Hierarchical Peer-to-peer Systems

L. Garcés-Erice[1], E.W. Biersack[1], P.A. Felber[1],
K.W. Ross[2], and G. Urvoy-Keller[1]

[1] Institut EURECOM
06904 Sophia Antipolis, France
{garces|erbi|felber|urvoy}@eurecom.fr

[2] Polytechnic University
Brooklyn, NY 11201, USA
ross@poly.edu

Abstract. Structured peer-to-peer (P2P) lookup services—such as Chord, CAN, Pastry and Tapestry—organize peers into a flat overlay network and offer distributed hash table (DHT) functionality. In these systems, data is associated with keys and each peer is responsible for a subset of the keys. We study hierarchical DHTs, in which peers are organized into groups, and each group has its autonomous intra-group overlay network and lookup service. The groups themselves are organized in a top-level overlay network. To find a peer that is responsible for a key, the top-level overlay first determines the group responsible for the key; the responsible group then uses its intra-group overlay to determine the specific peer that is responsible for the key. After providing a general framework for hierarchical P2P lookup, we consider the specific case of a two-tier hierarchy that uses Chord for the top level. Our analysis shows that by designating the most reliable peers in the groups as superpeers, the hierarchical design can significantly reduce the expected number of hops in Chord. We also propose a scalable design for managing the groups and the superpeers.

1 Introduction

Peer-to-peer (P2P) systems are gaining increased popularity, as they make it possible to harness the computing power and resources of large populations of networked computers in a cost-effective manner. A central problem of P2P system is to assign and locate resources among peers. This task is achieved by a P2P *lookup service*.

Several important proposals have been recently put forth for implementing distributed P2P lookup services, including Chord [1], CAN [2], Pastry [3] and Tapestry [4]. In these lookup services, each key for a data item is assigned to the live peer whose node identifier is "closest" to the key (according to some metric). The lookup service essentially performs the basic function of determining the peer that is responsible for a given key. The lookup service is implemented by organizing the peers in a structured overlay network, and routing a message through the overlay to the responsible peer. The efficiency of a lookup service is generally measured as a function of the number of peer hops needed to route a message to the responsible peer, as well as the size of the routing table maintained by each peer. For example, Chord requires $O(\log N)$ peer hops and $O(\log N)$ routing table entries when there are N peers in the overlay. Implementations of the distributed lookup service are often referred to as Distributed Hash Tables (DHTs).

Distributed DHTs are the central components of a wide range of new distributed applications, including distributed persistent file storage [5, 6], Web caching [7], multicast [8, 9], or computational grids [10]. DHTs generally provide improvement to an application's resilience to faults and attacks.

Chord, CAN, Pastry and Tapestry are all flat DHT designs without hierarchical routing. Each peer is indistinguishable from another in the sense that all peers use the same rules for determining the routes for lookup messages. This approach is strikingly different from routing in the Internet, which uses hierarchical routing. Specifically, in the Internet, routers are grouped into autonomous systems (ASes). Within an AS, all routers run the same intra-AS routing protocol (e.g., RIP or OSPF). Special gateway routers in the various ASes run an inter-AS routing protocol (BGP) that determines the path among the ASes. Hierarchical routing in the Internet offers several benefits over non-hierarchical routing, including scalability and administrative autonomy (e.g., at the level of a university, a corporate campus, or even the coverage area of a base station in a mobile network).

In this paper we explore hierarchical DHTs. Inspired by hierarchical routing in the Internet, we examine two-tier DHTs in which (i) peers are organized in disjoint groups, and (ii) lookup messages are first routed to the destination group using an inter-group overlay, and then routed to the destination peer using an intra-group overlay. We will argue that hierarchical DHTs have a number of advantages, including:

- They significantly reduce the average number of peer hops in a lookup, particularly when nodes have heterogeneous availabilities.
- They significantly reduce the lookup latency when the peers in the same group are topologically close and cooperative caching is used within the groups.
- They facilitate the large-scale deployment of a P2P lookup service by providing administrative autonomy to participating organizations. In particular, in the hierarchical framework that we present, each participating organization (e.g., institutions and ISPs) can choose its own lookup protocol (e.g., Chord, CAN, Pastry, Tapestry).

We present a general framework for hierarchical DHTs. In the framework, each group maintains its own overlay network and uses its own intra-group lookup service. A top-level overlay is also defined among the groups. Within each group, a subset of peers are labeled as “superpeers”. Superpeers, which are analogous to gateway routers in hierarchical IP networks, are used by the top-level overlay to route messages among groups. We consider designs for which peers in the same group are locally close. We describe a cooperative caching scheme that can significantly reduce average data transfer delays. Finally, we also provide a scalable algorithm for assigning peers to groups, identifying superpeers, and maintaining the overlays.

After presenting the general framework, we explore in detail a particular instantiation in which Chord is used for the top-level overlay. Thus, in this instantiation, Chord is analogous to BGP in Internet routing, and the intra-group lookup services are analogous to intra-AS routing protocols. Using a novel analytical model, we analyze the expected number of peer hops that are required for a lookup in the hierarchical Chord instantiation. Our model explicitly captures inaccuracies in the routing tables due to peer failures.

The paper is organized as follows: We first discuss related work in Section 2. We then present the general framework for hierarchical DHT’s in Section 3. We discuss the particular case of a two-tier Chord instantiation in Section 4, and we quantify the improvement of lookup latency due to the hierarchical organization of the peers.

2 Related Work

P2P networks can be classified as being either unstructured or structured. Chord [1], CAN [2], Pastry [3], Tapestry [4], and P-Grid [11], which use highly structured overlays and use hashing for targeted data placement, are examples of structured P2P networks. These P2P networks are all flat designs (P-Grid is based on a virtual distributed search tree, but peer nodes are located at the leaves level and the tree is used solely for routing purposes). Gnutella [12] and KaZaA [13], whose overlays grow organically and use random data placement, are examples of unstructured P2P networks.

Ratnasamy et al. [14] explore using landmark nodes to bin peers into groups. The basic idea is for each peer to measure its round-trip time (RTT) to M landmarks, order the resulting RTTs, and then assign itself to one of $M!$ groups. The authors then apply this binning technique to CAN, to construct a locality-aware overlay. In this binning-CAN scheme, the node id space is partitioned into $M!$ equal-size portions, one portion corresponding to each group. When a peer wants to join the overlay, it pings the landmarks to determine the group, and hence the portion of the id space, to which it belongs; the peer then gets assigned a node id, uniformly chosen from that portion of the node id space. This implies that during a lookup, typically short topological hops are taken while the lookup message travels through a group; and then longer topological jumps are taken when the message reaches the boundary of a group. Our hierarchical DHT schemes bear little resemblance to the scheme in [14]. Although in [14] the peers are organized in groups according to locality, the lookup algorithm applies only to CAN, does not use superpeers, and is not a multi-level hierarchical algorithm.

Our approach has been influenced by KaZaA, an enormously successful unstructured P2P file sharing service. (Today, KaZaA has typically several million participating peers at the same time.) KaZaA designates the more available and powerful peers as *supernodes*. In KaZaA, when a new peer wants to join, it bins itself with the existing supernodes, and establishes an overlay connection with the supernode that has the shortest RTT. The supernodes are connected through a top-level overlay network, using a proprietary design. A similar architecture has been proposed in CAP [15], a two-tier unstructured P2P network that focuses on scalability and stability. Our design is a blend of the supernode/hierarchy/heterogeniety of KaZaA with the lookup services in the structured DHTs.

Brocade [16] proposes to organize the peers in a two-level overlay. All peers form a *single* overlay O_L. Geographically close peers are then grouped together and get assigned a representative called “supernode”. Supernodes are typically well connected and situated near network access points. The supernodes form another overlay O_H, and each of them must somehow announce which peers are reachable through him. Brocade is not truly hierarchical since *all* peers are part of O_L, which prevents it from reaping the benefits of hierarchically organized overlays discussed in section 3.1.

Finally, Castro et al. present in [17] a topology-aware version of Pastry [3]. At each hop Pastry presents multiple equivalent choices to route a request. By choosing the closest (smallest network delay) peer at each hop, they try to minimize network delay. However, at each step the possibilities decrease exponentially, so delay is mainly determined by the last hop, usually the longest. Our approach is somewhat the opposite, as we propose large hops to first get to a group, and then shorter local hops inside the group. Note that our architecture leads to a more natural caching scheme, as shown later in section 3.4.

3 Hierarchical Framework

We begin by presenting a general framework for a hierarchical DHT. Although we focus on a two-tier hierarchy, the framework can be extended to a general tier hierarchy in a straightforward manner.

Let P denote the set of peers participating in the system. Each peer has a node id. Each peer also has an IP address, which may change whenever it re-connects to the system. The peers are interconnected through a network of links and switching equipment (routers, bridges, etc.) The peers send lookup query messages to each other using a hierarchical overlay network, as described below.

The peers are organized into groups. We will discuss how groups are created and managed and how peers are assigned to groups in Section 3.3. The groups may or may not be such that the peers in the same group are topologically close to each other, depending on the application needs. Each group has a unique group id. Let I be the number of groups, G_i the peers in group i, and g_i the id for group i.

The groups are organized into a *top-level overlay network* defined by a directed graph (X, U), where $X = \{g_1, \ldots, g_I\}$ is the set of all the groups and U is a given set of virtual edges between the nodes (that is, groups) in X. The graph (X, U) is required to be connected, that is, between any two nodes g and g' in X there is a directed path from g to g' that uses the edges in U. It is important to note that this overlay network defines directed edges among groups and not among specific peers in the groups.

Each group is required to have one or more *superpeers*. Superpeers, as we will discuss below, have special characteristics and responsibilities. Let $S_i \subseteq G_i$ be the set of superpeers in group i. Our architecture allows for $S_i = G_i$ for all $i = 1, \ldots, I$, in which case all peers are superpeers (and distinction between regular peers and superpeers becomes superfluous). We refer to architectures for which all peers are superpeers as the *symmetric design*. Our architecture also allows $|S_i| = 1$ for all $i = 1, \ldots, I$, in which case each group has exactly one superpeer. Let $R_i = G_i - S_i$ be the set of all "regular peers" in group g_i. For non-symmetric designs ($S_i \neq G_i$), an attempt is made to designate the more powerful peers as superpeers. By "more powerful," we primarily mean the peers that are up and connected the most. But as secondary criteria, superpeers will be the peers that have high CPU power and/or network connection bandwidth.

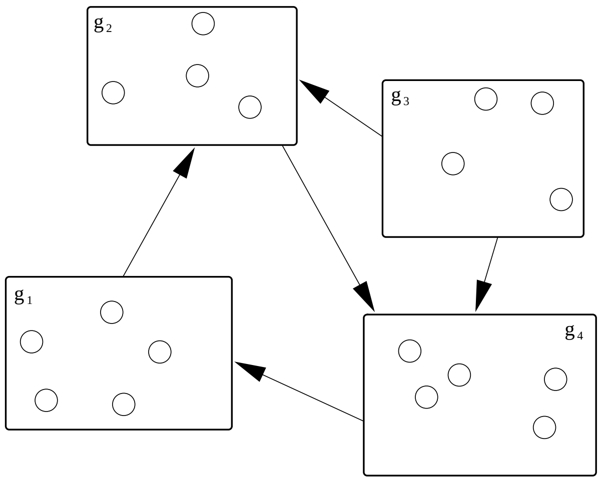

Fig. 1. Communication relationships between groups in the overlay network.

The superpeers are gateways between the groups: they are used for inter-group query propagation. To this end, we require that if s_i is a superpeer in G_i, and (g_i, g_j) is an edge in the top-level overlay network (X, U), then s_i knows the name and the current IP address of at least one superpeer $s_j \in S_j$. With this knowledge, s_i can send query messages to s_j. On the other hand, if p is a regular peer, then p must first send intra-group query messages to a superpeer in its group, which can then forward the query message to another group. Regular peers must thus know the name and IP address of the superpeers in their group. Figure 1 shows a top-level overlay network. Figure 2 shows possible communication relationships between the corresponding superpeers. Figure 3 shows an example for which there is one superpeer in each group and the top-level overlay network is a ring.

Within each group there is also an overlay network that is used for query communication among the peers in the group. Each of the groups operates autonomously from the other groups. For example, some groups can use Chord, others CAN, others Pastry, and yet others Tapestry.

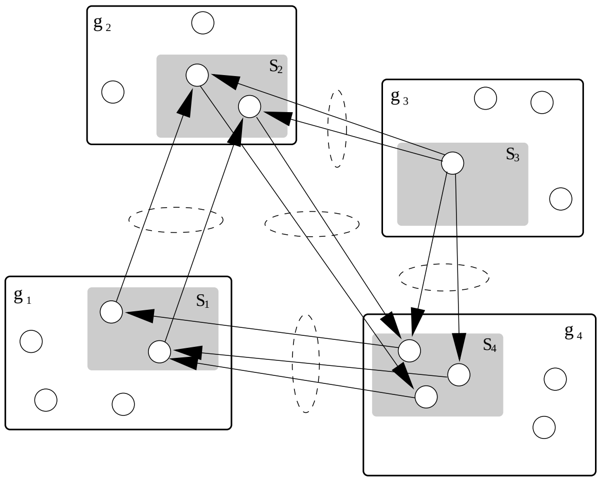

Fig. 2. Communication relationships between superpeers in neighboring groups.

3.1 Hierarchical Lookup Service

Let us first consider a two-level lookup service, where the top level manages peer groups and the bottom level peer nodes. Given a key k, we say that group g_j is responsible for k if g_j is the "closest" group to k among all the groups. Here "closest" is defined by the specific top-level lookup service (e.g., Chord, CAN, Pastry, or Tapestry).

The implementation of the lookup service exploits the hierarchical architecture: first, the lookup service finds the group that is responsible for the key; then it finds the peer within that group that is responsible for the key. Specifically, our two-tier DHT operates as follows. Suppose a peer $p_i \in G_i$ wants to determine the peer that is responsible for a key k.

1. Using the overlay network in group i, peer p_i sends a query message to one of the superpeers in S_i.

2. Once the query reaches a superpeer, the top-level lookup service routes the query through (X, U) to the group G_j that is responsible for the key k. During this phase, the query only passes through superpeers, hopping from one group to the next. A superpeer in one group uses its knowledge of the IP addresses of superpeers in the subsequent group along the route to forward the query message from group to group. Eventually, the query message arrives at some superpeer $s_j \in G_j$.
3. Using the overlay network in group j, the the superpeer s_j routes the query to the peer $p_j \in G_j$ that is responsible for the key k.
4. Peer p_j sends a response back to querying peer p_i. Depending on the design, this response message can follow the reverse path of the path taken by the query message, or can be sent directly from peer p_j to peer p_i (ignoring the overlay networks).

This approach can be generalized to an arbitrary number of levels. A request is first routed through the top-most overlay network to some superpeer at the next level below, which in turn routes the request through its "local" overlay network, and so on until the request finally reaches some peer node at the bottom-most level. In the rest of this paper, we will focus on the case of a two-level lookup service.

The hierarchical architecture has several important advantages when compared to the flat overlay networks.

– *Exploiting heterogeneous peers:* By designating as superpeers the peers that are "up" the most, the top-level overlay network will be more stable than the corresponding flat overlay network (for which there is no hierarchy). This increased stability enables the lookup service to approach its theoretical optimal lookup hop performance (for example, on average $\frac{1}{2} \log N$ for Chord, where N is the number of peers in the Chord overlay).
– *Transparency:* When a key is moved from one peer to another within a group, the search for the peer holding the key is completely transparent to the top-level algorithm. Similarly, if a group changes its intra-group lookup algorithm, the change is completely transparent to the other groups and to the top-level lookup algorithm. Also, the failure of a regular peer $r_i \in G_i$ (or the appearance of a new peer) will be *local* to G_i; routing tables in peers outside of G_i are not effected.
– *Faster lookup time:* Because the number of groups will be typically orders of magnitude smaller than the total number of peers, queries travel over fewer hops. As we shall soon see, this property along with the enhanced stability of the top-level overlay can significantly reduce querying delays.
– *Less messages in the wide-area:* If the most stable peers form the top-level DHT, most overlay reconstruction messages happen inside groups, which gather peers that are topologically close. Less hops per lookup means also less messages exchanged for the same number of requests. Finally, as we shall see shortly, the hierarchical organization of the peer groups is perfectly adapted to content caching, which can further reduce the number of messages that need to get out of the group.

3.2 Intra-Group Lookup

The framework we just described is quite flexible and accommodates any one of a number of overlay structures and lookup services at any level in the hierarchy. At the intra-group level, the groups can use different overlays, which could all be different from the top-level overlay structure.

If a group has a small number of peers (say, in the tens), each peer in the group could track all the other peers in the group (their ids and IP addresses); the group could then use CARP [18] or consistent hashing [19] to assign and locate keys within the group. The number of steps to perform such an intra-group lookup in the destination group is $O(1)$, since each peer runs a local hash algorithm to determine the peer in the group responsible for a key (g_2 in Figure 3).

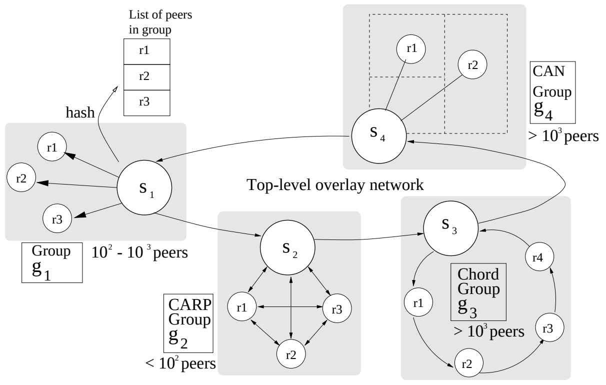

Fig. 3. The case of a ring-like overlay network with a single superpeer per group. Intra-group lookup is implemented using different lookup services (CARP, Chord, CAN).

If the group is a little larger (say, in the hundreds), then the superpeers could track all the peers in the group. In this case, by forwarding a query to a local superpeer, a peer can do a local lookup in $O(1)$ steps (g_1 in Figure 3).

Finally, if the group is large (say, thousands of peers or more) then a DHT such as Chord, CAN, Pastry, or Tapestry can be used within the group (g_3 and g_4 in Figure 3). In this case, the number of steps in the local lookup is $O(\log M)$, where M is the number of peers in the group.

3.3 Hierarchy and Group Management

In the two-tier hierarchical DHT, peers are organized in groups and, in turn, groups are divided in regular peers and superpeers. We now briefly describe the protocols used to manage groups.

Consider peer p joining the hierarchical DHT. We assume that p is able to get the id g of the group it belongs to (e.g., g may correspond to the name of p's ISP or university campus). First, p contacts and asks another peer p' already part of the P2P network to look up p's group using key g. Following the first step of the hierarchical lookup, p' locates and returns the IP address of the superpeer(s) of the group responsible for key g. If the group id of the returned superpeer(s) is precisely g, then p joins the group using the regular join mechanisms of the underlying intra-group lookup service; additionally, p notifies the superpeer(s) of its CPU and bandwidth resources. If the group id is not g, then a new group is created with id g and p as only (super)peer.

In a network with m superpeers per group, the first m peers to join a group g become the superpeers of that group. However, as previously mentioned, superpeers are expected to be the most stable and powerful group nodes. Therefore, we let superpeers monitor the peers that join a group and present "good" characteristics. Superpeers keep an ordered list of the superpeer candidates: the longer a peer remains connected and the higher its resources, the better a superpeer candidate it becomes. This list is sent periodically to the regular peers of the group. When a superpeer fails or disconnects, the first regular peer in the list becomes superpeer and joins the top-level overlay. It informs all peers in its group of its arrival, as well as the superpeers of the neighboring groups.

We are thus able to provide stability to the top-level overlay using multiple superpeers, promote the most stable peers as superpeers, and rapidly repair the infrequent failures or departures of superpeers.

3.4 Content Caching

In many P2P applications, once a peer p determines the peer p' that is responsible for a key, p then asks p' for the file associated with the key. This file might be a music file (MP3), a video, a software package, a document, etc. If the path from p' to p traverses a congested or low-speed link, the file transfer delay will be long. In this section, we describe how hierarchical DHTs can be used to create groups of cooperative caches, which can significantly reduce file transfer delays.

In many hierarchical DHT setups, we expect the peers in a same group to be topologically close and to be interconnected by high-speed links. For example, a group might consist of peers in a corporate or university campus, with campus networks using 100 Mbps Ethernet. In such an environment, it is typically faster to retrieve files from other peers in the local group, rather than to retrieve the file from the global Internet. Also, by frequently confining file transfers to intra-group transfers, we reduce traffic loads on the access links between the groups and higher-tier ISPs.

Such hierarchical setups can be naturally extended to implement cooperative caching. Consider the following modification to the lookup algorithm. When a peer $p \in G_i$ wants to obtain the file associated with some key k, it first uses group G_i's intra-lookup algorithm to find the peer $p' \in G_i$ that would be responsible for k if G_i were the entire set of peers. If p' has a local copy of the file associated with k, it returns the file to p; otherwise, p' obtains the file (using the hierarchical DHT), caches a copy, and forwards the file to p. In this manner, files are cached in the groups where they have been previously requested (until they are replaced by a cache replacement algorithm, such as LRU). If a significant fraction of requests are for popular files, then a significant fraction of file transfers will be intra-group transfers. Thus, the hierarchical design can be used to create a P2P content distribution network. As a function of file sizes, cache sizes, request distributions and access bandwidth, one can use standard analytical techniques [20] to quantify the reduction in average file transfer time and load on access links.

4 Chord Instantiation

For the remainder of this paper we focus on a specific top-level DHT, namely, Chord. In this context, we show that the two-tier design can significantly reduce lookup latency and file transfer latency.

4.1 Overview of Chord

In Chord, each peer and each key has a m-bit id. Ids are ordered on a circle modulo 2^m. Key k is assigned to the first peer whose identifier is equal to or follows k in the identifier space. This peer is called the successor of key k.

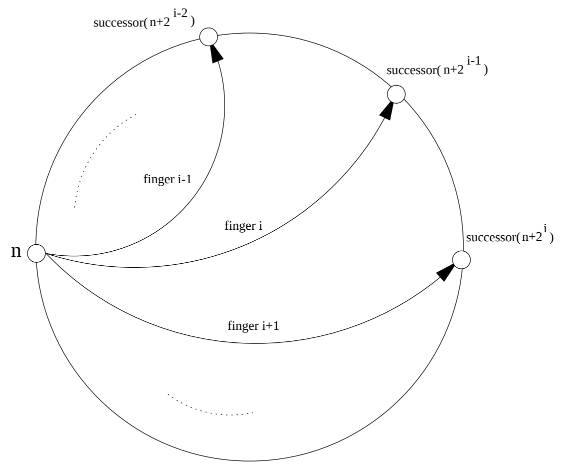

Fig. 4. Fingers of a peer in a Chord ring.

Each peer tracks its successor and predecessor peer in the ring. In addition, each peer tracks m other peers, called *fingers*; specifically, a peer with id p tracks all the successors of the ids $p + 2^{j-1}$ for each $j = 1, \ldots, m$ (note that p's first finger is in fact its successor). The successor, predecessor, and fingers make up the Chord routing table (see Figure 4).

During a lookup, a peer forwards a query to the finger with the largest id that precedes the key value. The process is repeated from peer to peer until the peer preceding the key is reached, which is the "closest" peer to the key. When there are P peers, the number of hops needed to reach the destination is $O(\log P)$ [1].

4.2 Inter-Group Chord Ring

In the top-level overlay network, each "node" is actually a group of peers. This implies that the top-level lookup system must manage an overlay of groups, each of which is represented by a set of superpeers that may fail independently from the group they belong to. Chord requires some adaptations for being used in the top-level overlay network and managing groups instead of nodes. We will refer to the modified version of Chord as "top-level Chord".

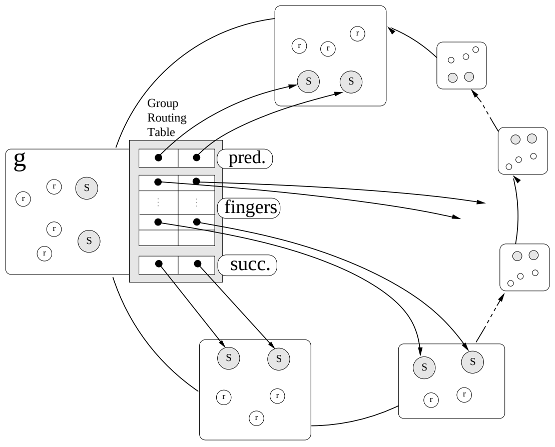

Fig. 5. Group routing table in the top-level Chord overlay network.

Each node in top-level Chord has a predecessor and successor *vector* (instead of a pointer). The vectors hold the IP addresses of the superpeers of the predecessor and successor group in the ring, respectively. Each finger is also a vector, which holds the IP addresses of the superpeers of another group an the ring. The routing table of a top-level Chord group is shown in Figure 5.

The population of groups in the top-level overlay network is expected to be rather stable. However, individual superpeers may fail and disconnect the top-level Chord ring. When the identity of the superpeers S_i of a group g_i changes, the new superpeers eagerly update the vectors of the predecessor and successor groups. This guarantees that each group has an up-to-date view of its neighboring groups and that the ring is never disconnected.

Fingers improve the lookup performance, but are not necessary for successfully routing requests (a request can always be routed by following the successor pointers until it reaches its destination. However, this routing leads to $O(P)$ hops to reach the destination). Therefore, when the superpeers S_i of a group g_i change, we do not immediately update the fingers that point to g_i. Instead, we lazily update the finger tables when we detect that they contain invalid references (similarly to the lazy update of the fingers in regular Chord rings [1]). It is worth noticing that the regular Chord must perform a lookup operation to find a lost finger. Due to the redundancy that our multiple superpeer approach provides, we can choose without delay another superpeer in the finger vector for the same group.

To route a request to a group pointed to by a vector (successor or finger), we choose a random IP address from the vector and forward the request to that superpeer. When groups have more than one superpeer, this load balancing strategy avoids overloading specific nodes or communication links, and increases the scalability of the system.

4.3 Lookup Latency with Hierarchical Chord

In this section, we quantify the improvement of lookup latency due to the hierarchical organization of the peers. To this end, we compare the lookup performance of two DHTs. The first DHT is the flat Chord DHT. The second DHT is a two-tier hierarchy in which Chord is used for the top level overlay, and arbitrary DHTs are used for the bottom level overlays. For each bottom level group, we only suppose that the peers in the group are topologically close so that intra-group lookup delays are negligible. We shall show that the hierarchical DHT can significantly reduce the lookup latency, owing to the heterogeneous availabilities of the peers and the reduction of nodes in the Chord ring.

In order to make a fair comparison, we suppose that both the flat and hierarchical DHTs have the same number of peers, denoted by P. Let I be the number of groups in the hierarchical design. Because peers are joining and leaving the ring, the finger entries in the peers will not all be accurate. This is more than probable, since fingers are updated lazily. To capture the heterogeneity of the peers, we suppose that there are two categories of peers:

– *Stable peers*, for which each peer is down with probability p_s.
– *Instable peers*, for which each peer is down with probability p_r, with $p_r >> p_s$.

We suppose that the vast majority of the peers are instable peers. In real P2P networks, like Gnutella, most peers just remain connected the time of getting data from other peers. This fact has already been observed in [21]. For the hierarchical organization, we select superpeers from the set of stable peers, and we suppose there is at least one stable peer in each group. Because there are many more instable peers than stable peers, the probability that a randomly chosen Chord node is down in the flat DHT is approximately p_r. In the hierarchical system, because all the superpeers are stable peers, the probability that a Chord node is down is p_s.

To compare the lookup delay for flat and hierarchical DHTs, we thus only need to consider a Chord ring with N peers, with each peer having the same probability p of being down. The flat DHT corresponds to $(N,p) = (P,p_r)$ and the hierarchical DHT corresponds to $(N,p) = (I,p_s)$. We now proceed to analyze the lookup of the Chord ring (N,p). To simplify the analysis, we assume the N peers are equally spaced on the ring, i.e., the distance between two adjacent peers is $\frac{2^m}{N}$. Our model implies that when a peer attempts to contact a peer in its finger table, the peer in the finger table will be down with probability p, except if this is the successor peer, for which we suppose that the finger entry is always correct (i.e., the successor is up or the peer is able to find the new successor. This assures the correct routing of lookup queries).

Given an initial peer and a randomly generated key, let the random variable H denote the number of Chord hops needed to reach the target peer, that is, to reach the peer responsible for the key. Let T be the random variable that is the clockwise distance in number of peers from the initial peer to the target peer. We want to compute the expectation $E[H]$. Clearly

$$\begin{aligned} E[H] &= \sum_{n=0}^{N-1} P(T=n)E[H|T=n] \\ &= \frac{1}{N}\sum_{n=0}^{N-1} E[H|T=n] \end{aligned} \tag{1}$$

From (1), it suffices to calculate $E[H|T=n]$ to compute $E[H]$. Let $h(n) = E[H|T=n]$. Note that $h(0)=0$ and $h(1)=1$. Let

$$j_n = \max\{j \;:\; 2^j \le \frac{2^m n}{N}\}$$

The value j_n represents the number of finger entries that precede the target peer, excluding finger 0, the successor. For each of the finger entries, the probability that the corresponding peer is up is p (the successor is assumed always up, so Chord routing is assured to eventually succeed).

Starting at the initial peer, when hopping to the next peer, the query will advance $\lceil \frac{2^{j_n}}{2^m/N} \rceil$ peers if the j_nth finger peer is up; if this peer is down but the (j_n-1)th finger peer is up, the query will advance $\lceil \frac{2^{j_n-1}}{2^m/N} \rceil$; and so on. Let $q_n(i)$ denote the probability that the ith finger is used. We therefore have

$$h(n) = 1 + \sum_{i=0}^{j_n} q_n(i) h\Big(n - \Big\lceil \frac{2^i}{2^m/N} \Big\rceil\Big)$$

The probability that the ith finger is used is given by

$$q_n(i) = p^{j_n - i}(1-p) \;\; i = 1,\ldots,j_n$$

and by $q_n(0) = p^{j_n}$. Combining the above two equations we obtain

$$h(n) = 1 + p^{j_n} h(n-1) + (1-p)\sum_{i=1}^{j_n} p^{j_n-i} h\Big(n - \Big\lceil \frac{2^i}{2^m/N} \Big\rceil\Big)$$

Using this recursion, we can calculate all the $h(n)$'s beginning at $h(0)=0$. We then use (1) to obtain the expected number of hops, $E[H]$.

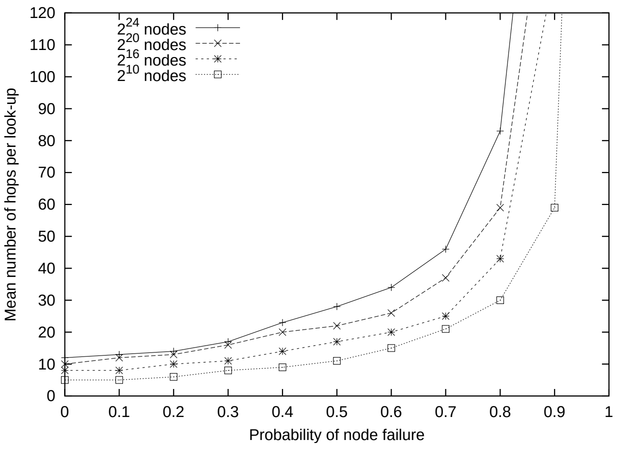

Fig. 6. Mean number of hops per lookup in Chord.

In Figure 6, we plot the expected number of hops in a lookup as a function of the availability of the peers in a Chord system, for different values of N. We can observe an exponential increase in the number of steps only for $p > 0.7$. With smaller values of p, although peers in the ring are not totally reliable, we are still able to advance quite quickly on the ring. Indeed, while the best finger for a target peer is unavailable with probability p, the probability of the second best choice to be also down is p^2, which is far smaller than p.

Despite the good scalability of the Chord lookup algorithm in a flat configuration, the hierarchical architecture can yet significantly decrease the lookup delay. Table 1 gives the expected number of hops for the flat and hierarchical schemes, for different values of P, I, and p_r ($p_s = 0$). We suppose in all cases groups of $\frac{P}{I}$ peers. As an example, for

	Flat		Hierarchical
	$p_r = 0.5$	$p_r = 0.8$	$p_s = 0$
$P = 2^{16}$ $I = 2^{10}$	17	43	5
$P = 2^{20}$ $I = 2^{16}$	22	59	8
$P = 2^{24}$ $I = 2^{20}$	28	83	10
$P = 2^{24}$ $I = 2^{16}$	28	83	8

Table 1. Comparing the flat and hierarchical networks.

$P = 2^{20} = 1,048,676$ peers, and $p_r = 0.8$, we observe that about 59 hops are needed on average. However, if we organize these peers in $I = 2^{16} = 65536$ groups of 16 peers each, the number of hops is reduced to 8. Since the number of steps is directly related to the lookup delay, we can conclude that the average lookup delay is divided by a factor of 7 in this case.

5 Conclusion

Hierarchical organizations in general improve overall system scalability. An example of a hierarchical organization is the Internet routing architecture with intra-domain routing protocols such as RIP or OSPF, and with BGP as inter-domain routing protocol. In this paper, we have proposed a generic framework for the hierarchical organization of peer-to-peer overlay network, and we have demonstrated the various advantages it offers over a flat organization. A hierarchical design offers higher stability by using more "reliable" peers (superpeers) at the top levels. It can use various inter- and intra-group lookup algorithms simultaneously, and treats join/leave events and key migration as local events that affect a single group. By gathering peers into groups based on topological proximity, a hierarchical organization also generates less messages in the wide area and can significantly improve the lookup performance. Finally, as shown in Section 3, our architecture is ideally suited for caching popular content in local groups. By first querying the responsible peer within ones own group, popular objects are dynamically pulled into the various groups. This local-group caching can dramatically reduce download delays in peer-to-peer file-sharing systems.

We have presented an instantiation of our hierarchical peer organization using Chord at the top level. The Chord lookup algorithm required only minor adaptations to deal with groups instead of individual peers. We have analyzed and quantified the improvement in lookup performance of hierarchical Chord. When all peers are available, an hierarchical organization reduces the length of the lookup path by a factor of $\frac{\log P}{\log I}$, where I is the number of groups and P is the total number of peers. When each peer is down with a certain probability, a hierarchical organization reduces the length of the lookup path dramatically for the case where the failure probability of superpeers is smaller than the failure probability of regular peers.

References

1. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of SIGCOMM 2001*, August 2001.
2. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of SIGCOMM 2001*, Aug. 2001.
3. 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)*, (Heidelberg, Germany), pp. 329–350, November 2001.
4. B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr 2001.
5. A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of SOSP 2001*, Oct. 2001.
6. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *Proceedings of SOSP 2001*, Oct. 2001.
7. S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized, peer-to-peer web cache," in *Proceedings of PDOC 2002*, July 2002.
8. A. Rowstron, A.-M. Kermarrec, P. Druschel, and M. Castro, "Scribe: The design of a large scale event notification infrastructure," in *Proceedings of NGC 2001*, Nov. 2001.
9. S. Zhuang, I. Stoica, D. Adkins, S. Ratnasamy, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proceedings of IPTPS'02*, (Cambridge, MA), Mar. 2002.
10. A. Andrzejak and Z. Xu, "Scalable, efficient range queries for grid information services," in *Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing*, Linkping University, Sweden, September 2002.
11. K. Aberer, "P-grid: A self-organizing access structure for p2p information systems," in *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS 2001)*, (Trento, Italy), 2001.
12. "Gnutella." http://gnutella.wego.com.
13. "Kazaa." http://www.kazaa.com.
14. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proceedings of Infocom'02*, (New York City, NY), 2002.
15. B. Krishnamurthy, J. Wang, and Y. Xie, "Early measurements of a cluster-based architecture for p2p systems," in *ACM SIGCOMM Internet Measurement Workshop*, (San Francisco, CA), November 2001.
16. B. Y. Zhao, Y. Duan, L. Huang, A. D. Joseph, and J. D. Kubiatowicz, "Brocade: Landmark routing on overlay networks," in *In Proceedings of IPTPS'02*, (Cambridge, MA), March 2002.
17. M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting network proximity in peer-to-peer overlay networks," Tech. Rep. MSR-TR-2002-82, Microsoft Research, One Microsoft Way, Redmond, WA 98052, 2002.
18. K. W. Ross, "Hash-routing for collections of shared web caches," *IEEE Network Magazine*, vol. 11, 7, pp. 37–44, Nov-Dec 1997.

19. D. Karger, A. Sherman, A. Berkhemier, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, “Web caching with consistent hashing,” in *Eighth International World Wide Web Conference*, May 1999.
20. J. F. Kurose and K. W. Ross, *Computer Networks: A Top-Down Approach Featuring the Internet, 2nd edition*. Addison Wesley, 2002.
21. E. Adar and B. A. Huberman, “Free riding on gnutella,” *First Monday*, vol. 5, Oct. 2000.