D1HT: A Distributed One Hop Hash Table [*]

Luiz R. Monnerat[•,⋆] and Claudio L. Amorim[⋆]
[⋆]COPPE - Computer and Systems Engineering [•]TI/TI-E&P/STEP
Federal University of Rio de Janeiro PETROBRAS
{monnerat,amorim}@cos.ufrj.br

Abstract

Distributed Hash Tables (DHTs) have been used in a variety of applications, but most DHTs so far have opted to solve lookups with multiple hops, which sacrifices performance in order to keep little routing information and minimize maintenance traffic. In this paper, we introduce D1HT, a novel single hop DHT that is able to maximize performance with reasonable maintenance traffic overhead even for huge and dynamic peer-to-peer (P2P) systems. We formally define the algorithm we propose to detect and notify any membership change in the system, prove its correctness and performance properties, and present a Quarantine-like mechanism to reduce the overhead caused by volatile peers. Our analyses show that D1HT has reasonable maintenance bandwidth requirements even for very large systems, while presenting at least twice less bandwidth overhead than previous single hop DHT.

1. Introduction

Distributed hash table systems (DHTs) provide scalable and practical solutions to store, locate, and retrieve information widely dispersed in huge distributed environments. For this reason, DHTs have already been proposed as a base for a variety of distributed and P2P applications, ranging from grid services [23] to databases [7], showing the large acceptance of DHTs as a useful distributed software.

DHT systems implement a hash-table-like lookup facility where the keys (information) are distributed among the participant nodes. In order to route a given lookup from its origin to the node in charge of the target key, DHTs implement overlay networks with routing information stored in each node (routing tables). Unless each routing table is large enough to hold the IP addresses of all participant nodes, the routing of a single lookup is likely to require multiple hops, i.e., the lookup should hop through a number of nodes before reaching the target.

While big routing tables allow faster lookups, they require higher communication bandwidth in order to be kept up to date as nodes join and leave the system, specially in very dynamic systems (i.e., systems with a high frequency of node joins and leaves). As a result, DHTs tradeoff lower lookup's latency (number of hops) for less bandwidth overhead (in order to maintain the routing tables). In a society where speed and information are critical while network bandwidth improves over time, we think that this tradeoff should favor latency rather than bandwidth. In contrast, most DHTs that have been proposed so far solve the lookups with multiple hops (e.g. [14, 18, 21, 24, 25]) in an attempt to minimize the *maintenance traffic* (network traffic required to maintain the routing tables). However, recent results [10] have shown that in some cases single-hop DHTs may generate less traffic than multi-hop ones, even for dynamic systems. Those results corroborate previous work [19], which indicated that low-overhead multi-hop DHTs are required only for vast and very dynamic systems. On the other hand, there is only one proposed DHT system that ensures that most lookups are really solved with only one hop [4], but this system imposes high levels of load imbalance and bandwidth overheads in order to maintain the routing tables.

We consider that an effective single-hop DHT must exhibit the following four main properties: 1) to solve a large fraction of all lookups with one single hop (e.g. 99%); 2) to have low bandwidth overheads; 3) to provide good load balance of the maintenance traffic among the nodes; 4) to be able to adapt to changes in the system dynamics. In this paper, we present D1HT, a novel one-hop P2P DHT system that is able to attend all four essential characteristics with an efficient Event Detection and Reporting Algorithm (EDRA). We formally describe this algorithm and prove its correctness, performance, and load balance properties. Our analytical results show that D1HT nodes have at least twice and up to one order of magnitude less maintenance bandwidth requirements than those of nodes in previous single-hop DHT [4]. Our results also show that D1HT is able to support vast P2P systems whose dynamics are similar to those of widely deployed P2P applications, such as Gnutella [22] and BitTorrent [2], with reasonable maintenance bandwidth demands. For instance, a huge one-million D1HT system, with dynamics similar to BitTorrent, would require only 3 kbps of duplex maintenance traffic to assure

[*]This research was partially sponsored by Brazilian CNPq and FINEP.

1-4244-0054-6/06/$20.00 ©2006 IEEE

that 99% of the lookups are solved with just one hop. For a 100K node D1HT system these requirements will drop to 0.4 kbps, which are negligible for most node connections. We also presented a Quarantine mechanism that is able to reduce the overhead caused by volatile nodes, but requires that lookups issued by recently connected nodes take two hops to be solved.

The remainder of this paper is organized as follows. Section 2 presents the D1HT design. Section 3 describes EDRA and proves its correctness and performance properties. Section 4 shows how EDRA behaves in the presence of message delays and other practical issues. In Section 5, we analyze D1HT performance. Section 6 discuss related work and Section 7 concludes the paper.

2. System Design

A D1HT system is composed of a set $\mathbb{D}$ of n peers[1] and maps items (or keys) to peers based on *consistent hashing* [8], where both peers and keys are hashed to integer identifiers (IDs) in the same ID space $[0..N]$, $N \gg n$. Typically a key ID is the cryptographic hash SHA-1 of the key value, a peer ID is based on the SHA-1 hash of it's IP address (or the SHA-1 hash of the user name), and $N = 2^{160} - 1$. For simplicity, from now on we will refer to the peers and keys IDs as the peers and keys themselves.

As in Chord[24], D1HT uses a ring topology where ID 0 succeeds ID N, and the *successor* and *predecessor* of an ID i are respectively the first living peers clockwise and counterclockwise from i in the ring. Each key is assigned to the key's successor and is replicated on the following $\log_2(n)$ peers clockwise in the ring.

Each peer in a D1HT system maintains a routing table with the IP addresses of all peers in the system, and so any lookup is trivially solved with just one hop, provided that the local routing table is up to date. Note that if a peer p does not acknowledge an event caused by a membership change in the system, p may route a lookup to a wrong peer or to a peer that has already left the system. In the former case, the peer that received the lookup will forward it according to its own routing table. In the latter, a time out will occur and p will re-issue the lookup to the successor of the original target. In both cases, the lookup should eventually succeed, but it will take longer than initially expected. As one of the main goals of one hop DHTs is performance, we should try to keep those routing failures[2] to a minimum, by means of an algorithm that allows fast dissemination of the events without high bandwidth overheads and load imbalance. This algorithm will be presented in Section 3.

D1HT adds small memory overhead to each peer by exploiting the fact that the ID space is sparsely occupied so that each peer can store its routing table as a local hash table. The table index is based on the first bits of the peer IDs, avoiding the need to store the IDs themselves. In this way, D1HT routing tables will require approximately $4n$ bytes, plus some extra space to allow D1HT peers to treat the eventual collisions.

To join a D1HT system a peer should first be able to locate just one peer p_{any} already in the system. The joining peer then hashes its IP address (or the local user name) to get its ID p and asks p_{any} to issue a lookup for p, which will return the IP address of p's successor p_{succ}. The joining peer p will then contact p_{succ} in order to be inserted in the ring and to get the information about the keys it will be responsible for. To feed its routing table, p will ask p_{succ} for the addresses of a number of peers in the system. Peer p will then *ping* each one of those peers and choose the nearest ones to ask for the routing table. In Section 4.3 we will present an alternative joining method aiming to reduce the overhead caused by volatile peers. To track peer crashing, each peer is in charge of detecting if its predecessor has left the system.

In this paper, we will not address issues related to malicious nodes and network attacks, although it is clear that, due to their high out degree, one hop DHTs are naturally less vulnerable to those kinds of menaces than low-degree multi-hop DHTs.

Before proceeding to the next sections, we will introduce a few functions to make the presentation clearer. For any $i \in \mathbb{N}$ and $p \in \mathbb{D}$, the i_{th} successor of p is given by the function $succ(p, i)$, where $succ(p, 0) = p$ and $succ(p, i)$ is the successor of $succ(p, i - 1))$ for $i > 0$. Note that for $i \geq n$, $succ(p, i) = succ(p, i - n)$. In the same way, the i_{th} predecessor of a peer p is given by the function $pred(p, i)$, where $pred(p, 0) = p$ and $pred(p, i)$ is the predecessor of $pred(p, i - 1))$, for $i > 0$. As in [14], for any $p \in \mathbb{D}$ and $k \in \mathbb{N}$, $stretch(p, k) = \{\forall p_i \in \mathbb{D} \mid p_i = succ(p, i) \wedge 0 \leq i \leq k\}$. Note that $stretch(p, n - 1) = \mathbb{D}$ for any $p \in \mathbb{D}$.

3. Routing Table Maintenance

As each peer in a D1HT system should know the IP address of every other peer, any event (from now on we will refer to peer joins and leaves simply as *events*) should be acknowledged[3] by all peers in the system in a timely fashion in order to avoid stale entries in routing tables. On the other hand, as we address large and dynamic systems, we should avoid fast but naïve ways to disseminate information about the events (e.g., broadcast), as they can easily overload the network and create hot spots. In that way, the detection and propagation of events impose three important challenges

[1] Since D1HT uses a pure P2P architecture, we will refer to the D1HT nodes simply as *peers*.

[2] As the lookup will eventually succeed, we do consider it as *routing failure* instead of *lookup failure*.

[3] We define that a peer *acknowledges* an event when either it detects the join (or leave) of its predecessor or when it receives a message notifying an event.

to D1HT: minimize bandwidth consumption, provide fair load balance, and assure an upper bound on the fraction of stale entries in routing tables. To accomplish these requirements, we propose the Event Detection and Report Algorithm (EDRA for short) that is able to notify an event to the whole system in logarithmic time and yet to have good load-balance properties coupled with very low bandwidth overhead. Additionally, EDRA is able to dynamically adapt to changes in system behavior to continuously satisfy a pre-defined upper bound on the fraction of routing failures.

3.1. Event Dissemination

We will begin this section with a brief description of EDRA, and we will then formally define it. To disseminate the information about the events, each peer p sends up to ρ propagation messages at each Θ secs time interval, where $\rho = \lceil \log_2(n) \rceil$ and Θ is based on the system dynamics (as it will be seen in Section 4.2). Each message $M(l)$ will have a Time-To-Live (TTL) counter l in the range $[0..\rho)$, and will be addressed to $succ(p, 2^l)$. Besides, p will include in each message $M(l)$ all events brought to p by any message $M(j), j > l$, received in the last Θ secs. To initiate an event report, the successor of the peer suffering the event will include it in all messages sent at the end of the current Θ interval. Figure 1, which will be further described in Section 3.2, illustrates the dissemination of an event in a D1HT system with 11 peers.

The rules below formally define the EDRA algorithm we described above:

Rule 1: Every peer will send at least one and up to ρ messages at the end of each Θ secs interval (Θ interval), where $\rho = \lceil \log_2(n) \rceil$.

Rule 2: Each message will have a Time To Live counter (TTL) in the range 0 to $\rho - 1$, and carry a number of events. All events brought by a message with $TTL = l$ will be acknowledged with $TTL = l$ by the receiving peer.

Rule 3: A message will only contain events acknowledged during the ending Θ interval. An event acknowledged with $TTL = l, l > 0$, will be included in all messages with $TTL < l$ sent at the end of the current Θ interval. Events acknowledged with $TTL = 0$ will not be included in any message.

Rule 4: The message with $TTL = 0$ will be sent even if there is no event to report. Messages with $TTL > 0$ will only be sent if there are events to be reported.

Rule 5: If a peer does not receive any message from its predecessor for T_{detect} secs, it assumes that the predecessor has left the system.

Rule 6: When a peer detects an event in its predecessor (it has joined or left the system), this event is considered to have been acknowledged with $TTL = \rho$, and so is reported through ρ messages according to rule 3.

Rule 7: A peer p will send all messages with $TTL = l$ to $succ(p, 2^l)$.

Rule 8: Before sending a message to $succ(p, k)$, p will discharge all events related to any peer in $stretch(p, k)$.

3.2. EDRA Correctness

The above rules ensure that EDRA will deliver any event to all peers in a D1HT system in logarithmic time, as we will show in Theorem 3.1 shortly. For that theorem we will ignore message delays and we will consider that all peers have synchronous intervals, i.e., the Θ intervals of all peers start at exactly the same time. In Section 4.1 we will take into account those effects. The absence of message delays means that any message will arrive immediately at its destination, and since we are also considering synchronous Θ intervals, any message sent at the end of a Θ interval will arrive at its destination at the beginning of the subsequent Θ interval (as represented in Figure 2, Section 4.1).

Theorem 3.1. *An event ε that is acknowledged by a peer p with $TTL = l$, and by no other peers in $\mathbb{D}$, will be forwarded by p through l messages in a way that ε will be acknowledged exactly once by all peers in $stretch(p, 2^l - 1)$ and by no other peer in the system. The average time T_{sync} for a peer in $stretch(p, 2^l - 1)$ to acknowledge ε will be at most $l \cdot \Theta/2$ secs after p had acknowledged ε.*

Proof: By strong induction in l. For $l = 1$ the rules imply that p will only forward ε through a message with $TTL = 0$ addressed to $succ(p, 1)$. As this message should be sent at the end of the current Θ interval, $succ(p, 1)$ will acknowledge ε at most Θ secs after p had acknowledged it, making the average time for peers in $stretch(p, 1) = \{p, succ(p, 1)\}$ to be $T_{sync} = \Theta/2$ (at most). So the claim holds for $l = 1$.

For $l > 1$, the rules imply that p will forward ε through l messages at the end of the current Θ interval, each one with a TTL in the range 0 to $l - 1$. In that way, after Θ secs each peer $p_k = succ(p, 2^k)$, $0 \leq k < l$, will have acknowledged ε with $TTL = k$. Applying the induction hypothesis to each of those l acknowledgements, we have that each acknowledgment made by a peer p_k will imply that all peers in $stretch(p_k, 2^k - 1)$ will acknowledge ε exactly once. Accounting for all $l - 1$ acknowledgments made by the peers p_k, and remembering that rule 8 will prevent ε to be acknowledged twice by any peer in $stretch(p, 2^\rho - n)$, we will have that ε will be acknowledged exactly once by all peers in $stretch(p, 2^l - 1)$. As none of those peers will forward ε to a peer outside this range, ε will not be acknowledged by any other peers in the system. The induction hypothesis also assures that the average time for the peers in each $stretch(p_k, 2^k - 1)$ to acknowledge ε will be $k \cdot \Theta/2$ secs (at most) after the respective peer p_k had acknowledged it,

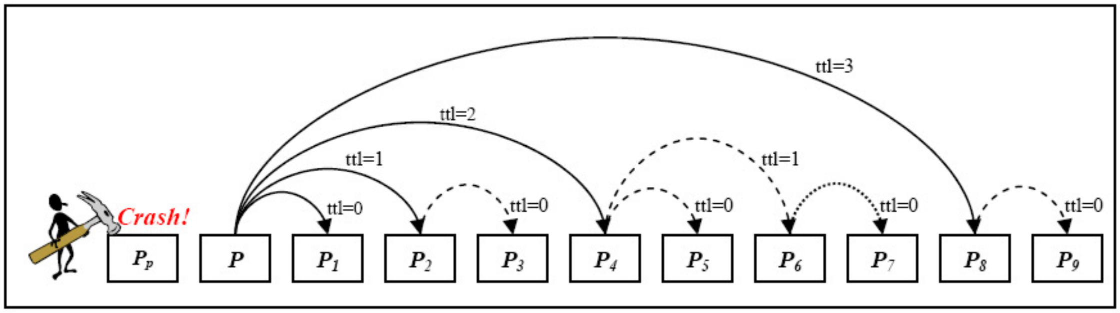

Fig. 1. Event propagation in a D1HT system with 11 peers (ρ=4), where $P_i = succ(P, i), 1 \leq i \leq 9$.

which will lead to $T_{sync} = l \cdot \Theta/2$ (at most) for peers in $stretch(p, 2^l - 1)$. ■

Applying Theorem 3.1 and the EDRA rules to a peer join (or leave) that is acknowledged by its successor p, we will have that this event will be further acknowledged exactly once by all peers in $stretch(p, n-1) = \mathbb{D}$. Besides, the average acknowledge time will be $\rho \cdot \Theta/2$ secs (at most). We can also show that the last peer to acknowledge the event will be $succ(p, n-1)$, $\rho \cdot \Theta$ secs after p had acknowledged the event.

Figure 1 shows how EDRA disseminates information about events and illustrates the properties that Theorem 3.1 has proved. The figure presents a D1HT system with 11 peers ($\rho = 4$), where peer P_p crashes and this event ε is detected and reported by its successor P. The peers are shown in a line instead of a ring to facilitate the presentation. Note that P acknowledges ε after T_{detect} secs (rule 5) with $TTL = \rho$ (rule 6). According to rules 3 and 7, P will forward ε with $\rho = 4$ messages addressed to $P_1 = succ(P, 2^0)$, $P_2 = succ(P, 2^1)$, $P_4 = succ(P, 2^2)$, and $P_8 = succ(P, 2^3)$, as represented by the solid arrows in the figure. Peers P_2, P_4, and P_8 will acknowledge ε with $TTL > 0$ (rule 2) and so those peers will forward ε with messages addressed to $P_3 = succ(P_2, 2^0)$, $P_5 = succ(P_4, 2^0)$, $P_6 = succ(P_4, 2^1)$, and $P_9 = succ(P_8, 2^0)$ represented by the dashed arrows in the figure. As P_6 will acknowledge ε with $TTL = 1$, it will further forward it to $P_7 = succ(P_6, 2^0)$ (doted arrow). Note that rule 8 prevents P_8 to forward ε to $succ(P_8, 2^1)$ and $succ(P_8, 2^2)$, which in fact are P and P_3, avoiding these two peers to acknowledge ε twice.

3.3. Load Balance and Performance

Theorem 3.1 not only proves that all peers will receive the necessary information to maintain their routing tables in logarithmic time, but also assures that no peer will receive redundant information. These results confirm that EDRA makes good use of the available bandwidth and provide perfect load balance in terms of incoming traffic.

As no peer will exchange maintenance messages with any other peer outside $\mathbb{D}$, we may assert that the average outgoing and incoming bandwidth requirements are the same, as well as the total number of messages sent and received. On the other hand, at first glance EDRA seems not to provide good balance in terms of outgoing traffic. For instance, an event ε with a peer p will be reported by its successor p_s through ρ messages, while p_s's successor will not even send a single message reporting ε. It is easy to show that in relation to the outgoing traffic to report one event, the maximum load will be on the successor of the peer that the event occurred, and it will be $O(\log(n))$ greater than the average load. However, this punctual load imbalance is not a main concern, as our target is large and dynamic systems, in which several events happen at every second, so that we should not be too concerned with the particular load that is generated by a single event. Nevertheless, we must guarantee good balance in respect to the aggregate traffic that is necessary to disseminate information about all the events as they happen.

In a D1HT system the load balance in terms of number of messages and outgoing bandwidth will rely on the random distribution properties of the hashing function it uses. The chosen hash function is expected to randomly distribute the peers IDs along the ring, which can be accomplished by using a cryptographic hash function such as SHA-1[16]. Then, as in many other studies (e.g. [4, 9, 10, 12, 14, 24]), we will assume that the events are oblivious to the peers IDs, leading to a randomly distributed rate of r events per second in the system, and so the average amounts of incoming and outgoing traffic per peer will be (including message acknowledgments):

$$(2 \cdot N_{msgs} \cdot v + r \cdot m \cdot \Theta)/\Theta \text{ bits/secs} \qquad (3.1)$$

where N_{msgs} is the average number of messages a peer sends (and receives) per Θ interval, m is the average number of bits necessary to describe an event, and v is the bit overhead per message.

We should point out that Equation 3.1 does not require r to be fixed. In fact, r will vary even in our simplest approach, since we will assume that the dynamics of a given D1HT system can be represented by its average session length S_{avg}, as in [4]. Here we refer to *session length* as the amount of time a peer is continuously connected to the D1HT system, i.e., the amount of time between a peer join and its subsequent leave. As each peer will generate two events per session (one join and one leave), the event rate can be calculated as follows:

$$r = 2 \cdot n / S_{avg} \tag{3.2}$$

Since the average session lengths of a number of different P2P systems have already been measured [2, 22], the equation above allows us to calculate event rates that are representative of widely deployed P2P applications.

3.4. Number of Messages

Equation 3.1 requires us to know the average number of messages a peer sends and receives, which is exactly the purpose of the following theorem.

Theorem 3.2. *The set of peers S for which a generic peer p acknowledges events with $TTL \geq l$ is such that $|S| = 2^{\rho - l}$.*

Proof: By induction on j, where $j = \rho - l$. For $j = 0$, rule 2 assures that there is no message with $TTL \geq l = \rho$. Then the only events that p acknowledges with $TTL \geq \rho$ are those related to its predecessor (rule 6), and so $S = \{pred(p, 1)\}$ and $|S| = 1 = 2^0 = 2^{\rho - l}$.

For $j > 0$, $l = \rho - j < \rho$. As S is the set of events that peer p acknowledged with $TTL \geq l$, we can say that $S = S1 \cup S2$, where $S1$ and $S2$ are the sets of events that were acknowledged with $TTL = l$ and $TTL > l$, respectively. From the induction hypothesis, we have that $|S2| = 2^{\rho-(l+1)}$. As $l < \rho$, $S1$ will not include the p predecessor (rule 6) and, as rule 7 assures that p only receives message with $TTL = l$ from a peer k, $k = pred(p, 2^l)$, we have that $S1$ will be the set of events that k sent through messages with $TTL = l$. From rule 3, we then have that $S1$ is the set of events that k acknowledged with $TTL = l + 1$ and, as the induction hypothesis also applies to the peer k, we have that $|S1| = 2^{\rho-(l+1)}$. From Theorem 3.1 we know that peer p acknowledges each event only once, assuring that $S1$ and $S2$ are disjoints and so $|S| = |S1| + |S2| = 2^{\rho-(l+1)} + 2^{\rho-(l+1)} = 2^{\rho-l}$. ■

Rules 3 and 4 assure that a peer p will only send a message with $TTL = l > 0$ if it acknowledges at least one event with $TTL \geq l + 1$. Based on Theorem 3.2 we can then say that p will only send a message with $TTL = l > 0$ if at least one in a set of $2^{\rho-l-1}$ peers suffers an event. As the probability of a generic peer to suffer an event in a Θ interval is $\Theta \cdot r/n$, and with the help of Equation 3.2, we can assure that the probability $P(l)$ of a generic peer to send a message with $TTL = l > 0$ at the end of each Θ interval is:

$$P(l) = 1 - (1 - 2\Theta/S_{avg})^k, \text{ with } k = 2^{\rho-l-1} \tag{3.3}$$

As the message with $TTL = 0$ will be sent in every Θ interval, we will then have that the average number of messages sent (and received) by each peer per Θ interval is:

$$N_{msgs} = 1 + \sum_{l=2}^{\rho} P(l) \tag{3.4}$$

Equations 3.1, 3.3, and 3.4 allow us to calculate the average maintenance traffic per peer based on the rate of events r, the system size n, and the duration of the Θ interval.

4. Practical Aspects

In this section, we will show how EDRA performs in the presence of message delays and asynchronous intervals, and how it can be tuned to adapt to changes in the system dynamics. We will also present the Quarantine mechanism to minimize the overheads caused by volatile peers.

4.1. Message Delays and Asynchronous Intervals

In Theorem 3.1, we did not consider the effects of message delays and asynchronous Θ intervals, so we will turn to them in this section.

Figures 2 and 3 show timelines representing the propagation of an event ε in ideal circumstances and in the presence of message delays and asynchronous Θ intervals. Each of those two figures illustrates three Θ intervals for each peer. Figure 2 shows the ideal situation where there is no message delay and the Θ interval of all peers starts simultaneously, and the dotted arrows indicate the messages reporting ε. In this hypothetical situation, each peer will add exactly Θ secs on the propagation time for ε, leading to $T_{sync} = \rho \cdot \Theta/2$ secs, as shown in Theorem 3.1.

Figure 3 illustrates a typical situation where the various Θ intervals are not synchronized and each message suffers a delay. We will consider an average message delay δ_{avg} for the whole system (which includes the average time spent with retransmissions). In this case, on average each message will take δ_{avg} secs to reach the target peer and will arrive at the middle of the Θ interval. So, each peer in the event dissemination path will add $\delta_{avg} + \Theta/2$ secs on average to the event propagation time, leading to the adjusted value $T_{async} = \rho \cdot (2 \cdot \delta_{avg} + \Theta)/4$ secs. Note that T_{async} has not yet considered the time to detect the event. As a peer will take up to T_{detect} secs to detect an event in its predecessor, the average acknowledge time will be $T_{detect} + T_{async}$ secs after the event had happened.

From now on, we will consider that $T_{detect} = 2\Theta$, which reflects the case where after one missing message with $TTL = 0$, a peer p will probe its predecessor p_p and, once it has confirmed that the p_p had left the system, p will report p_p failure at the end of the next Θ interval. So we can calculate the expected average acknowledge time for any

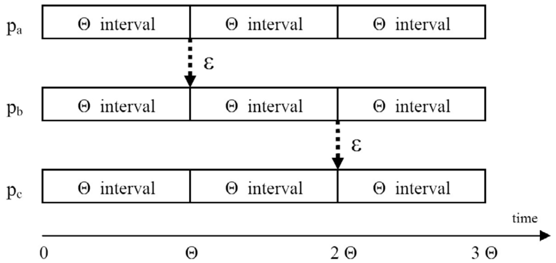

Fig. 2. Propagation of an event ε with synchronous Θ intervals and in the absence of message delays.

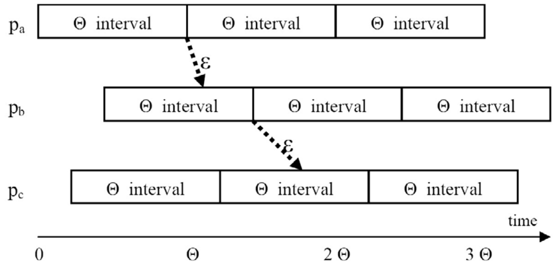

Fig. 3. Propagation of an event ε with asynchronous Θ intervals and message delays.

event:

$$T_{avg} = 2 \cdot \Theta + \rho \cdot (\Theta + 2 \cdot \delta_{avg})/4 \; secs \qquad (4.1)$$

Equation 4.1 is conservative since it only considers the worst case of peer failures, while $T_{detect} = 0$ for joins and voluntary leaves.

4.2. Tuning EDRA

By following the results as presented in [4], in this section we will show how to tune the event detection and reporting algorithm used by D1HT (EDRA) in order to assure that a high fraction of lookups (e.g. 99%) will be solved in the first attempt. In other words, our goal will be to assure that the fraction of the routing failures is below an acceptable maximum f as defined by the user (e.g. $f = 1\%$).

As the lookups are solved with just one hop, to achieve f it is enough to assure that the hops will fail with probability f at most. Assuming that the lookup targets are randomly spread along the ring (as in many other studies, e.g. [4, 9, 12, 10, 13, 15, 24]), the average fraction of routing failures will be a direct result of the number of stale routing tables' entries. In that manner, to satisfy a pre-defined average fraction of routing failures f, it suffices[4] to assure that the average fraction of stale routing table entries is kept below f [4].

As the average acknowledge time is T_{avg}, the average number of stale entries in the routing tables will be given by the numbers of events occurred in the last T_{avg} seconds, i.e., $T_{avg} \cdot r$. This implies that to accomplish a given f we should satisfy the inequality $T_{avg} \cdot r/n \leq f$. With this inequality and Equations 3.2 and 4.1, we have that the maximum value of Θ to satisfy a given f will be:

$$\Theta = \frac{2 \cdot f \cdot S_{avg} - 2 \cdot \rho \cdot \delta_{avg}}{8 + \rho} \; secs \qquad (4.2)$$

where both S_{avg} and δ_{avg} should be expressed in seconds.

As we have already pointed out in Section 3.3, it is not reasonable to expect r to be constant, and Equation 4.2 provides a way for EDRA to adapt to changes in the system dynamics, as it allows each peer to dynamically calculate Θ based on the rate of events that is observed locally.

4.3. Quarantine

In any DHT system, peer joins are costly as the joining peer has to collect information about its keys and the IP addresses to fill in its routing table, and this joining overhead may be useless if the peer quickly departs from the system. This problem is aggravated in the case of single hop DHTs as any joining peer should be acknowledged by the whole system. On the other hand, P2P measurement studies [3, 22] have shown that the statistical distributions of peer session lengths are usually heavy tailed, which means that peers that are connected to the system for a long time are likely to remain alive longer than newly arrived peers. To address those issues we proposed a *Quarantine* mechanism, where a joining peer will not be granted to immediately take part of the D1HT overlay network, though it will be allowed to perform lookups at any moment.

In the basic D1HT joining mechanism, a joining peer p retrieves the keys and IP addresses not only from its successor but also from a number of nearby peers (as described in Section 2). With Quarantine, those peers simply wait for a pre-defined quarantine period T_q before sending the keys and IP addresses to p, postponing its insertion in the D1HT overlay network. While p does not receive its keys and the necessary IP addresses, its join will not be reported and it will not be responsible for any key, but p will already be able to perform lookups by forwarding them to one of those nearby peers. In that way, we avoid the join overhead for peers with session lengths smaller than T_q, but newly incoming peers will have their lookups solved with two hops. We consider this extra hop penalty to be acceptable as the additional hop will be addressed to a nearby peer, while we expect to significantly reduce the impact of join overheads on D1HT. Besides, Quarantine may help to prevent malicious attacks, as we could tune it in a way that suspicious peers would take longer to be fully accepted by the D1HT overlay network.

[4] In fact it is another conservative assumption. Since each key is replicated along ρ consecutive peers, the lookup will probably succeed in the first attempt even if the peer issuing the lookup is not aware of the joining of up to $\rho - 1$ consecutive peers.

Parameter	OneHop	D1HT	Description
n	$[10^5, 10^6]$	$[10^5, 10^6]$, $[10^4, 10^7]$	Number of nodes in the system.
S_{avg}	174	60, 174, 300, 780	Average session duration in minutes.
v	160	160	Overhead per message (headers, etc.), in bits.
m	80	80	Number of bits necessary to describe an event.
f	1%	1%, 5%, 10%	Maximum acceptable fraction of routing failures.
δ_{avg}	-	0.280	Average message delay in seconds.

Table 1. Parameters we used in our analysis. The underlined values were used only in Section 5.4.

In a Quarantine D1HT system with n peers, only the q peers with session lengths longer than T_q will effectively take part of the overlay network and have their events reported, allowing a reduction in the maintenance traffic. We can quantify this maintenance traffic reduction by replacing n by q in Equation 3.2, leading to:

$$r = 2 \cdot q / S_{avg} \tag{4.3}$$

As the results from all other equations presented do not depend on n, they remain valid with Quarantine.

5. Analysis

In this section, we will quantify the amount of bandwidth required to maintain the routing tables in D1HT, and compare those results with the one hop DHT (OneHop) results as presented in [4]. In Section 5.4 we will present an extended D1HT analysis.

5.1. The OneHop DHT

OneHop was the first proposed DHT to assure that a high fraction of the lookups are solved with only one hop. In contrast to the pure P2P D1HT approach, the dissemination of the events in OneHop is based on a hierarchy, where the nodes[5] are grouped in *units*, which in turn are grouped in *slices*. As each unit and slice has a *leader*, the imposed hierarchy divides the nodes in three levels: *slice leaders*, *unit leaders*, and *ordinary nodes*. The propagation of events imposes the highest maintenance load on the slice leaders and the lowest load on the ordinary nodes. More details about OneHop can be found in [4].

5.2. Methodology

The evaluations of both D1HT and OneHop presented here are based on analytical results. The D1HT results are derived from Equations 3.1, 3.2, 3.3, 3.4 and 4.2. For OneHop we will present the analytical results reported in [4], which do not consider messages delays nor the overheads caused by slice and group leaders failures. In contrast, the D1HT results presented are based on proven properties and do consider messages delays and failures of any type of node. The results from both systems assume that the events and lookup targets are randomly distributed along the ring.

The results were obtained with the parameters listed in Table 1 (the D1HT's underlined values will only be used in Section 5.4). For simplicity we will express the S_{avg} values in minutes or hours instead of seconds. To assure fairness, the parameters used in Section 5.3 for both systems were taken from [4], where the average session duration was based on a study of Gnutella behavior[22]. The only exception is δ_{avg}, as the OneHop results do not consider message delays. For D1HT we used 0.280 secs for δ_{avg} (which already incorporates the average overhead due to message retransmissions), which is quite conservative in relation to the results presented in [22]. As in [4], the event rates were based on the average session length S_{avg}, according to Equation 3.2.

We should point out that while results based on simulations or real implementations are usually the preferred choice for systems evaluation, we argue that in our case the analytical results have special value, as they allow the study of very large systems. Note that it is not feasible to implement or even simulate a system with millions of peers just to evaluate a new proposal. In fact, so far most DHT evaluations based on real implementations used a hundred *physical* nodes at most (e.g. [7, 25]), while the DHT simulations presented are usually restricted to a maximum of 20K nodes (e.g. [4, 5, 7, 9, 10, 11, 15, 17, 24, 25]), and so they are not representative of popular P2P systems, which are able to support up to millions of users [1]. On the other hand, we believe that to be accepted as good estimates of real implementations, the analytical results should be based on proven properties and consider the most common and important real world problems, such as messages delays and retransmissions, which is the case with the D1HT results presented in this paper.

5.3. Comparative Analysis

In this section, we will study the maintenance bandwidth demands of D1HT and OneHop analytically. We will compare the demands of a D1HT peer against those of the best (ordinary nodes) and worst (slice leaders) OneHop cases.

We limited our comparison to system sizes in the range $[10^5, 10^6]$, since it was the interval with analytical results as reported in [4]. The OneHop analytical outgoing bandwidth requirements reported for ordinary nodes and slice leaders were respectively 3.84 kbps and 35 kbps for $n = 10^5$, rais-

[5] As it imposes a hierarchy among the nodes, we will avoid the term *peer* for OneHop.

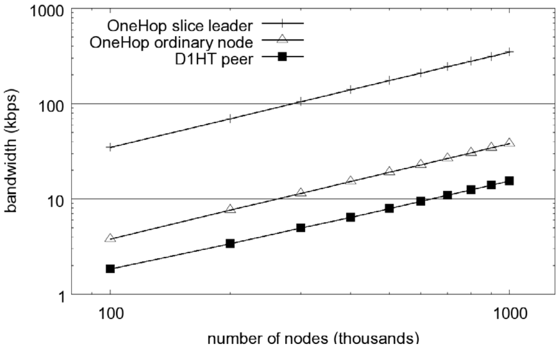

Fig. 4. Outgoing bandwidth demands for OneHop and D1HT.

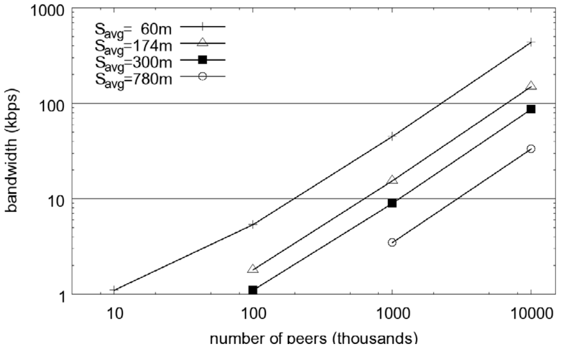

Fig. 5. D1HT peer bandwidth demands for $f = 1\%$ and different S_{avg} values.

ing linearly up to 38 kbps and 350 kbps for $n = 10^6$ [4]. Those results are plotted in Figure 4 (both axes are logarithmic), as well as the requirements for a D1HT peer.

Figure 4 shows that the outgoing bandwidth requirements for an OneHop ordinary node and a slice leader are at least twice and one order of magnitude higher, respectively, than those from a D1HT peer. For example, for $n = 10^5$ the demands for a D1HT peer, a OneHop ordinary node, and a slice leader are 1.8 kbps, 3.8 kbps, and 35 kbps respectively, growing to 16 kbps, 38 kbps and 350 kbps for $n = 10^6$.

5.4. Extended Analysis

In this section, we will study the D1HT sensitivity to variations in some analysis parameters according to the underlined values in Table 1. We will also study the Quarantine mechanism presented in Section 4.3, and extend the range of system sizes to $[10^4, 10^7]$, which are representative of current popular P2P systems like Gnutella, FastTrack, Overnet and eDonkey [1].

In the previous section, we showed both D1HT and OneHop requirements for systems with 2.9 hours of average session duration, as it was the value used in [4] based on the Gnutella behavior. However, recent measurements [2] have shown that other systems have much less dynamics, as the measured average session length for BitTorrent was 13 hours. On the other hand, we believe that a DHT system should also be prepared to face systems with smaller session lengths as well. To analyze those issues we studied the maintenance bandwidth requirements for a D1HT peer in systems with average sessions of 60, 174, 300, and 780 minutes. Besides being representative of widely deployed P2P applications such as Gnutella and BitTorrent, that range of values is more comprehensive than the ones used in most published DHT evaluations (e.g. [4, 9, 10, 11, 15]). Figure 5 plots those bandwidth requirements, omitting the results below 1 kbps as the axes are logarithmic. The figure shows that D1HT's bandwidth requirements are roughly linearly dependent on both the system size and the inverse of S_{avg}. For example, the requirements for a D1HT peer in systems with $n = 10^5$ and average sessions of 60, 174, 300, and 780 minutes are respectively 5 kbps, 1.8 kbps, 1.1 kbps, and 0.4 kbps, growing to 45 kbps, 16 kbps, 9 kbps, and 3.5 kbps in that order for $n = 10^6$. We believe those results show that with the technology available today, the D1HT maintenance overheads are acceptable for systems with S_{avg} as low as 60 minutes and up to 100 thousand peers, while only systems with S_{avg} larger than 300 minutes can support the D1HT requirements for one million peers.

In Figure 6 we plot the average number of messages sent per minute by a D1HT peer, according to the system size and S_{avg}. The figure shows that the number of messages sent is linearly proportional to both n and S_{avg}. For most combinations studied a D1HT peer sends less than one message per second, which is quite reasonable.

Figure 7 shows D1HT's bandwidth requirements for different values of f, omitting the results below 1 kbps as the axes are logarithmic. We notice small variations in bandwidth demands for different values of f in the interval of system sizes studied, which indicates that it is not the case to increase f in order to reduce the bandwidth requirements.

Figure 8 shows the Θ values that are necessary to achieve $f = 1\%$ for some values of S_{avg}. Note that Θ should be bigger than the average message delay in order to allow a peer to correctly detect its predecessor crashed. We see that values of Θ well above 1 sec are enough to satisfy $f = 1\%$, even for systems with 10 million peers and S_{avg}=1 hour.

The analysis of the Quarantine mechanism will be based on the Gnutella measurements presented in [3]. Those results show that 31% of the Gnutella sessions last less than 10 minutes, which is a convenient value for the Quarantine period T_q. Figure 9 plots the maintenance bandwidth requirements for D1HT systems with and without Quarantine (according to Equations 4.3 and 3.2 respectively), where $T_q = 10$ min and $q = 0.69 \cdot n$. The other parameters are the same as used in Section 5.3. As we expected, the maintenance overhead reductions are close to 31%, showing the effectiveness of our Quarantine mechanism.

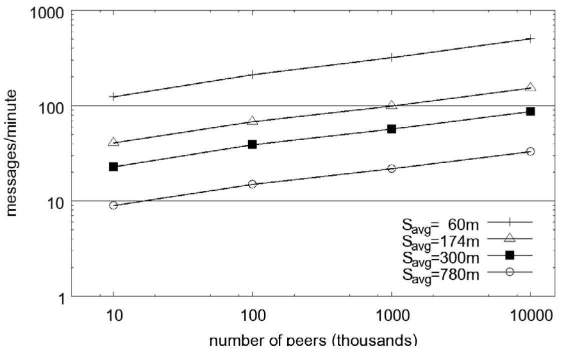

Fig. 6. Average number of messages sent by a D1HT peer for $f = 1\%$.

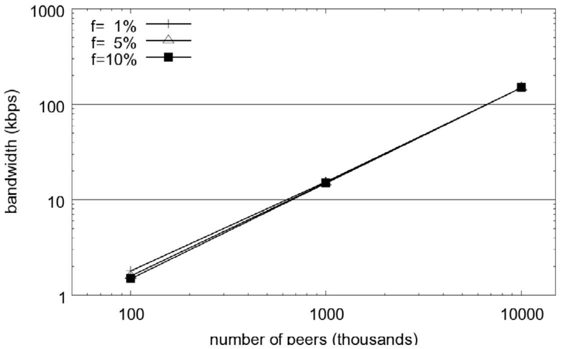

Fig. 7. D1HT peer bandwidth demands in kbits/sec for $S_{avg} = 2.9$ hours.

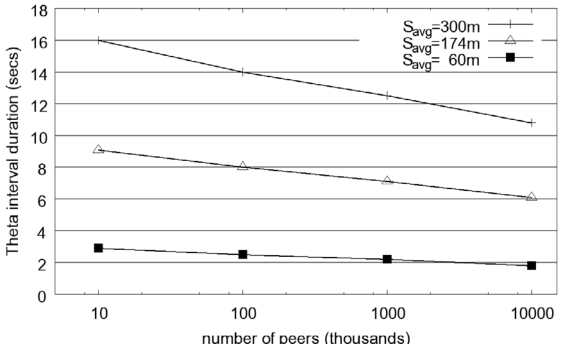

Fig. 8. Duration of the Θ interval in seconds for $f = 1\%$ and different S_{avg} values.

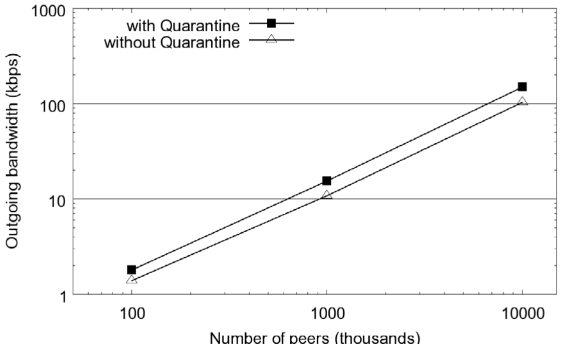

Fig. 9. D1HT bandwidth requirements with and without Quarantine (T_q=10min).

6. Related Work

Rodrigues et al [20] proposed a single hop DHT in a complete different context from ours, as their system was based on dedicated servers arranged on a two level hierarchy, and their main goal was to obtain robustness against malicious network attacks. Besides, their system was not able to guarantee an upper bound on the number of routing failures, the events were reported using a gossip method, and no performance analysis or evaluation was presented.

D1HT assures that a large fraction of the lookups takes just one hop even for very large and dynamic systems. In contrast, a number of systems (e.g. [5, 13, 15, 17]) solve the lookups with a constant number of multiple hops and are not able to ensure an upper bound on the number of routing failures. In addition, those systems differ from D1HT in other important aspects. Kelips[5] maintains routing tables with $O(\sqrt{n})$ IP addresses to solve the lookups with two hops. LH*[13] divides the nodes in client and servers, and solves the lookups with up to three hops. Structured Superpeers[15] implements a hierarchical topology with intrinsic load balance issues to solve lookups with three hops. Beehive[17] is not a DHT by itself, but a replication framework that can be applied to DHTs in order to reduce the number of hops for popular keys.

There is a number of systems, including Chord[24] and SkipNet[6], where each peer uses pointers to nodes (fingers) with 2^i distances (usually $0 \leq i \leq log(N)$), but those pointers are used only to route the lookups in $O(\log(n))$ hops. In contrast, D1HT uses its 2^l pointers solely for event reporting. Besides, those systems were not able to assure an upper bound in the number of routing failures and solve the lookups with multiple hops, while D1HT assures that a high fraction of the lookups takes just one hop. To the best of our knowledge, there is no DHT system proposed so far that uses an event reporting algorithm similar to EDRA.

Accordion[11] also addresses the tradeoff between lookup latency and bandwidth requirements, but its approach is quite different from ours. Accordion implements some very clever adaptation techniques that aim to speed up the lookup performance under pre-defined bandwidth restrictions, but it is not able to enforce a maximum fraction of the routing failures. In contrast, D1HT aims to provide the best lookup performance and adapts to the system dynamics in order to comply with a pre-defined upper bound on the number of routing failures. Besides, D1HT has proven correctness and load balance properties.

Although using a hierarchical approach - in contrast to D1HT pure P2P architecture - the OneHop system [4] is the most similar to ours, as it was the first DHT that was able to assure that a large fraction of the lookups takes only one hop even in dynamic networks. In this paper, we compared this system against D1HT, and showed that D1HT is able to provide superior maintenance load balance and has bandwidth requirements up to one order of magnitude smaller.

7. Conclusion

In this paper, we introduced D1HT, a novel single-hop distributed hash table that is able to 1) assure that a large fraction of the lookups are solved with one hop (e.g. 99%); 2) demand low bandwidth overheads; 3) provide good balance of the maintenance traffic among the peers; and 4) adapt to changes in the system dynamics. We proposed and formally described the Event Detection and Dissemination Algorithm (EDRA) used by D1HT, and proved its correctness and performance properties.

We presented performance analyses showing that D1HT has at least twice and up to one order of magnitude less maintenance bandwidth requirements than those of nodes in previous single-hop DHT. Our analysis also showed that D1HT has reasonable bandwidth demands even for huge systems with dynamics similar to those of popular P2P applications. More specifically, our analysis showed that D1HT requires only 3 kbps of maintenance overhead in huge systems with one million peers and dynamics similar to that of BitTorrent, a widely deployed P2P application. We also presented a Quarantine mechanism that reduces the overhead caused by volatile peers and may help to prevent malicious attacks to the system.

Acknowledgements

The authors would like to thank Ricardo Bianchini for his helpful comments.

References

[1] www.slyck.com/stats.php, Oct 2005.

[2] A. Bellissimo, P. Shenoy, and B. Levine. Exploring the use of BitTorrent as the basis for a large trace repository. Technical Report 04-41, Department of Computer Science, U. of Massachusetts, Jun 2004.

[3] J. Chu, K. Labonte, and B. Levine. Availability and locality measurements of peer-to-peer file systems. In *Proc. of SPIE*, Jul 2002.

[4] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *Proc. of NSDI*, Mar 2004.

[5] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proc. of IPTPS*, Feb 2003.

[6] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *In Proc. of the 4th USITS*, Mar 2003.

[7] R. Huebsch, J. Hellerstein, N. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *International Conference on Very Large Databases*, Sep 2003.

[8] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. of the Symposium on Theory of Computing*, May 1997.

[9] J. Li, J. Stribling, T. Gil, R. Morris, and F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proc. of IPTPS*, 2004.

[10] J. Li, J. Stribling, R. Morris, and M. Frans. A performance vs. cost framework for evaluating DHT design tradeoffs. In *Proc. of INFOCOM*, Mar 2005.

[11] J. Li, J. Stribling, R. Morris, and M. Kaashoek. Bandwidth-efficient management of DHT routing tables. In *Proc. of NSDI*, May 2005.

[12] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proc. of the 21st PODC*, Jul 2002.

[13] W. Litwin, M. Neimat, and D. Schneider. LH* - a scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.

[14] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. of the 21st PODC*, Jul 2002.

[15] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured Superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proc. of the 3rd Workshop on Internet Applications*, Jun 2003.

[16] NIST. Secure Hash Standard (SHS). *FIPS Publication 180-1*, Apr 1995.

[17] V. Ramasubramanian and E. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Proc. of NSDI*, Mar 2004.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of SIGCOMM*, 2001.

[19] R. Rodrigues and C. Blake. When multi-hop peer-to-peer routing matters. In *Proc. of IPTPS*, Feb 2004.

[20] R. Rodrigues, B. Liskov, and L. Shrira. The design of a robust peer-to-peer system. In *Proc. of the 10th ACM SIGOPS European Workshop*, Sep 2002.

[21] A. Rowstron and P. Druschel. Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, Nov 2001.

[22] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of SPIE/ACM MMCN*, Jan 2002.

[23] F. Schintke, T. Schutt, and A. Reinefeld. A framework for self-optimizing Grids using P2P components. In *Proc. of the 14th IEEE DEXA*, 2003.

[24] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, Feb 2003.

[25] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *Journal on Selected Areas in Communications*, Jan 2004.