eQuus: A Provably Robust and Locality-Aware Peer-to-Peer System

Thomas Locher, Stefan Schmid, Roger Wattenhofer
{lochert, schmiste, wattenhofer}@tik.ee.ethz.ch
Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland

Abstract

Peer-to-peer systems (p2p) are highly dynamic in nature. They may consist of millions of peers joining only for a limited period of time, resulting in hundreds of join and leave events per second. In this paper we introduce eQuus, a novel distributed hash table (DHT) suitable for highly dynamic environments. eQuus guarantees that lookups are always fast—in terms of both the delay and the total number of routing hops—, although peers may join and leave the network at any time and concurrently.

1 Introduction

Much research in the last few years has been devoted to the development of efficient, structured peer-to-peer network overlays,and a plethora of peer-to-peer (p2p) systems [8, 11, 13, 16, 18] has been proposed. All these systems belong to the class of distributed hash tables (DHT). A DHT is a decentralized and distributed system that partitions an ID space among all participating peers, which are responsible for all keys that lie in their respective fraction of the ID space. The resulting structure is referred to as an *overlay network*, a common term for a network built on top of an underlying network. In a DHT, looking up the peer that is responsible for any key can be done efficiently, typically requiring $\mathcal{O}(\log n)$ hops, where n denotes the current number of peers in the system. Practically all of the proposed DHTs further guarantee small routing tables at each participating peer and also a good load balancing among all peers.

Unfortunately, most proposed peer-to-peer architectures only maintain these properties in static environments. It has been pointed out [15] that p2p systems are usually highly dynamic in the sense that peers join and leave the system frequently and concurrently. Therefore, the performance—and success!—of a peer-to-peer system in practice crucially depends on the system's capability to handle *churn*[1].

In this paper, we present a novel DHT, called *eQuus*, which is specifically designed to cope with high dynamics and failures. eQuus provably maintains properties such as a low peer degree and lookup operations in $\mathcal{O}(\log n)$ hops in the presence of highly transient peers. In particular, unlike other systems, our system does not require a large message overhead in order to adapt itself quickly to changes, detecting and compensating both random and correlated failures efficiently. Finally, a redundancy mechanism is employed to minimize the probability of data loss. Note that, in eQuus, only indexing information is stored. In the remainder of this paper, the terms "data" and "data item" always refer to the index data stored at each *node*[2].

Besides the optimization of the classic p2p criteria described so far under churn (peer degree, network diameter, etc.), an important design goal of eQuus is *locality awareness*. The locality awareness criterion is somehow complementary to the network diameter (expressed in the number of hops), and captures the *latency* of a lookup operation: If in each hop during the lookup operation, a packet is sent between geographically distant peers, the total latency is large. By means of a smart neighbor selection policy, eQuus' routing scheme guarantees that the distance traveled by a packet in the network is not much larger than the shortest distance between the source and the destination peer. As a result, excellent response times for the lookup operations are achieved.

As in many existing systems, the nodes in eQuus are arranged in a hypercubic topology. However, *groups* of nodes that are close to each other according to the chosen proximity metric form the vertices of a partial hypercube. Within such a group, all nodes share the same node ID and all nodes know about each other[3]. Since the nodes in each group form a complete graph, these groups are denoted *cliques* (cf Fig. 1).

In addition to the links to all other *clique members*, i.e. nodes that belong to the same clique, each node has links to nodes in other cliques. These additional links ensure that the entire system is connected and that paths from any node to another node are short. Data items are replicated among all nodes of a particular clique thereby introducing a natural form of redundancy. Since these nodes are geographically close to each other (in terms of latency), consistency can be maintained efficiently. What is more, this approach reduces the communication overhead in the network in general, since nodes joining and leaving cliques only trigger communication among the nodes in the same clique; all other nodes in the system are not affected. Changes in the routing tables are only necessary if entire cliques appear or disappear, due to the arrival or departure of a large number of nodes. This entails that the dynamics of the network can be controlled better, for the life time of cliques is much longer than the life time of individual nodes.

Throughout this paper, we assume that all nodes are uni-

[1] In p2p lingo, the fast and permanent joining and leaving of peers is called churn.

[2] Throughout this paper, the terms *node* and *peer* are used interchangeably.

[3] We will say that "node v knows about w" or alternatively "there is a link from node v to w," if node v stores the address of node w in its routing table.

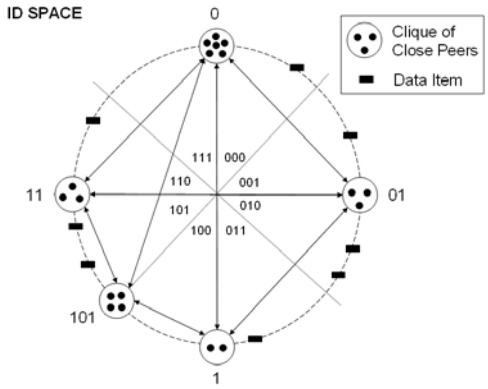

Figure 1. An example network consisting of 5 cliques is shown. Nodes that are close-by belong to the same clique and share the same ID. All nodes within the same clique are responsible for the same set of data items.

formly distributed in a two dimensional *Euclidean space*. While this assumption is clearly not an ideal approximation of the node distribution in large networks such as the Internet, it allows for a simple formal analysis of some of the system's properties. Moreover, we believe that our results indicate that eQuus also performs well in any real network.

The rest of the paper is organized as follows. In Section 2, related work in this field is summarized. The design of the entire DHT is described in detail in Section 3. The ability of eQuus to cope with random failures and highly dynamic networks is treated in Section 4. The results of the simulations that have been run to show the good locality properties as well as the theoretical results are presented in Section 5, and Section 6 concludes.

2 Related Work

Subsequent to the seminal work of Plaxton et al. [10], many p2p systems have been developed, e.g. [8, 11, 16, 18]. All of these systems are scalable and provide fast lookups while each individual peer needs to store only a small amount of information about other peers in the network. In order to effectively cope with churn, any p2p system further has to provide specific mechanisms ensuring a high degree of fault-tolerance. While some solutions have been analyzed in this respect, it seems that only a few systems inherently incorporate fault-tolerance in the sense that robustness was a clear design goal from the beginning. We believe that this stands in contrast to the high dynamics observed in today's p2p networks (e.g. [15]).

In the following, we first review relevant related work on fault-tolerance. Fiat et al. [4, 14] have studied peer failures occurring in a worst-case fashion, for example caused by an adversary or a p2p worm exploiting the overlay network structure. The authors introduce a system where, with high probability, $(1-\varepsilon)$-fractions of peers and data survive the adversarial removal of up to half of all nodes. However, the whole network has to be rebuilt from scratch if the total number of peers changes by a constant factor. Also Li et al. [7] analyze their system from a worst-case perspective. Using rigorous, formal proofs they show that their system tolerates concurrent, ongoing and asynchronous joins and leaves of peers. Unfortunately, leaving peers are not allowed to crash; instead, they have to execute an appropriate exit protocol. A more practical study by Rhea et al. [12] compares DHTs by simulation and shows that several structured p2p overlays cannot handle churn rates as high as those observed in today's p2p networks.

eQuus shares the most commonalities with the work by Kuhn et al. [6]. Similarly to eQuus, [6] achieves a better robustness by having more than one peer responsible for each ID, as opposed to assigning a unique ID to each node. In contrast to our work, their system has higher maintenance costs as it requires a background process estimating the current network size in order to balance the ID assignment through a global operation. On the contrary, eQuus reacts to imbalances using *local* merges or splits of adjacent IDs only. An additional advantage of eQuus is that it has a low expected stretch, as described in a later section.

In order to evaluate the performance of lookup operations in p2p systems, usually only the total number of hops is considered. In systems such as [6] or [16], the neighbors of a given peer are determined by applying hash functions to the peer's IP address and potentially other parameters. Thus, the resulting structure is completely independent of the actual geographic peer locations. It is therefore possible that, with each hop, a node on a different continent is contacted, even if the target node is close-by, which is clearly not desirable. Furthermore, when data items are replicated among nodes with numerically close identifiers, constantly ensuring that the data items are stored on the correct nodes is potentially a costly operation, since these nodes might be very far away.

The approach taken by Pastry [13] when performing a lookup is to choose the peer that is the geographically closest among a possible set of neighbors. According to their simulations, Pastry achieves a low stretch of around 1.3 to 1.4. This is a heuristic approach, and there is no guarantee on the worst-case stretch. Using results from *name-independent routing* (e.g. [9]), Tulip [1] and LAND [2] achieve *provably* low stretches. Unfortunately, however, these systems lack the robustness property.

The main contribution of eQuus is to unify well-known and novel techniques, resulting in an efficient, structured overlay that comprises *both* provable fault-tolerance *and* provable locality-awareness. As an additional feature, our system strives to achieve these goals with a minimal maintenance overhead by carefully specifying the granularity at which synchronization is required between nodes that share data.

3 System Overview

In eQuus, groups of nodes that are close-by form *cliques*. Within such a clique, each node has the same ID, which is a bit string of a predefined length d. The length d of the IDs is referred to as the *dimension* of the network. Every clique has its own unique ID. Since these nodes share the same identifier, they are also responsible for the same fraction of the

ID space. This has two interesting properties. First of all, these nodes ensure a certain degree of redundancy that is required lest data is lost due to the sudden departure or failure of a particular node. Second, if those nodes storing the same information are close to each other, establishing consistency among these nodes can be done quickly due to the short distance between those nodes.

Since the degree of each node should not exceed $\mathcal{O}(\log n)$, the number of nodes in a clique has to be limited. Once the number of nodes undershoots a certain threshold, we are confronted with a higher probability of data loss. This observation entails that, once the number of nodes reaches a specific upper bound, this clique has to be split into two cliques. Likewise, if the number of nodes reaches a certain lower bound, the remaining nodes in the clique have to join another clique, thus the two cliques have to merge. Hence it follows that, apart from the standard operations, such as JOIN and LOOKUP, two additional operations, namely MERGE and SPLIT, are essential in eQuus.

We will first present the link structure that guarantees connectivity and fast lookups. Subsequently, the JOIN procedure is specified, followed by a detailed description of the MERGE and SPLIT procedure. In the last part of this section, the LOOKUP procedure is presented.

Note that there is no need for a specific LEAVE protocol. After a node could not be contacted by any clique member for a certain period of time, it is simply excluded from the clique and thus from the system.

3.1 Link Structure

A certain amount of links between the nodes in the network has to be maintained and updated periodically, in order to establish a structured network in which lookups are fast, the permanent joining and leaving of nodes is handled efficiently and a high degree of resilience to random as well as correlated failures is guaranteed.

As mentioned in the previous section, each node v knows about all other members of its clique c. Apart from these links, each node v has links to nodes in other cliques in order to guarantee connectivity and fast lookups in the network.

The routing mechanism is basically a generalized form of routing on hypercubes, where b bits are corrected in each hop. The number of bits b that can be corrected in a single hop is denoted the *base* of the system. This procedure is very similar to the one described for instance in [13], the main difference being that in eQuus, each entry represents an entire clique and not a single node. Let N denote the total number of cliques in the system. In order to ensure permanent connectivity, a constant number of k nodes are known in each of the approximately $(2^b-1)\lceil\log_{2^b} N\rceil$ cliques in the routing table, thus in total $k(2^b-1)\lceil\log_{2^b} N\rceil$ links are stored in the routing table, for a given base b. The average lookup path requires about $\lceil\log_{2^b} n\rceil$ hops, as we will show in Section 3.4. It is checked periodically if the cliques in the routing table are still alive. Every time a node in a specific clique is contacted, it returns a fresh, random list of k live nodes in the corresponding clique and this list is stored, replacing the old list in the routing table. In case all k nodes of a certain clique stored in the routing table have failed since the last request, which is an improbable event, another node in the own clique can be contacted in order to get a fresh list of nodes in this clique. Because all nodes in a clique store links to the same cliques, however they do not share the same subset of nodes in this clique, it is very likely that there is another clique member storing the address of a node in the corresponding clique that is still alive.

Additional k links lead to nodes in the clique that is the predecessor in the ID space and another k links lead to nodes in the successor clique. Thus, the total number of links is bounded by $\mathcal{U}+k(2^b-1)\lceil\log_{2^b} N\rceil+2k=\mathcal{O}(\log n)$ links, where $\mathcal{U}$ denotes the upper bound on the number of nodes in a clique. Knowing both the predecessor clique and the successor clique is necessary for the MERGE and also the SPLIT operation. We say that a clique c, or any node in clique c, is *responsible* for a data item, if its key is in the range $[c.id, c.successor.id)$, where $c.id$ and $c.successor.id$ denote the ID of the clique c and the ID of the successor clique of c, respectively. In the following, let $\mathcal{N}_v$ denote the neighborhood of node v consisting of all cliques in its routing table.

3.2 JOIN Procedure

In order to ensure that all nodes belonging to the same clique are close to each other and also that good locality properties such as a low stretch are maintained, a newly arriving node must join the closest clique in the system.

How a given node determines the closest clique is straightforward. The overall procedure is similar to the mechanism described in [17]. In the first step, a node contacts an arbitrary bootstrap node. The contacted node returns the address of one node of each clique in its routing table. The new node then contacts all those nodes and determines the closest one, e.g. by sending several ping messages and waiting for the replies. Subsequently, a JOIN request is sent to the closest node which again returns addresses of cliques in its routing table. This step is repeated until the closest clique has been found, which will happen in $\mathcal{O}(\log n)$ rounds. After sending a JOIN message to a node in the closest clique, this contacted node will inform all the other nodes in the clique about the arrival of a new node and give the new node all the information it needs to become a fully integrated clique member, which includes the ID of the clique, the addresses of all other clique members, the routing table and also the necessary information about all data items whose key lies in this clique's fraction of the ID space. The routing table can be copied from any other clique member, since they all share links to the same cliques. The total message complexity is obviously $\mathcal{O}(\log^2 n)$, since $\mathcal{O}(\log n)$ messages have to be sent in each of the $\mathcal{O}(\log n)$ rounds.

The first clique in the system bears the ID 0^d. As soon as it contains $\mathcal{U}$ nodes, it is split into two cliques, each obtaining half of the nodes. One clique keeps the ID 0^d while the other clique gets the new ID 10^{d-1}. The new clique with ID 10^{d-1} is the predecessor clique and the successor clique of the clique with ID 0^d and vice versa. How cliques are split is treated in greater detail in the next section. The nodes with ID 0^d are then responsible for all data items whose key is in the range $[0^d, 01^{d-1}]$ and the nodes with ID 10^{d-1} are responsible for the keys in the range $[10^{d-1}, 1^d]$ in the ID space. A newly arriving node always joins the closest clique. This clique is split again once its size reaches $\mathcal{U}$ and the new clique gets the ID in the middle between the ID of the current

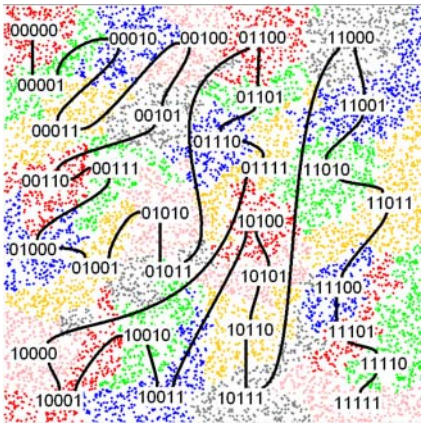

Figure 2. This figure visualizes the different prefix areas (nodes sharing a 5-bit prefix have the same color) after the joining of 10,000 nodes. We have additionally connected the prefix areas in increasing binary order, demonstrating that the prefix areas of the same order are indeed close to each other ("space filling curve").

clique and the successor clique etc.

Nodes within a clique have to communicate regularly in order to keep their routing tables fairly consistent. This replication does not incur a large overhead for several reasons. As mentioned before, nodes need to have links to the same cliques, but not to the same set of nodes within those cliques, thus no communication is required until a clique in the routing table is split or merged. Likewise, the index tables have to be consistent, but the information about which nodes in the network actually possess a copy of the indexed data items only has to be loosely synchronized, as it is sufficient if each node knows a subset of these nodes. The information about newly arriving and leaving nodes can be broadcast quickly within the clique, thus it is also fairly cheap to maintain consensus about the current set of nodes belonging to the clique. The only resource- and time-consuming operations are MERGE and SPLIT, which are rather infrequent, as we will show in Section 4.2. Thus, a high degree of redundancy is introduced at the cost of a modest message and space overhead. Furthermore, since all members of a clique are likely to be close to each other according to the chosen proximity metric and since links to cliques far away have to be checked only occasionally, a high percentage of the generated traffic is over short distances.

Fig. 2 depicts an example network after 10,000 joins. In this figure, the areas are highlighted where peers of certain prefixes (up to 5 bits) are located. Additionally, we have manually connected the areas of (binary) increasing prefixes. This "space filling curve" indicates that peers of prefixes of the same order are indeed close to each other.

3.3 MERGE and SPLIT Procedure

In case a node joins or leaves a clique, it has to be verified that the number of nodes in the clique is still in the range $[\mathcal{L},\mathcal{U}]$, where $\mathcal{L}$ and $\mathcal{U}$ denote the lower and upper bound on the number of nodes in a clique, respectively. It is important to set both $\mathcal{L}$ and $\mathcal{U}$ to reasonable values, such that enough redundancy is introduced and data loss can be avoided, but the routing tables do not become too large. As the synchronization between clique members does not impose a substantial burden on the system, it is expedient to set $\mathcal{L}$ and $\mathcal{U}$ to values that are at least logarithmic in the number of nodes. Thus, we can choose constants l_1, l_2 and u_1, u_2 such that $\mathcal{L} := l_1 d + l_2$ and $\mathcal{U} := u_1 d + u_2$, since $d = \Omega(\log n)$. For simplicity, in the remainder of the paper, we set $\mathcal{L} := \frac{d}{2} + 1$ and $\mathcal{U} := 2d - 1$. When setting the constants to other values of the same magnitude, all theorems are still correct up to small constant factors.

Once the size of a clique reaches $\mathcal{L} - 1$, it has to merge with the preceding clique in the ID space. For that purpose, the preceding clique is informed and the two cliques are combined into one, by adjusting the predecessor of the merging clique and the successor of the preceding clique. Furthermore, all nodes in the merging clique have to adapt the ID and the routing table from the members of their new clique. Before merging, both cliques were responsible for a particular fraction of the ID space. The new clique consisting of both cliques is then responsible for the combination of the two adjacent fractions of the ID space. It is therefore necessary to transfer the corresponding information between members of the two cliques. In addition, the successor clique of the merging node has to be informed about the topology change.

In case the size of a clique exceeds $\mathcal{U}$, the SPLIT procedure is initiated in which half of the nodes form a new clique. The nodes in the old and the newly created clique are then responsible for approximately half of the data items they were responsible before. In order to ensure a certain locality, the nodes closer to the preceding clique in the ID space keep their ID while the others get the new, higher ID, which is the ID between the current ID and the ID of the successor clique. Nodes store information about the average delay to nodes in the predecessor clique for this purpose. The $\frac{\mathcal{U}}{2}$ nodes with the lowest average distance to nodes in the predecessor clique keep their ID. However, if there is only one clique in the system, the node with the highest average distance to the other nodes is determined. This node and the $\frac{\mathcal{U}}{2} - 1$ nodes closest to itself keep the old ID and the remaining nodes form the new clique. Note that no data has to be copied between members of the two newly created cliques. After the splitting procedure is completed, both cliques can simply remove all data items associated with the fraction of the ID space that they are no longer responsible for.

It is possible that after the merging of two cliques, they are split again shortly thereafter. This is the case if the two cliques combined contain more than $\mathcal{U}$ nodes. By merging two cliques and splitting them again, as opposed to merely moving nodes from the clique with fewer nodes to the larger clique, the ID space is continuously partitioned locally in an optimal manner, because the new clique always gets the ID between its old ID and the ID of its successor clique. This subsequent splitting has to be delayed for a short period of

time, as the new nodes in the overcrowded clique first have to gather information about the distance to their new predecessor clique. This short delay is not crucial, as the MERGE and SPLIT operations have to be performed rarely. Even if by chance several cliques collapsed almost instantly, splitting the resulting clique would not be particularly harder than splitting any other clique and would merely entail more network traffic at the predecessor clique until the large clique is split.

The nodes in the new clique have to build a new routing table; however, they can very likely keep the higher entries in the old table. It is easy to see that the length of the prefix shared with the old ID determines the number of routing table entries that can be kept. While many entries can be kept immediately after the splitting, it is appropriate to update them continuously according to the following rule. The remainder of the ID following a given prefix is called *suffix*. For any relevant prefix, the clique whose ID has the longest suffix shared with the own ID following this specific prefix has to be stored in the routing table. When refreshing the routing table entries, the contacted nodes verify whether there is a clique in its routing table whose ID has the same prefix, but whose suffix is longer. If there is such a clique, this information is returned and the corresponding routing table entry is replaced in this clique. The goal of this scheme is to bound the number of cliques storing links to any particular clique. For example, the clique whose ID starts with 1011 prefers the clique whose ID starts with 0010 to a clique whose ID has the prefix 0000 for the prefix 00.

As far as the entries that cannot be kept are concerned, the following strategy allows for a quick replenishment. For each invalid entry in the routing table, a node in the corresponding clique is asked for a suitable replacement, i.e. a clique whose ID has the desired prefix for this entry and the new ID. The contacted node is much closer to the desired clique and can thus very likely answer this query. If it cannot, it can forward the request to an even closer clique etc., until a suitable clique is found. This operation is basically a regular lookup, which will be treated next.

3.4 LOOKUP Procedure

The main operation that the network has to be able to perform is lookup a clique, given a specific key $s \in \{0,1\}^d$. We define the metric $\delta : \{0,1\}^d \times \{0,1\}^d \rightarrow \{0,\ldots,d\}$ as follows. For any bit two strings s, s' of length d, $\delta(s,s') = i \iff s[d,i+1] = s'[d,i+1] \wedge s[i] \neq s'[i]$, where $s[i,j], i > j$, denotes the substring of s in the range $[i,j]$. Thus, $\delta(s,s')$ is the position of the highest order bit at which the two bit strings s and s' differ. This metric allows for a simple specification of the lookup protocol. It is further useful when it comes to analyzing the locality properties of eQuus.

The following simple algorithm performs this task independent of the chosen base, see Algorithm 1. The algorithm is similar to the lookup procedure described in [13]. The main difference is that our algorithm exploits the knowledge of the predecessor and the successor clique. It is first checked if node v in clique c is responsible for the key s, by testing if s lies between its ID and the ID of the successor clique. If this is not the case, it will forward the request to a node in a clique with a longer matching prefix. If there is no such node in the routing table, two cases have to be considered. If $s > v.id$, the request is forwarded to the clique with the largest ID in $\mathcal{N}_v$, subject to the constraint that the length of the matching prefix is not reduced. Otherwise, i.e. $v.id > s$, the request is simply forwarded to a node in the predecessor clique.

Algorithm 1 LOOKUP (Code for node v in clique c)

```
1: {u started LOOKUP request for key s}
2: if s ∈ [c.id, c.successor.id) then
3:    send(LOOKUP_DONE,c.id) to u;
4: else
5:    c̃ = arg min_{c'∈N_v} δ(c'.id, s);
6:    if not δ(c̃.id, s) < δ(c.id, s) then
7:       if s > c.id then
8:          c̃ = arg max_{c'∈N_v:δ(c'.id,s)=δ(c.id,s)} c'.id;
9:       else
10:         c̃ = c.predecessor;
11:      end if
12:   end if
13:   w =getNodeFromClique(c̃);
14:   send(LOOKUP,u, s) to w;
15: end if
```

The following theorems summarize the properties of the LOOKUP algorithm. In this analysis, it is assumed that all routing tables are accurate, i.e. each node knows about all other cliques that are relevant for its routing table. Note that this does not mean that all lists of k nodes for each clique must be up to date. Merely the information about the existence of cliques must be accurate and at least one of the k nodes of each clique stored in the routing table must be still alive.

Theorem 3.1 *The LOOKUP algorithm is correct, i.e. it always finds the clique responsible for any key, if the routing table entries are accurate.*

PROOF. Let the start node be v in clique c. It tries to find the clique responsible for the key s. If the key is in its own fraction of the ID space, i.e. $s \in [c.id, c.successor.id)$, then the LOOKUP procedure terminates correctly. Otherwise, it checks its routing table for a clique whose ID has a longer matching prefix. If such a clique exists, then the request will be forwarded to this clique. The new clique is closer to the responsible clique, since the matching prefix is at least one bit longer.

If there is no such clique in $\mathcal{N}_v$, two cases have to be distinguished. If $s > c.id$, then we have to forward the request to a clique with a larger ID. Let $i = \delta(c.id, s)$, thus $c.id[i] = 0$, $s[i] = 1$ and $c.id[j] = s[j]$ for all $i < j \leq d$. We assume that routing tables are accurate, thus there is no clique $\tilde{c}$ in the network whose ID is larger than the ID of c, $\delta(\tilde{c}, s) = i$ and $\tilde{c}.id[i] = 1$, otherwise v would know about it. Thus, it is best to forward the request to the clique with the largest ID among all cliques $\tilde{c}$ in $\mathcal{N}_v$ for which it holds that $\delta(\tilde{c}, s) = i$. There is always at least one clique satisfying this constraint. Assuming no such clique exists in the routing table, then it follows that $\delta(c.successor.id, s) > i$ and therefore $c.successor.id > s$, in contradiction to the assumption that $s \notin [c.id, c.successor.id)$.

Similarly, if $c.id > s$, it holds that $c.id[i] = 1$ and $s[i] = 0$. If there is no clique $\tilde{c}$ such that $\delta(\tilde{c}.id, s) < \delta(c.id, s)$, then it holds for all cliques with a lower ID than c that their identifiers are lower than s. Naturally, this also holds for the predecessor of c and thus the lookup terminates at the predecessor, because $c.predecessor.id < s < c.id$.

Therefore, requests are routed closer and closer to the destination in every step, which concludes the proof. □

It is also essential to bound the number of hops required to reach the target clique. The following theorem states that, w.h.p.[4], the number of hops does not exceed $\lceil \log_{2^b} n \rceil$ plus a small constant.

Theorem 3.2 *If all n nodes are uniformly distributed, then a LOOKUP terminates successfully after at most $\lceil \log_{2^b} n \rceil + o(1)$ hops w.h.p., if the routing table entries are accurate.*

PROOF. In the first step, we will show that the number of hops is upper bounded by the number of bits that are needed to uniquely identify all cliques.

Let node v in clique c be the node initiating the LOOKUP call for key s and let $\delta(c.id, s) = i$. Note that if there is no clique $\tilde{c}$ such that $\delta(\tilde{c}.id, s) < \delta(c.id, s)$, we cannot correct the i^{th} bit, thus we cannot argue that the distance measured with the metric δ decreases in each step. However, we can argue that the number of bits that still have to be considered is decreasing with each hop. Apparently, if there is a clique $\tilde{c}$ such that $\delta(\tilde{c}.id, s) < \delta(c.id, s)$, forwarding the request to this clique will fix at least the i^{th} bit and all bits in the range $[i, d]$ do not have to be considered anymore, thus the search space will be reduced by at least one bit. If we are in the situation that there is no such clique, then we know that the i^{th} bit does no longer have to be corrected, since there is no clique that could fix it. Hence, the search space is reduced by at least one bit in each step.

Similarly, for general bases b, each step reduces the search space by at least b bits, if all cliques in the routing table are still alive.

Now we will show that $\lceil \log n + 4 \rceil$ bits suffice with high probability to uniquely identify all cliques. Let the ball $B_v(\alpha)$ denote the set of nodes around node v at a distance of at most α. For all $x \in [0, 1]$, it holds that $p(|B_v(x\Delta)| \geq 2d) = \sum_{i=2d}^{n} \binom{n}{i} (x^2\pi)^i (1 - x^2\pi)^{n-i} < \binom{n}{2d} (x^2\pi)^{2d} < \left(\frac{ne}{2d}\right)^{2d} (x^2\pi)^{2d}$, where Δ denotes the diameter of the network. Let the random variable X denote the event that there is a node v such that $|B_v(x\Delta)| \geq 2d$. Setting $x := \sqrt{\frac{2(d - \ln n)}{ne\pi}}$, it follows that $p(X) \leq \sum_{v \in V} p(|B_v(x\Delta)| \geq 2d) < \left(\frac{nex^2\pi}{2d}\right)^{2d} n = \left(\frac{ne2(d-\ln n)\pi}{ne\pi 2d}\right)^{2d} n = \left(1 - \frac{2\ln n}{2d}\right)^{2d} n < e^{-2\ln n} n < \frac{1}{n}$.

Thus, with high probability, there are less than $2d$ nodes at a distance of at most $x\Delta$ for each node. All cliques and consequently all IDs are spread in a two dimensional Euclidean space. Whenever a clique splits into two, both cliques occupy half of the original space and the diameter of each of those two subspaces is only a factor of $\frac{1}{\sqrt{2}}$ of the original diameter in expectation. It has to be determined how often the Euclidean space must be partitioned until the diameter of each subspace is at most $x\Delta$. If this is the case, there is no subspace that contains $2d$ or more nodes and thus no subspace has to be divided any further, with high probability. The Euclidean space must be partitioned i times so that $\frac{1}{\sqrt{2^i}}\Delta \leq x\Delta$. This holds if $i := \log n + 4$, and thus $\log n + 4$ bits suffice, since each splitting costs one bit.

Hence, with high probability, the number of hops is at most $\lceil \frac{\log n + 4}{b} \rceil$, for a specific base b. □

While the number of hops is already small with high probability, even fewer hops are required on average. According to simulations, the average number of hops is lower than $\lceil \log_{2^b} n \rceil$, if nodes are uniformly distributed. Fig. 3 depicts the average number of hops required in a network of dimension $d = 64$, consisting of up to one million nodes. 10,000 lookups initiated at a random node and searching for a random key s have been performed for base $b = 1, 2$, and 4. As long as the number of nodes is lower than $2d = 128$, the entire network is a complete graph in which each node is responsible for the entire ID space, therefore the number of hops is always 0. The main reason why the average number of hops is less than $\lceil \log_{2^b} n \rceil$ is because there are only $\Theta(\frac{n}{d})$ cliques.

It is worth noting that the distribution of the number of hops required varies remarkably, depending on the base b. While the distribution resembles a normal distribution for $b = 1$, the distribution for $b = 4$ is one-sided and a large fraction of all lookups require about the average number of hops.

Since each node has k links to all cliques in its routing table, it is very likely—this probability depends on the choice of k—that the lookup will reach the destination, because the

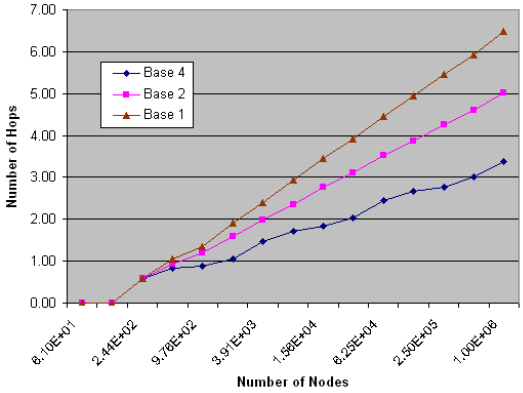

Figure 3. The average number of hops depending on the number of nodes in the system and the chosen base b is displayed. For each number of nodes n and for each base b, 10,000 lookups initiated at a random node and searching for a random key have been performed. The dimension d of the network is 64 in each run.

[4]By "with high probability," or short "w.h.p.," we mean with probability at least $1 - \frac{1}{n}$, where n is the number of nodes in the system.

request can be forwarded to another node, in case one node on the path does no longer respond.

So far, we have assumed that all routing entries contain correct information about the state of the network, which is—even though changes, i.e. MERGE and SPLIT operations can be performed fast and do not occur often locally—overly optimistic.

For example, it is possible that a clique has just been split and the i^{th} bit could have been corrected directly, had the node forwarding the request already learnt about it, resulting in an additional hop. The LOOKUP algorithm, however, is still correct. More accurate lookups can be achieved by asking for an update in case the request cannot be forwarded to a clique that reduces the distance according to the metric δ. The drawback of this approach is that more messages have to be exchanged and, as a result, searching takes longer. Since this situation rarely occurs and the LOOKUP algorithm behaves correctly in each scenario, this matter is not further investigated.

4 Fault Tolerance

Replicating data by creating cliques of nodes that all cover the same portion of the ID space ensures a certain degree of robustness by itself. Even in case of a correlated failure of $\mathcal{L}-1$ nodes, there is at least one node left that can merge with the previous clique, because each clique consists of more than $\mathcal{L}-1$ nodes. Ensuring that no data is ever lost is one of the main objectives of any fault-tolerant system. In the first part of this section, we will show that the probability of data loss is very low, even if communication in the entire network failed for a relatively long period of time.

Apart from preventing data loss, it is essential to keep up the network structure, even in the presence of churn. In order to quantify the resilience of eQuus to churn, it is desirable to derive bounds on the induced message overhead, depending on the number of JOIN and LEAVE events and the size of the network. This is the subject of the second part of this section.

4.1 Communication Failures

Under normal operation, each node refreshes its routing table by regularly requesting new lists of k live nodes from each clique stored in its routing table. By performing this update frequently, the probability that a considerable fraction of the nodes whose addresses are stored in the routing table are no longer alive is negligible.

In fact, the probability that any data is lost is even very low in case all communication ceases for a long period of time. Let $\lambda(p)$ be the period in which each node disappears with probability p. For instance, in a network of dimension $d = 64$ consisting of one million nodes, the probability that no data is lost, if no communication occurs in a period of $\lambda(\frac{1}{2})$ is higher than 0.99999. This holds because an entire clique has to fail before being able to merge and the probability that any clique fails is less than $p^{\frac{d}{2}}$. The total number of cliques is lower than $\frac{2n}{d}$ and thus the probability that no data is lost is higher than $(1-p^{\frac{d}{2}})^{\frac{2n}{d}}$.

If the life time of nodes and the total number of nodes that will ever participate simultaneously can be estimated, the probability of data loss will be arbitrarily small by setting the update frequency to an appropriate value.

4.2 Dealing with Churn

eQuus effectively deals with churn, by reducing the number of topology changes that affect nodes in different cliques. Whenever a node joins or leaves the network, communication is primarily needed between members of the corresponding clique and no other routing table update is required due to this event. Changes in the routing tables of various cliques are only required if a clique either splits or merges with its predecessor. Hence, in order to evaluate the resistance to churn, it suffices to show that it takes a large number of join and leave events globally, before any clique either has to split or merge.

Let N denote the number of cliques in the network and let m denote the number of JOIN/LEAVE events. The probability that the next event occurs at any given clique is $\frac{1}{N}$, according to our uniform distribution model. It is essential to estimate the expected maximum number of JOIN and LEAVE occurring at any clique.

Lemma 4.1 *If the network consists of N cliques and m JOIN/LEAVE events occur, the expected maximum number of JOIN/LEAVE events on any clique is bounded by $\mathcal{O}(\frac{m}{N} + \log N)$.*

PROOF. Let the random variable $X_N^i(m)$ denote the number of events occurring at clique i in a network consisting of N cliques in which m JOIN/LEAVE events occur. We want to derive a bound for $E[\max_{1\le i\le N} X_N^i(m)]$. It holds that $E[2^{\max_{1\le i\le N} X_N^i(m)}] \le E[\sum_{i=1}^{N} 2^{X_N^i(m)}] = N \cdot (\sum_{j=0}^{m} 2^j \binom{m}{j} (\frac{1}{N})^j (1-\frac{1}{N})^{m-j}) \le N \cdot (1 + \sum_{j=1}^{m} (\frac{2me}{jN})^j e^{-\frac{m-j}{N}}) \le N \cdot (1 + \sum_{j=1}^{m} e^{-(1-\frac{2me}{jN})j} e^{-\frac{m-j}{N}}) \le N \cdot (e^{\frac{(2e-1)m}{N}} \sum_{j=0}^{m} e^{(\frac{1}{N}-1)j}) = \mathcal{O}(N2^{\mathcal{O}(\frac{m}{N})})$. Hence it follows that $E[\max_{1\le i\le N} X_N^i(m)] \le \log(E[2^{\max_{1\le i\le N} X_N^i(m)}]) = \mathcal{O}(\frac{m}{N} + \log N)$. □

Now we are in the position to derive a lower bound for the expected number of global events before any clique has to either merge or split, depending on the number of nodes currently in the system.

Theorem 4.1 *If all n nodes are uniformly distributed, then $\Omega(n)$ JOIN/LEAVE events are required in expectation before either a MERGE or a SPLIT operation has to be performed.*

PROOF. In expectation, $\Theta(d)$ events are required before any clique has to either merge or split. This holds, since the expected number of nodes in a clique is $\Theta(d)$ and thus $\Theta(d)$ nodes have to either join or leave, independent of the probabilities of those events, before a MERGE or SPLIT operation has to be performed. By Lemma 4.1, it follows that $m = \Omega(Nd - N\log N)$. Since $N = \Theta(\frac{n}{d})$ and $d = \Omega(\log n)$, it holds that $m = \Omega(n - \frac{n}{d}\log n + \frac{n}{d}\log d) = \Omega(n)$. □

This theorem shows that, in expectation, a large number of JOIN and LEAVE events is required before any relevant topology change occurs. Consequently, in large networks, the permanent joining and leaving can be handled efficiently. Due to the rare occurrence of those relatively costly operations, it is simple for the network to update the routing tables in due time and thus maintain the desired structure.

5 Locality

In the first part of this section, the locality properties of eQuus are analyzed formally in the uniform distribution model. The second part presents results of the system in the same model obtained by simulation. The goal of the formal analysis is to derive upper bounds on the expected total path lengths and the expected stretch, while the second part presents results of several simulations run in order to determine the locality properties in an emulated environment.

5.1 Formal Analysis

Some other DHTs do not consider the problem of having potentially long lookup paths. In expectation, each hop incurs a delay of $\frac{\Delta}{2}$, where Δ denotes the network diameter, thus the expected path length of a path consisting of h hops is $\frac{h}{2}\Delta$. If h is large, these paths can become very long, although the destination node might be very close to the node initiating the lookup call. The goal is to guarantee that all paths are only a small factor longer than the direct paths. Unfortunately, our JOIN procedure does not yield a good worst-case bound on the stretch. In order to estimate the quality of our mechanisms, it is appropriate to analyze the average case behavior.

In this section, we assume an *Euclidean metric space*, a special case of the well-known *doubling* (e.g. [3]) and *growth bounded* (e.g. [5]) metric spaces. Concretely, we assume that all nodes lie in a two dimensional plane and have Euclidean distances to each other, i.e. $c(u,v) = ||u-v||_2$. This assumption is obviously not an optimal approximation for large networks such as the Internet. However, if the delay is used as the proximity metric, then our assumption is reasonable, since there is a correlation between distance and delay [19].

Let $\Delta(\mathcal{S})$ denote the diameter of the network consisting of all nodes in the set $\mathcal{S}$. If node v in clique c performs a lookup for key s, where $\delta(c.id, s) = i$, only the subset $\mathcal{B}_v^i$ of all nodes u in any clique $\tilde{c}$ for which it holds that $\delta(\tilde{c}.id, s) \leq i$ have to be considered, due to the property of the LOOKUP procedure that the length of the shared prefix can only increase with each hop. More formally, let V be the set of all nodes in the network, then $\mathcal{B}_v^i := \{u \in V \mid u \in \tilde{c} \wedge \delta(\tilde{c}.id, s) \leq i\}$.

The following lemmas are used in order to establish our main result. The first lemma states that the expected diameter of the network consisting of all nodes in $\mathcal{B}_v^i$ is small if and only if i is small, i.e. the key s and the ID of clique c that node v belongs to share a long prefix.

Lemma 5.1 *Let node v in clique c be the node initiating the lookup call for key s. If $\delta(c.id, s) = i$, then it holds that $E[\Delta(\mathcal{B}_v^i)] \leq \frac{\Delta}{(\sqrt{2})^{d-i}}$.*

PROOF. After a clique split into two, the area both cliques are responsible for is only half of the original area in expectation. A clique is responsible for a certain area if newly arriving nodes in this area send a JOIN message to a node in this clique. Since the area is halved in expectation, it holds that the diameter is a factor of $\sqrt{2}$ shorter in expectation. Therefore it holds that $E[\Delta(\mathcal{B}_v^{i-1})] = \frac{1}{\sqrt{2}} E[\Delta(\mathcal{B}_v^i)]$, and due to the fact that $\Delta(\mathcal{B}_v^i) \leq \Delta$ for all $v \in V$, the claim follows. □

For the keys are random, any node in $\mathcal{B}_v^i$ has an equal chance to be the destination node of the lookup. Due to the uniform distribution of all nodes, the expected distance to the destination node is half of $\Delta(\mathcal{B}_v^i)$.

Lemma 5.2 *Let v in clique c be the node initiating the lookup call for key s and let u be the destination node. If $\delta(c.id, s) = i$, then the expected distance between v and u is $\frac{\Delta(\mathcal{B}_v^i)}{2}$.*

These lemmas suffice to prove the following upper bound on the expected stretch.

Theorem 5.1 *The expected stretch of a lookup call in eQuus is at most $\frac{2^{\frac{b}{2}+1}}{2^{\frac{b}{2}}-1}$ for a particular base b.*

PROOF. Let node v in clique c be any node initiating a lookup call for key s, let u be the destination node of the lookup and let $\delta(c.id, s) = i$. Lemma 5.2 states that $E[c(v,u) \mid d(c.id, s) = i] = \frac{\Delta(\mathcal{B}_v^i)}{2}$. This is true independent of the base b.

Since b bits are corrected with the first hop to node w in clique $\tilde{c}$, it holds that $\delta(c.id, \tilde{c}.id) \leq i - \min\{b, i\}$ and, according to Lemma 5.1, the expected diameter of $\mathcal{B}_w^{\delta(\tilde{c}.id,s)} \subseteq \mathcal{B}_v^i$ is bounded by $2^{-\frac{b}{2}}\Delta(\mathcal{B}_v^i)$.

Inductively, the total path length is therefore at most $\frac{1}{1-2^{-\frac{b}{2}}}\Delta(\mathcal{B}_v^i)$. The expected stretch is upper bounded by the ratio between the expected maximum total path length and the expected distance to the destination. Hence, it holds that $E[Stretch] \leq \frac{\frac{1}{1-2^{-\frac{b}{2}}}\Delta(\mathcal{B}_v^i)}{\frac{\Delta(\mathcal{B}_v^i)}{2}} = \frac{2^{\frac{b}{2}+1}}{2^{\frac{b}{2}}-1}$. □

Note that the expected total path length between any two nodes is at most $\frac{2^{\frac{b}{2}}}{2^{\frac{b}{2}}-1}\Delta$ in expectation, independent of the dimension d and the number of hops. Setting b to a moderately large value will incur an expected stretch of around 2. If $b = 4$, the expected stretch is already less than 3, which is already a satisfactory result. In the following section, this result is supplemented and affirmed by simulation.

5.2 Simulation

Various simulations have been run in order to study the locality properties of the system. In particular, the expected stretch has been analyzed. In Fig. 4, the average stretch of lookups in networks of dimension $d = 64$ with up to one million nodes are depicted for the bases $b = 1, 2$ and 4.

10,000 lookups have been performed for each network size and base. Obviously, the stretch factor is 1 as long as each node has links to nodes in all other cliques. The stretch slowly increases, as the networks grows because more hops are needed in order to reach a node in the desired clique. However, the stretch only grows as long as it is below the constant expected stretch for the given base, a bound that is seemingly reached at around 1.5 for base $b = 4$. This is a much better result than the upper bound of $\frac{8}{3}$ on the expected stretch for $b = 4$ derived in the previous section.

In the second simulation, the effect of the dimension d on the stretch has been tested for $d = 32, 64$ and 128. As

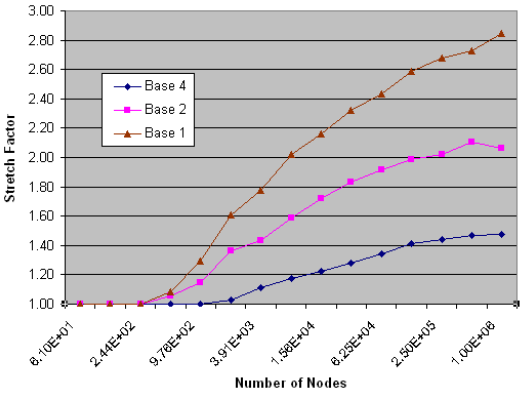

Figure 4. The average stretch depending on the number of nodes in the system and the chosen base b is displayed. For each number of nodes n and for each base b, 10,000 lookups initiated at a random node and searching for a random key have been performed. The dimension d of the network is 64 in each run.

expected, the stretch does not depend on the dimension directly. Choosing a large dimension will result in a slightly lower stretch, mainly due to the lower number of cliques in the network.

The results of our simulations show that the system has good locality properties, such as a low expected stretch and a low expected total path length. They further indicate that the analytical results are conservative, since the simulations yield better results.

6 Conclusion

Many existing p2p systems have some form of fault-tolerance or even locality awareness. The main contribution of eQuus is that it combines existing and novel building blocks in a way which yields a system of highly desirable properties, such as inherently strong resilience to churn, or low expected stretch. Another strength of eQuus is that the maintenance overhead is low, although all data items are replicated among several peers, as synchronization is only required on cluster level. Furthermore, most communication triggered by the maintenance protocols is local in nature, thus maintenance operations can be performed quickly and not much wide-area traffic is induced.

Spurred by the promising results, some of which have been presented in this paper, we have started to implement the eQuus distributed hash table.

Of course, when building applications on top of eQuus, many challenges orthogonal to the DHT design will arise. For example, in order to be robust against massive geographically correlated failures, one could think of replicating data in *two* clusters which are located diametrical to each other in the identifier space (e.g., the second cluster is given by the bit-inverse identifier of the first cluster).

Acknowledgements

We would like to thank Thomas Moscibroda for many fruitful discussions and the anonymous reviewers for their valuable feedback. This research has been supported by the Swiss National Science Foundation and the Hasler Stiftung.

References

[1] I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron. Practical Locality-Awareness for Large Scale Information Sharing. In *Proc. 4th IPTPS*, 2005.

[2] I. Abraham, D. Malkhi, and O. Dobzinski. LAND: Stretch (1 + ε) Locality-Aware Networks for DHTs. In *Proc. 15th SODA*, pages 550–559, 2004.

[3] H. T.-H. Chan, A. Gupta, B. M. Maggs, and S. Zhou. On Hierarchical Routing in Doubling Metrics. In *Proc. 17th SODA*, pages 762–771, 2005.

[4] A. Fiat and J. Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proc. 13th SODA*, 2002.

[5] D. R. Karger and M. Ruhl. Finding Nearest Neighbors in Growth-Restricted Metrics. In *Proc. 34th STOC*, pages 741–750, 2002.

[6] F. Kuhn, S. Schmid, and R. Wattenhofer. A Self-Repairing Peer-to-Peer System Resilient to Dynamic Adversarial Churn. In *Proc. 4th IPTPS*, 2005.

[7] X. Li, J. Misra, and C. G. Plaxton. Active and Concurrent Topology Maintenance. In *Proc. 18th Ann. Conference on Distributed Computing (DISC)*, 2004.

[8] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proc. 1st IPTPS*, 2002.

[9] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[10] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proc. 9th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.

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

[12] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proc. USENIX Ann. Technical Conference*, 2004.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. 18th IFIP/ACM Int. Conference on Distributed Systems Platforms (MiddlewareS)*, 2001.

[14] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. In *Proc. 1st IPTPS*, 2002.

[15] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement study of Peer-to-Peer File Sharing Systems. In *Proc. SPIE Multimedia Computing and Networking (MMCN)*, 2002.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM Conference*, 2001.

[17] M. Waldvogel and R. Rinaldi. Efficient Topology-Aware Overlay Network. In *Proc. HotNets-I*, Princeton, NJ, USA, 2002.

[18] B. Y. Zhao, L. Huang, J. Stribling, 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), 2004.

[19] A. Ziviani, S. Fdida, J. F. de Rezende, and O. C. M. B. Duarte. Toward a measurement-based geographic location service. In *Proc. of the Passive and Active Measurement Workshop (PAM)*.