D1HT: A Distributed One Hop Hash Table

Luiz R. Monnerat•,⋆ and Claudio L. Amorim⋆

⋆COPPE - Computer and Systems Engineering
Federal University of Rio de Janeiro

•TI/TI-E&P/STEP
PETROBRAS

{monnerat,claudio}@cos.ufrj.br

Technical Report ES-676/05 - May 2005, COPPE/UFRJ

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 P2P systems. We formally define the algorithm we use to detect and notify any membership change in the system, prove its correctness and performance properties. Our analysis show that D1HT has reasonable maintenance bandwidth requirements even for multi-million node systems while presenting up to one order of magnitude less bandwidth overhead than previous single hop DHT.

1 Introduction

Peer-to-peer (P2P) 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 been proposed as a substrate for a variety of distributed and P2P applications, including grid services [29], application-layer multicast [3, 23, 30], name resolution [5], distributed storage systems [14, 27], backup facilities [4], web caching [10], DDoS attack protection [13], spam filtering [33], and databases [9].

DHT systems implement a hash-table-like lookup facility [1] where the keys (information) are distributed among the participant nodes (peers). In order to route a given lookup from its origin to the peer that is responsible for the target key, DHTs implement overlay networks with routing information stored in each peer (routing tables). Unless each routing table is large enough to hold the IP addresses of all participant peers, the routing of a single lookup is likely to require multiple hops, i.e., the lookup should hop through a number of peers 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 peers join and leave the system, specially in very dynamic systems (i.e., systems with a high frequency of peer 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 addition, we believe in a steady increase in the number of users willing to pay for an extra few Kbps of bandwidth in order to get the desired information as fast as possible, provided that the resulting bandwidth's costs are reasonable and compatible with the expected improvements in lookup's performance. In contrast, most DHT systems that have been proposed so far solve the lookups with multiple hops in an attempt to minimize the maintenance traffic, e.g. [11, 17, 18, 20, 22, 26, 31, 32], as it was generally accepted that single hop DHTs could lead to prohibitively high maintenance traffic for dynamic systems. However, recent results[15] 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[24], which indicated that low-overhead multi-hop DHTs are required only for vast and very dynamic peer networks. On the other hand, there is only one proposed DHT system that ensures that a large fraction of the lookups are really solved with only one hop, but that system imposes high levels of load imbalance and bandwidth overheads in order to maintain the routing tables up to date[6].

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 peers; 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, per-

formance, and load balance properties. Our analytical results show that a D1HT peer has at least twice and up to one order of magnitude less maintenance bandwidth requirements than that of a previous single-hop DHT [6]. Our results also show that D1HT is able to support multi-million peer systems with dynamics similar to those of Gnutella and BitTorrent, with reasonable maintenance bandwidth demands. For instance, a huge one-million D1HT system, with dynamics similar to BitTorrent[2], would require only 3 kbits/sec of duplex maintenance traffic to assure that 99% of the lookups are solved with just one message. For a 10K peer D1HT system that requirements will drop to 0.1 kbits/sec, which is negligible for most peer connections.

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 and maps items (or keys) to peers based on *consistent hashing* [12], where both the peers and the keys are hashed to integer identifiers (IDs) in the same ID space $[0..N]$, $N \gg n$. Typically a key ID k is the cryptographic hash SHA-1 of the key value, the ID p of a peer is 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.

D1HT uses a ring topology based on Chord[31], 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.

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 at least $\log_2(n)$ peers in the system. Peer p will then *ping* each one of those peers and choose the nearest one to ask for the routing table (in fact p may choose to get the routing table from more than one peer in order to distribute the load). To track peer crashing, each peer p is in charge of detecting if its predecessor p_{pred} has left the system.

Although using a ring topology, D1HT significantly differs from Chord in respect to the lookup algorithm. Since each peer has the IP addresses of all peers in the system, any lookup in a D1HT system is trivially solved with one hop, provided that the local routing table is up to date. Note that if a peer p does not acknowledge an event (another peer that has joined or left 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[1]. In the latter case, a time out will occur and p will reissue the lookup to the successor of the original target. In both cases, the lookup will 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 incurring in high bandwidth overheads and load imbalance. This algorithm will be presented in section 3.

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 clear. The i_{th} successor of a peer 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 $i > n - 1$, $SUCC(p, i) = SUCC(p, i - n)$. 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 $SUCC(p, i-1))$, for $i > 0$. The peer distance (number of peers) between two peers is given by the function $D(p_i, p_j)$, where $D(p_i, p_i) = 0$ and $D(p_i, p_j) = D(p_i, PRED(p_j, 1))+1$ for $p_i \neq p_j$. Note that if $p_i \neq p_j$ then $D(p_i, p_j) + D(p_j, p_i) = n$. As in [17], for any two peers p_i and p_j the $Stretch(p_i, p_j) = \{\forall p \in \mathbb{D} \mid p = SUCC(p_i, i) = PRED(p_j, j) \quad \wedge \quad i + j = D(p_i, p_j)\}$. Note that $Stretch(p, SUCC(p, n - 1)) = \mathbb{D}$.

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 by all peers in the system in a

[1] In fact, as each key is replicated on $\log_2(n)$ succeeding peers, the target peer will probably still hold a copy of the key, and it will then be able to answer the lookup without the need to forward it.

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

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 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, D1HT uses an 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 stale routing table entries.

In this section, we will define EDRA by means of a number of rules, prove its correction, and its optimal behavior in terms of bandwidth use and incoming traffic load balance. We will also show that EDRA has good outgoing traffic load balance for dynamic systems provided that the events are evenly spread along the ring.

3.1 Event Dissemination

The rules below define how EDRA detects and disseminates the events in a D1HT system:

Rule 1: Every peer will send at least one and up to ρ messages at the end of each Θ seconds time 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: The messages will only contain events acknowledged during the ending Θ interval. The sender will include in the message with $TTL = l$ all events acknowledged with $TTL > l$ during the ending Θ interval. Events acknowledged with $TTL = 0$ will not be forwarded through 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 a peer p_j, peer p_i will discharge all events related to any peer in $Strecht(p_i, p_j)$.

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 below. For this 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.

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 last peer to acknowledge ε will be $SUCC(p, 2^l - 1)$ after $T_{tot}^{'} = l \cdot \Theta$ secs. The average time $T_{avg}^{'}$ for a peer in $Stretch(p, SUCC(p, 2^l - 1))$ to acknowledge ε will be $l \cdot \Theta/2$ secs.*

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 peer $SUCC(p, 1)$. As this message should be sent at the end of the current Θ interval, $SUCC(p, 1)$ will acknowledge ε after $T_{tot}^{'} = \Theta$ secs, making the average time for peers p and $SUCC(p, 1)$ to be $T_{avg}^{'} = \Theta/2$. 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 $p_k = SUCC(p, 2^k)$, $0 \leq k \leq l - 1$, 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 $Stretch(p_k, SUCC(p_k, 2^k - 1))$ will acknowledge ε exactly once with $T_{avg}^{'} = k \cdot \Theta/2$ secs, and the last peer in each $Stretch(p_k, SUCC(p_k, 2^k - 1)$ to acknowledge ε will be $SUCC(p_k, 2^k - 1)$ in $k \cdot \Theta$ secs. Accounting for all $l-1$ acknowledgments made by the peers p_k, we will then have that ε will be acknowledged once by all peers in $Stretch(p, 2^l - 1)$. As none of those peers will

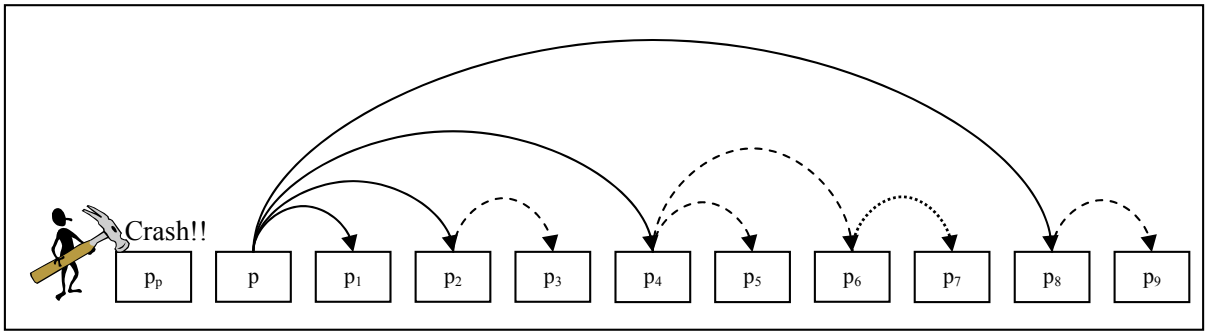

Fig. 1: This figure shows a D1HT system with 11 peers ($\rho = 4$), where peer p_p crashes and this event is detected and reported by its successor p. The peers are represented in a line instead of a ring to facilitate the presentation. In the figure, peers p_i are such that $p_i = SUCC(p, i)$.

forward ε to a peer outside this range, ε will not be acknowledged by any other peers in the system. Besides, as for $k = l - 1$ we have that $p_{l-1} = SUCC(p, 2^{l-1})$ will be the one to receive ε from p with the highest TTL, and so the last peer to acknowledge ε will be peer $SUCC(p_{l-1}, 2^{l-2}) = SUCC(p, 2^l - 1)$, and it will happen $(l - 1) \cdot \Theta$ secs after p_{l-1} had received ε, i.e., $l \cdot \Theta$ secs after p had acknowledge ε, and so $T'_{tot} = l \cdot \Theta$ secs. The average time for the peers in each $Stretch(p_k, SUCC(p_k, 2^k - 1))$, $0 \leq k \leq l - 1$, to acknowledge ε will be $k \cdot \Theta/2$ secs after the respective peer p_k had acknowledged it, which will lead to $T'_{avg} = l \cdot \Theta/2$ for peers in $Stretch(p, 2^l - 1)$. ■

Applying theorem 3.1 and the EDRA rules to a peer p join (or leave) that was acknowledged by its successor p_s, we will have that this event will be further acknowledged exactly once by all peers in $Stretch(p_s, SUCC(p_s, n - 1)) = \mathbb{D}$. Besides, the average and maximum acknowledge times will be respectively $\rho \cdot \Theta/2$ secs and $\rho \cdot \Theta$ secs after p_s had acknowledged the event. Note that as usually $n < 2^\rho$, rule 8 is necessary in order to avoid the event to be forwarded to $Stretch(p_s, SUCC(p_s, 2^\rho))$ instead of $Stretch(p_s, SUCC(p_s, n - 1))$.

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 then 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). Peer p_7 will then be the last peer to acknowledge ε, $3 \cdot \Theta$ secs after p had acknowledged it, which is less than the maximum as defined by theorem 3.1. 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 enable EDRA to make 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 bandwidth of outgoing and incoming traffics 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 single events, 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 serious problem as our target is large and dynamic systems, in which several events happen at every second, so that we should not be 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 num-

ber of messages and outgoing bandwidth demand will rely on the random distribution properties of the hashing function it uses (typically SHA-1). The chosen hash function is expected to randomly distribute the peers IDs in order to obtain a good intermix of stable and volatile peers in the ring[3], which in turn should lead to an even distribution of the events along the ring. Note that even if a certain *lucky* peer p_{lucky} happens to have a very stable predecessor p_{stable} (which could allow p_{lucky} to have an outgoing traffic load well below the average) the systems dynamics should sooner or later bring a new volatile peer $p_{volatile}$ to join the system in between p_{stable} and p_{lucky}, which brings the load on p_{lucky} nearer to the average.

Here and thereafter, we will then consider that the chosen hash function provides a peer's ID distribution that will allow for an even balance of the outgoing traffic and messages that are sent and received. More specifically, assuming that we have an evenly distributed rate of r events per second in the system, the average amounts of incoming and outgoing traffic per peer will be (including message acknowledgments):

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

where N_{msgs} is the average number of messages a peer sends (and receives) per Θ interval, m is the average number of bytes necessary to describe an event, and v is the byte 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 each peer will generate two events per session (one join and one leave), the event rate can be calculated as follows:

$$r = 2 \cdot n/S_{avg} \qquad (3.2)$$

In that way, r will vary according to the system size. A more realistic assumption would consider that the average session length itself varies with time to address dynamics where, for example, the average session length during the day is different from that observed at night. In section 4.2, we will show that EDRA can adapt to variations in r in order to assure a maximum fraction of routing failures, even when the system dynamics change over time, and in section 5 we will analyze the D1HT overheads for different values of r.

3.4 Number of Messages

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

Theorem 3.2. *The set of peers S for which a generic peer p acknowledges events with $TTL > l$ 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 $TTL \geq \rho = l$. 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, which assures that $S1$ and $S2$ are disjoints and so $|St| = |S1| + |S2| = 2^{\rho-(l+1)} + 2^{\rho-(l+1)} = 2^{\rho-l}$. ■

Rule 4 states that a generic peer p will only send a message with $TTL = l > 0$ if it acknowledges at least one event with $TTL > l$. Based on theorem 3.2 we can then say that p will only send a message with $TTL = l > 0$ if at least one of set of $2^{\rho-l}$ peers suffers an event. As the probability of a specific peer to suffer an event in a Θ interval is $\Theta \cdot r/n$, the probability $P(l)$ of a generic peer to send a message with $TTL = l > 0$ at the end of each Θ interval is given by the formula:

$$P(l) = 1 - (1 - \Theta \cdot r/n)^k, \text{ where } k = 2^{\rho-l} \qquad (3.3)$$

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

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

Equations 3.1, 3.3, and 3.4 allows us to calculate the average amount of incoming and outgoing traffic per peer based on the rate of events r, the system size n, and the duration of the Θ interval. This result will be used in the next sections to tune EDRA in order to assure an upper bound on the fraction of routing failures.

[3]Note that maintenance traffic load balance does not rely on the ability of consistent hashing to evenly distribute the keys among peers.

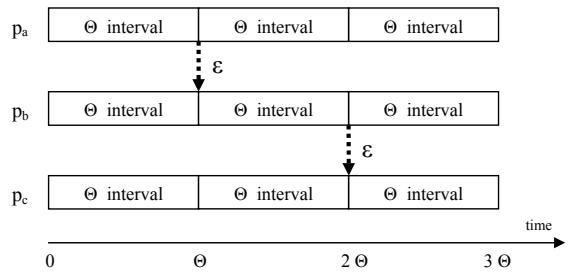

Fig. 2: Propagation of an event ε with Θ intervals synchronized and in the absence of message delays.

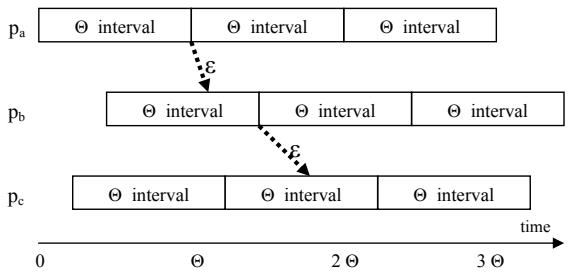

Fig. 3: Propagation of an event ε with asynchronous Θ intervals and message delays.

4 Practical Aspects

Up to now we have considered theoretical situations where there were no message delays and all Θ intervals were synchronous. In this section, we will show how EDRA performs in the presence of those problems, and how it can be tuned in order to comply to a user defined upper bound to the fraction of the routing failures.

4.1 Message Delays and Asynchronous Intervals

In theorem 3.1, we did not consider the effects of message delays and asynchronous Θ intervals. Although those problems do not affect EDRA's correctness, they do interfere on its performance, so we will turn to them in this section.

Figures 2 and 3 show timelines that represent 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, with the dotted arrows being used to indicate the messages reporting ε. In this hypothetical situation, each peer will add exactly Θ secs on the propagation time for ε, leading to $T_{tot}^{'} = \rho \cdot \Theta$ secs and $T_{avg}^{'} = \rho \cdot \Theta/2$ secs, as already 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). In this case, on average each message will take δ_{avg} 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 values $T_{tot}^{''} = \rho \cdot (\Theta + 2 \cdot \delta_{avg})/2$ secs and $T_{avg}^{''} = \rho \cdot (\Theta + 2 \cdot \delta_{avg})/4$ secs. Note that the above values of $T_{tot}^{''}$ and $T_{avg}^{''}$ have not yet considered the time to detect the event. Considering that a peer will take up to T_{detect} secs to detect an event in its predecessor, the average and total acknowledge times will be respectively $T_{detect} + T_{tot}^{''}$ secs and $T_{detect} + T_{avg}^{''}$ secs after the event had happened.

From now on, we will consider that $T_{detect} = 2\Theta$, which means that 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 has left the system, p will report p_p failure at the end of the next Θ interval. So we can calculate the expected average and total acknowledge times of an event after it had happened:

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

$$T_{tot} = 2 \cdot \Theta + \rho \cdot (\Theta + 2 \cdot \delta_{avg})/2 \; secs \qquad (4.2)$$

4.2 Tuning EDRA

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 that EDRA assures that the hops will fail with probability f at most. Assuming that the lookup targets are evenly spread along the ring, 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 target average fraction of routing failures f, it suffices[4] to assure that the average fraction of stale routing table entries is kept below f.

As each event will take on average T_{avg} seconds to be acknowledged by a D1HT system, 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., $T_{avg} \cdot r$, where once again r is the event rate (events per seconds). This implies that to accomplish a given f we should satisfy the inequality $T_{avg} \cdot r/n \leq f$. With the help of equation 4.1, we have the following inequation:

$$\Theta \leq \frac{f \cdot n/r - \rho \cdot \delta_{avg}/2}{2 + \rho/4} \; seconds \qquad (4.3)$$

[4]In fact it is a 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	$[10^5, 10^6]$	$[10^5, 10^6]$, $\underline{[10^4, 10^7]}$	Number of nodes in the system. $n = \|\mathbb{D}\|$
S_{avg}	174	$\underline{60}$, 174, $\underline{300}$, $\underline{780}$	Average session duration in minutes.
v	20	20	Overhead per message (headers, etc.), in bytes.
m	10	10	Number of bytes necessary to describe an event
f	1%	1%, $\underline{5\%}$, $\underline{10\%}$	Maximum acceptable fraction of routing failures.
δ_{avg}	-	0.2	Average message delay in seconds

Table 1: Parameters we used in our analysis. The underlined values were used only in section 5.3.

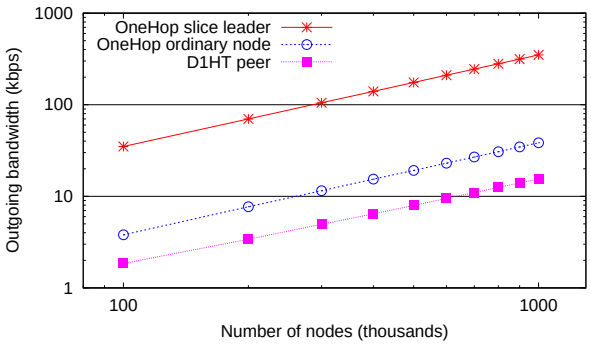

Fig. 4: Outgoing bandwidth demands in kbits/sec for OneHop (ordinary node and slice leader) and a D1HT peer.

In that way, for a given system with n peers, an event rate r, and an average message delay δ_{avg}, we can calculate the Θ interval duration necessary to satisfy a given upper bound on the number of routing failures (f).

As we have already pointed out in section 3.3, it is not reasonable to expect r to be constant, and equation 4.3 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. Note that, based on the results of theorem 3.1, we should expect that any change in the system dynamics will take up to T_{tot} seconds to be observed by all peers, which may lead to some peers to use different values for Θ for this short period of time. This is not a major problem however, as theorem 3.1 shows that EDRA correctness does not depend on the Θ value.

5 Analysis

In this section, we will quantify the amount of bandwidth that is necessary to maintain the routing tables in D1HT, and compare those results with the one hop DHT (OneHop) results as presented in [6]. We will briefly describe the OneHop system in section 5.1, and in section 5.2 we will compare it against D1HT. In section 5.3 we will analyze D1HT sensitivity to variations in some system parameters.

5.1 The OneHop DHT

The OneHop system[6] was the first proposed DHT to assure that a high fraction of the lookups are solved in the first attempt. As D1HT, OneHop also maps keys to nodes based on consistent hashing and treats the ID space as a ring. The lookups take just one hop, as every node has the IP addresses of every other node in the system. In contrast to D1HT, 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*. Typically, for a 10^5 node system, each unit will have 400 nodes, and each slice will have 5 units (2.000 nodes).

Each unit leader is in charge of collecting information about all events in its unit and forwarding them periodically to its slice leader. The slice leader groups the events from its various units and periodically sends messages to the other slices leaders reporting the events that happened in its own slice. Each slice leader will then acknowledge the events that happened in every other slice, and will periodically send one message to each of its unit leaders reporting those events. Finally, each unit leader propagates the information it received to the nodes in its own unit. More details about the OneHop DHT can be found in [6].

5.2 Comparative Analysis

In this section, we will analyze the outgoing bandwidth demands of D1HT and OneHop. We will compare the demands of a D1HT peer against the best (ordinary nodes) and worst (slice leaders) OneHop cases. The parameters used in this section are shown in table 1 (the D1HT's underlined values will be used only in section 5.3). For both systems we used the same values for S_{avg}, v, m and f, which were taken from [6], where the average session duration of 174 minutes (2.9 hours) was based on a study of Gnutella behavior[28]. The event rates were calculated with formula 3.2 based on S_{avg}. While the OneHop results do not take message delays into account, for D1HT we used 280 ms for the average message delay, which is quite conservative in relation to the results presented in [28], where 80% of the measured Gnutella latencies were below 280 ms.

We limited our comparison to system sizes (n) in the

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

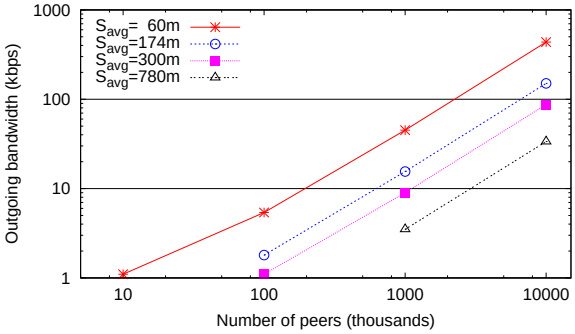

Fig. 5: D1HT peer bandwidth demands in kbits/sec for $f = 1\%$ and different values of average session length (S_{avg}).

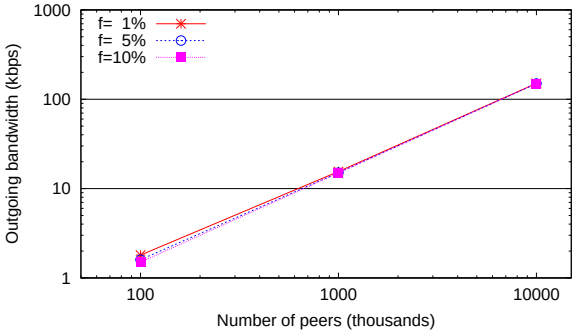

Fig. 7: D1HT peer bandwidth demands in kbits/sec for $S_{avg} = 2.9$ hours and different values of f.

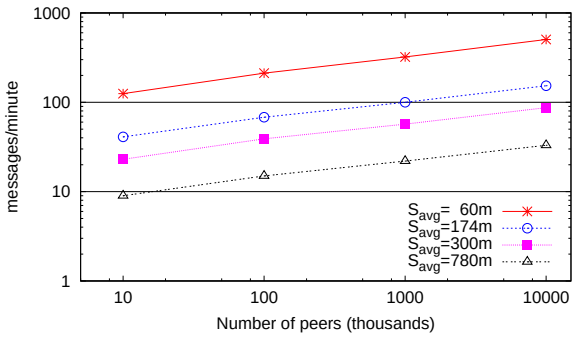

Fig. 6: Average number of messages sent by a D1HT peer for $f = 1\%$ and different values of average session length (S_{avg}).

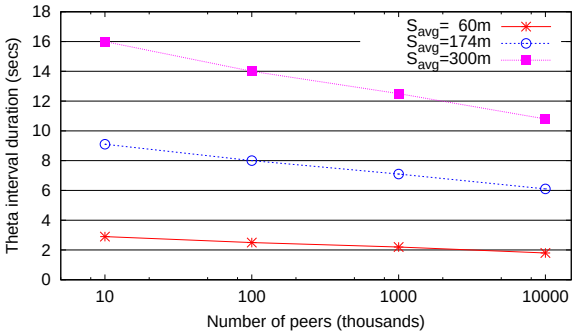

Fig. 8: Duration of the Θ interval in seconds for $f = 1\%$ and different values of average session length S_{avg}.

range $[10^5, 10^6]$, since it was the interval with bandwidth requirements as reported in [6]. The outgoing bandwidth requirements reported for ordinary nodes and slice leaders were respectively 3.84 kbps and 35 kbps for a 10^5 nodes system, raising linearly up to 38 kbps and 350 kbps for $n = 10^6$. Those results are plotted in Figure 4 (both axes are logarithmic), as well as the requirements for a D1HT peer based on formula 3.1.

Figure 4 shows that the outgoing bandwidth requirements for an OneHop ordinary node and a slice leader are twice and one order of magnitude higher, respectively, than those from a D1HT peer. For example, for a 10^5 node system 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 15 kbps, 38 kbps and 350 kbps, in that order for $n = 10^6$.

5.3 Parameter Analysis

In this section, we will study the D1HT sensitivity to variations in some system parameters according to the underlined values in Table 1, and extending the range of system sizes to $[10^4, 10^7]$. We will also analyze the values of Θ that are necessary to satisfy $f = 1\%$.

In previous section, we showed both D1HT and OneHop requirements for systems with 2.9 hours of average session duration, as it was the original value used in [6] based on the Gnutella behavior. However, a recent work [2] indicated that other systems might have much less dynamics, as the measured average session length for BitTorrent was 13 hours (780 minutes). 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 show in Figure 5 the maintenance bandwidth requirements for a D1HT peer in systems with average sessions of 60, 174, 300, and 780 minutes, where session durations of 174 and 780 minutes are representative of Gnutella and BitTorrent, respectively [28, 2]. We omitted some results since they were below 1 kbps and the axes are logarithmic. The figure shows that the bandwidth requirements are roughly linearly dependent on both the logarithm of the system size and the inverse of the average session duration S_{avg}. For example, the requirements for a D1HT peer in a system 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 $n = 10^6$.

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

Figure 7 shows D1HT bandwidth requirements for dif-

ferent values of f. The values for $n = 10^4$ are less than 1 kbps and are not shown in the graph as the axis is logarithmic. We notice small variations in bandwidth demands for different values of f. For instance, with $n = 10^5$ the requirements for $f = 1\%$, 5%, and 10% are 16 kbps, 15 kbps, and 15 kbps, respectively. Those results indicate that it is not the case to increase f in order to reduce the bandwidth requirements. In fact, as f is solely used to calculate Θ (equation 4.3), the results also show that the duration of the Θ interval has little effect on the bandwidth demands.

We will now analyze the values of Θ that are required for D1HT to comply with $f = 1\%$. Note that D1HT should not require very small values for Θ, e.g., less than one second, as it could allow the costs of message delays to interfere in its proven properties. Figure 8 shows the duration of the Θ interval that was necessary to achieve $f = 1\%$ for some values of S_{avg} (the Θ values for $S_{avg} = 780$ minutes were above 30 secs and are not plotted in the figure). We see that values of Θ well above 1 sec are enough to satisfy $f = 1\%$ even for systems with 10 million peers and one hour average session length.

6 Related Work

Rodrigues et al[25] also proposed a single hop DHT, but their system is quite different from ours. Their main goal was to obtain robustness against malicious network attacks with a system that was built with dedicated servers arranged in a two level hierarchy. The option for a single hop system was in fact a consequence of the expected low event rate, which in turn was due to the use of dedicated servers. Besides, their system was not able to guarantee an upper bound on the number of routing failures and the events were reported using a gossip method. In contrast, D1HT aims to provide a pure P2P single hop DHT that could be built from non dedicated and volatile client peers and yet be able to support single hop lookups in very dynamic environments.

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 recently proposed systems[7, 19, 21] 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[7] uses a gossip mechanism that takes more than one hour to notify an event to all peers in a 10^6 peer system, while D1HT takes less than one minute to do the same[6]. Structured Superpeers[19] implements a hierarchical topology with intrinsic load balance issues, while D1HT uses a well balanced pure P2P approach. Beehive[21] 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[31] and SkipNet[8], 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, 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[16] also addresses the tradeoff between lookup latency and bandwidth requirements, but its approach is quite different from ours. Accordion implements some very clever adaptation technics that aim to provide the best possible lookup performance under pre-defined bandwidth restrictions, but it is not able to enforce a maximum fraction of the routing failures. D1HT also aims to provide the best lookup performance, but it seeks to minimize bandwidth overhead while complying with a pre-defined upper bound on the number of routing failures. Besides, D1HT has proven correctness and load balance properties. Accordion was able to outperform OneHop while consuming about 20 bytes/sec/node [16], and, according to our analysis, D1HT would demand 8 bytes/sec/peer maintenance traffic with the same set of parameters ($f = 5\%$, $m = 4$ bytes, $S_{avg} = 10,000$ secs, $\delta_{avg} = 179$ ms and $n = 3,000$).

Although using a hierarchical approach - in contrast to D1HT pure P2P architecture - the OneHop system as presented in [6] is the most similar to ours, as it was the first DHT that was able to assure that a large fraction of the lookups succeds in the first attempt. 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 formally described the Event Detection and Dissemination Algorithm (EDRA) used by D1HT and proved its correctness and performance properties.

We presented performance analysis showing that D1HT has at least twice and up to one order of mag-

[6] In systems with average session durations of 2.9 hours.

nitude less maintenance bandwidth requirements than previous single-hop DHT with similar characteristics. Our analysis also showed that D1HT has reasonable bandwidth demands even for huge and dynamic systems. Specifically, D1HT required only 3 kbps of maintenance overhead in order to ensure that 99% of the lookups were successful in the first attempt for a million peer system with dynamics similar to that of BitTorrent.

Acknowledgements

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

References

[1] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, 2003.

[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, University of Massachusetts, June 2004.

[3] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.

[4] L. Cox and B. Noble. Pastiche: Making backup cheap and easy. In *Proc. of the Fifth OSDI*, December 2002.

[5] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using Chord. In *Proc. of the 1st IPTPS*, March 2002.

[6] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *Proc of the 1st NSDI*, March 2004.

[7] 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 the 2nd IPTPS*, February 2003.

[8] 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 Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[9] 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.

[10] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *21st PODC*, 2002.

[11] M. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proc of the 2nd IPTPS*, Feb 2003.

[12] 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 ACM Symposium on Theory of Computing*, May 1997.

[13] A. Keromytis, V. Misra, and D. Rubenstein. SOS: An architecture for mitigating DDoS attacks. *IEEE Journal on Selected Areas in Communications (JSAC)*, Jan 2004.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of ACM ASPLOS*, November 2000.

[15] J. Li, J. Stribling, R. Morris, and M. Frans. A performance vs. cost framework for evaluating DHT design tradeoffs. In *Proc. of the 24th Infocom*, March 2005.

[16] J. Li, J. Stribling, R. Morris, and M. Kaashoek. Bandwidth-efficient management of DHT routing tables. In *Proc. of the 2nd NSDI*, May 2005.

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

[18] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of the 1st IPTPS*, March 2002.

[19] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured Superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proc. of the Third IEEE Workshop on Internet Applications (WIAPP)*, June 2003.

[20] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the ACM SPAA*, June 1997.

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

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

[23] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC*, 2001.

[24] R. Rodrigues and C. Blake. When multi-hop peer-to-peer routing matters. In *Proc of the 3rd IPTPS*, February 2004.

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

[26] A. Rowstron and P. Druschel. Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In *Proc. of the International Conference on Distributed Systems Platforms*, Nov 2001.

[27] A. Rowstron and P. Druschel. Storage management and caching in PAST, A large-scale, persistent peer-to-peer storage utility. In *Proc. of the 18th SOSP*, Oct 2001.

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

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

[30] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. of the ACM SIGCOMM Conference*, August 2002.

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

[32] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

[33] F. Zhou, L. Zhuang, B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz. Approximate object location and spam filtering on peer-to-peer systems. In *Proc. of Middleware*, 2003.