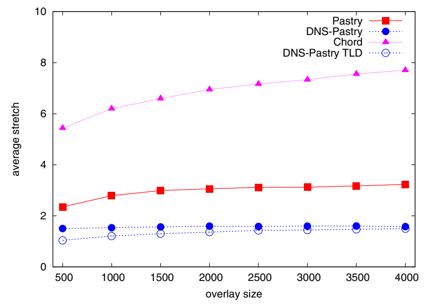
Some zones are not as tightly coupled as we assume in our H(100,10,1) and H(20,5,2) topologies. Examples could be *bbc.co.uk* and *amazon.co.uk*. We introduce a slowdown



(a) Average hop count

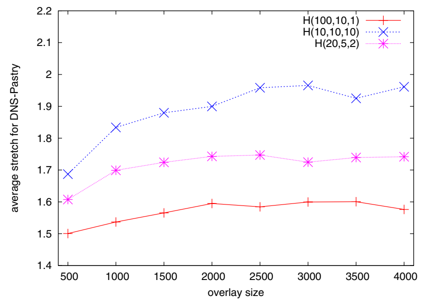


(b) Average round trip time

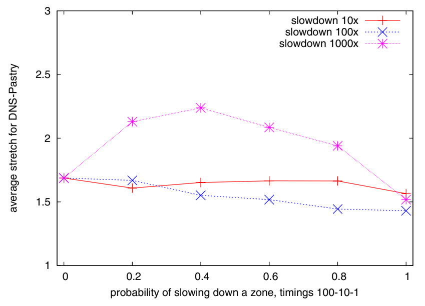


(c) Average stretch

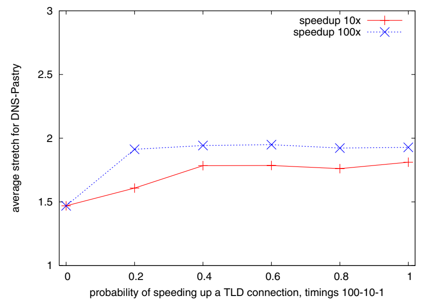
**Figure 2. Comparison of Pastry, DNS-Pastry, Chord**



(a) Differnet network delays



(b) Slow intra-stub connections



(c) Short intra-transit links

**Figure 3. DNS-Pastry average stretch**

factor for the intra-stub links and a probability that a zone is slowed down. In Figure 3(b) you can see, how DNS-Pastry behaves in this cases for a H(100,10,1) topology. We used 100,000 random ping messages and a probability of 0.2 for a factor of 10 for intra-transit speedup, which we are going to explain later on. A slowdown factor of 10 or 100 is not a problem, while factor 1000 increases the stretch significantly. However, factor 1000 is rather unrealistic, since it would mean that an intra-stub link is 10 times slower than intra-transit links. The probability that a zone is slowed down seems not to have an important influence.

Sometimes the delay between hosts in different TLDs is quite short, like between *.de* and *.nl* or between *.mil* and *.gov*, which are both used in the US. Therefore, we created topologies with a speedup factor for intra-transit links and a probability that a speedup is applied for a certain link. The results are depicted in Figure 3(c). For this simulation a probability of 0.2 for a slowdown factor 10 for the intra-stub slowdown has been used. We expect, that there might be many such links in the Internet, like between country code domains of countries that are geographically close to each other. However, the share of such short links among the intra-stub links has almost no influence on the stretch.

## 7 Conclusion

We presented a new design for a peer-to-peer overlay network. It can support and exploit locality correlated with the position of nodes in a hierarchical name space. Due to the novel design of its ID space, it also supports hierarchical names within organizations to speed up communication within large organizations. The deeper name space of these organizations will be reflected by longer IDs in the overlay. At the same time, prefixes in the application name space are used as locality oracle. This locality information is also reflected by overlay IDs.

We have shown that our design provides good performance for random ping messages. This proves that it is usable as multi purpose overlay - even for applications that do not provide locality in its access patterns. The only prerequisite is that nodes have DNS names.

For our performance evaluation we used a simulator with a layered architecture. We maintain the common API [5] thus it can easily be used to simulate any application on top of our peer-to-peer network. There is some future work: To evaluate our overlay design for workloads/applications that provide communication patterns with high locality, we want to simulate a DNS resolver application. We also plan to use PlanetLab and the simulator to explore the behavior of our overlay during node failures and network partitions.

## References

[1] Overlay Weaver: An Overlay Construction Toolkit. http://overlayweaver.sourceforge.net, October 2006.

[2] planetSim: An Overlay Network Simulation Framework. http://planet.urv.es/planetsim, October 2006.

[3] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a peer-to-peer lookup service. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 155–165, London, UK, 2002. Springer-Verlag.

[4] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, New York, NY, USA, 2004. ACM Press.

[5] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, February 2003.

[6] P. B. Danzig, K. Obraczka, and A. Kumar. An analysis of wide-area name server traffic: a study of the internet domain name system. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 281–292, New York, NY, USA, 1992. ACM Press.

[7] T. M. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling. p2psim: a simulator for peer-to-peer (P2P) protocols. http://pdos.csail.mit.edu/p2psim/, April 2006.

[8] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, New York, NY, USA, 2003. ACM Press.

[9] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary internet end hosts. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 5–18, New York, NY, USA, 2002. ACM Press.

[10] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *In proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, March 2003.

[11] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS performance and the effectiveness of caching. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 153–167, New York, NY, USA, 2001. ACM Press.

[12] P. G. López, C. Pairot, R. Mondéjar, J. P. Ahulló, H. Tejedor, and R. Rallo. PlanetSim: A New Overlay Network Simulation Framework. In *SEM*, pages 123–136, 2004.

[13] Y.-E. Lu, S. Hand, and P. Lio. Keyword searching in hypercubic manifolds. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 150–151, Washington, DC, USA, 2005. IEEE Computer Society.

[14] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A Survey of Peer-to-Peer Network Simulators. *Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK*, 2006.

[15] K. Park, V. S. Pai, L. L. Peterson, and Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *OSDI*, pages 199–214, 2004.

[16] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI*, pages 99–112. USENIX, 2004.

[17] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 331–342, New York, NY, USA, 2004. ACM Press.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *SIGCOMM '01: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172, San Diego, California, USA, 27–31 August 2001. ACM Press.

[19] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *21rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, 2002.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[21] N. S. Ting and R. Deters. 3ls - a peer-to-peer network simulator. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 212, Washington, DC, USA, 2003. IEEE Computer Society.

[22] Z. Xu, R. Min, and Y. Hu. Hieras: A dht based hierarchical p2p routing algorithm. *icpp*, 00:187, 2003.

[23] W. Yang and N. Abu-Ghazaleh. GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent. In *MASCOTS*, pages 425–434, 2005.

[24] B. Y. Zhao, Y. Duan, L. Huang, A. D. Joseph, and J. Kubiatowicz. Brocade: Landmark routing on overlay networks. In P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors, *IPTPS*, volume 2429 of *Lecture Notes in Computer Science*, pages 34–44. Springer, 2002.

[25] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Jan. 2004.