An effective single-hop distributed hash table with high lookup performance and low traffic overhead

Luiz Monnerat[•],[⋆] and Claudio L. Amorim[⋆]

[•]TIC/TIC-E&P
PETROBRAS
Rio de Janeiro, Brazil

[⋆]COPPE - Computer and Systems Engineering
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil

{monnerat,amorim}@cos.ufrj.br

[•] Corresponding Author: Address: Av. Chile 65 - 1501, Rio de Janeiro, RJ, Brazil, 20.031-912
e-mail: monnerat@cos.ufrj.br

Index Terms—**Distributed hash tables; overlay networks; P2P; distributed systems; performance**

Abstract—**Distributed Hash Tables (DHTs) have been used in several applications, but most DHTs have opted to solve lookups with multiple hops, to minimize bandwidth costs while sacrificing lookup latency. This paper presents D1HT, which has a peer-to-peer and self-organizing architecture and is the first DHT that maximizes lookup performance with reasonable maintenance traffic, and a Quarantine mechanism to reduce overheads caused by volatile peers. We implemented both D1HT and a prominent single-hop DHT, and we performed the broadest and most representative DHT experimental comparison to date, followed by complementary analytical studies. In comparison with current single-hop DHTs, our results showed that D1HT consistently had the lowest bandwidth requirements, with typical reductions of up to one order of magnitude, and that D1HT could be used even in popular Internet applications with millions of users. In addition, we ran the first latency experiments comparing DHTs to directory servers, which revealed that D1HT can achieve latencies equivalent to or better than a directory server, and confirmed its greater scalability properties. Overall, our extensive set of results allowed us to conclude that D1HT is the first DHT to provide a very effective solution for a broad range of environments, from large-scale corporate datacenters to widely deployed Internet applications.**

I. Introduction

Distributed hash tables (DHTs) are a highly scalable solution for efficiently locating information in large-scale distributed systems; thus they have been used in a wide range of applications, from Internet games to databases. While most DHTs incur in high latencies, recent results showed that DHTs can also be applied in significant classes of applications with performance constraints, such as Internet Service Providers (ISPs), as long as they guarantee low enough latency to access

information. Specifically, the development of a proprietary low-latency DHT was critical to the performance of the Amazons Dynamo system [12], where scalability, self-organization and robustness were fundamental to supporting a production system over thousands of error-prone nodes, whereas the use of central directories could lead to several problems [28]. However, the DHT implemented in Dynamo does not support open environments, has high levels of overhead and, according to its authors, is unable to scale to very large systems, besides being very application specific. In addition, recent trends in High Performance Computing (HPC) and ISP datacenters indicate significant increases in the system sizes [4], [17], including a huge demand from cloud computing [3], [7], which will challenge the scalability and fault tolerance of client/servers solutions. In fact, to support a wide span of large-scale distributed applications, new self-organizing DHTs with greater levels of scalability, performance and efficiency are required in order to be used as a commodity substrate for environments ranging from corporate datacenters to popular Internet applications.

The information stored in a DHT is located through `lookup` requests, which are solved with the use of *routing tables* stored on all participant peers. As peers can freely enter and leave the network, DHTs typically use maintenance messages to keep the routing tables up to date. However, maintenance messages increase the DHT's network traffic, which contributes adversely to both the lookup latency and network bandwidth overheads. Overall, the size of routing tables is a critical issue in a DHT system and poses a classic latency vs. bandwidth tradeoff. Concretely, large routing tables allow faster lookups because peers will have more routing options, but they increase the bandwidth overheads due to higher maintenance traffic.

In this regard, the first DHT proposals (e.g., [26], [27],

[38], [44], [47], [50]) opted to use small routing tables in such a way that each lookup takes $O(\log(n))$ hops to be solved (n is the system size), aiming to save bandwidth to the detriment of latency and thus compromising the use of such *multi-hop* DHTs for performance sensitive applications. However, as similar tradeoffs between latency and bandwidth occur across several technologies, the latency restrictions tend to be more critical in the long term, as it has already been shown that 'over time bandwidth typically improves by more than the square of the latency reductions' [35]. From this perspective, a number of *single-hop* DHTs have been proposed recently (e.g., [14], [30], [48]), which are able to provide low latency access to information because each peer maintains a full routing table. Therefore, the lookup performance achieved by these single-hop DHTs should allow their use even in latency-sensitive environments where multi-hop DHTs cannot satisfy the latency constraints. Besides, it has been shown that, for systems with high lookup rates, single-hop DHTs may in fact reduce the *total* bandwidth consumption, since each lookup in a multi-hop DHT typically consumes $O(\log(n))$ more bandwidth than a single-hop lookup, and this extra lookup overhead may offset the routing table maintenance traffic [42], [48]. Nevertheless, most single-hop DHTs still incur high bandwidth overheads, have high levels of load imbalance, or are unable to support dynamic environments.

With these problems in mind, this work provides several main contributions that will improve the understanding and use of single-hop DHTs in a wide range of distributed systems. We present D1HT, the first highly effective single-hop DHT combining low bandwidth overheads and good load balance even in dynamic environments, while being able to efficiently adapt to changes in the system behavior using a self-organizing and pure P2P approach. We will also present a Quarantine mechanism that can reduce the system overheads caused by volatile nodes in P2P systems.

To quantify the latencies and overheads of single-hop DHTs, we implemented D1HT and 1h-Calot [48] from scratch and evaluated both single-hop DHTs with up to 4,000 peers and 2,000 physical nodes in two radically different environments (an HPC datacenter and a worldwide dispersed network) under distinct churn rates. Those experiments provided a number of very important results, as they validated the analyses for both DHTs, confirmed their low latency characteristics, and showed that D1HT consistently has less bandwidth requirements than 1h-Calot. Besides, our experiments also showed that D1HT has negligible CPU and memory overheads that allow its use even in heavily loaded nodes, as it used less than 0.1% of the available CPU cycles and very small memory to store the routing tables, even under a high rate of concurrent peer joins and leaves.

Based on the validation of the D1HT and 1h-Calot analyses, we further performed an analytical comparison among D1HT, 1h-Calot and OneHop [14] for system sizes of up to 10 million peers. Our results revealed that D1HT consistently had the lowest maintenance overheads, with reductions of up to one order of magnitude in relation to both OneHop and 1h-Calot. Moreover, these results also showed that D1HT is able to support vast distributed environments with dynamics similar to

those of widely deployed P2P applications, such as BitTorrent, Gnutella and KAD, with reasonable maintenance bandwidth demands. Overall, D1HTs superior results are due to its novel mechanism that groups membership changes for propagation without sacrificing latency using a pure P2P topology. This mechanism was based on a theorem that will be presented in this paper, which allows each peer in a D1HT system to independently and dynamically adjust the duration of the buffering period, while assuring low latency lookups.

While scalable performance has been a fundamental argument in favor of DHTs over central directory servers, we are not aware of any published experiments demonstrating it. To fill in this gap, we performed the first experimental latency comparison among three DHTs and a directory server, using up to 4,000 peers. These experiments demonstrated the superior single-hop DHT scalability properties and provided us with other important results that will be presented in this work.

Except from our preliminary D1HT experiments [31], all previous DHT comparative evaluations with real implementations have used a few hundred *physical* nodes at most and have been restricted to a single environment (e.g., [14], [39], [50]). Thus, the evaluation presented in this paper, which used up to 4,000 peers in two radically distinct environments, can be regarded as the most representative experimental DHT comparison to date, and the first to compare the latencies provided by distinct DHTs and a directory server.

Finally, our extensive set of experimental and analytical results allowed us to conclude that D1HT consistently has the lowest overheads among the single-hop DHTs introduced so far, besides being more scalable than directory servers, and that D1HT can potentially be used in a multitude of environments ranging from HPC and ISP datacenters to applications widely deployed over the Internet.

The rest of this paper is organized as follows. The next two sections discuss related work and present the D1HT system design, and in Section IV we present the event dissemination mechanism used by D1HT. In Sections V and VI, we present Quarantine and our D1HT implementation. Sections VII and VIII present our experimental and analytical results, which are discussed in Section IX. We then conclude the paper.

II. RELATED WORK

In recent years, DHTs and P2P systems have been subjects of intense research. In particular, the design of a DHT that supports large-scale networks is a very difficult problem on its own, which poses specific challenges of scalability and efficiency. Therefore, in this work, we focus on single-hop DHTs whose event dissemination mechanisms aim at large and dynamic environments. In practice, besides D1HT, the only two other single-hop DHTs that support dynamic networks are the OneHop [14] and 1h-Calot [48] systems, both of which differ from D1HT in the following fundamental ways.

The 1h-Calot [48] DHT, which was introduced concurrently with D1HT [30], also uses a pure P2P topology, though they differ in significant ways. First, 1h-Calot uses event[1] propaga-

---

[1]From now on we will refer to peer joins and leaves simply as *events*.

tion trees based on peer ID intervals, while D1HT constructs its dissemination trees using message TTLs. Second, 1h-Calot uses explicit heartbeat messages to detect node failures, while D1HT relies on the maintenance messages. Third and most important, 1h-Calot peers are unable to effectively buffer events and, at the same time, ensure that the lookups will be solved with a single hop, even for hypothetical systems with fixed size and peer behavior. In contrast, D1HT is able to effectively buffer events for real and dynamic systems without sacrificing latency.

Besides D1HT, OneHop is the only other single-hop DHT that is able to buffer events for dissemination. However, while D1HT is a pure P2P and self-organizing system, OneHop relies on a three-level hierarchy to implement event buffering, and its topology incurs high levels of load imbalance among its different types of nodes. Additionally, to achieve its best performance, all nodes in an OneHop system must agree on some system-wide topological parameters [48], which are likely to be difficult to implement in practice, especially as the best parameters should change over time according to the system size and behavior.

In addition to the differences discussed above, D1HT is able to achieve overheads that are typically one order of magnitude smaller than those of both 1h-Calot and OneHop, as we will see in Section VIII.

Except for D1HT, 1h-Calot and OneHop, all other single-hop DHTs do not support large and dynamic environments [12], [21], [40], [41], [43]. Among these, our 1h-Calot overhead results should be valid for SFDHT [21] and 1HS [40], as 1HS is based on the 1h-Calot maintenance algorithm and SFDHT uses a similar event dissemination mechanism.

While D1HT solves lookups with just one hop even in dynamic environments, a few other systems (e.g., [1], [15], [22], [24], [29]) solve lookups with a constant number (i.e., $O(1)$) of multiple hops. Those systems are unable to provide optimal latencies and differ from D1HT in other important aspects. Z-Ring [22] uses Pastry [44] to solve lookups with two hops in systems with up to 16 million nodes. Tulip [1] and Kelips [15] use gossip to maintain routing tables of size $O(\sqrt{n})$ to solve lookups with two hops. Structured Superpeers [29] and LH* [24] use hierarchical topologies to solve lookups with three hops. Accordion [20] and EpiChord [18] do not ensure a maximum number of lookup hops, but they use parallel lookups and adaptation techniques to minimize lookup latencies. Beehive [37] is a replication framework to speed up lookups for popular keys.

Quarantine approaches have been proposed as a means of intervention for preventing vulnerabilities in the Internet, such as worm threats [32], but, to the best of our knowledge, we were the first to propose, evaluate and show the effectiveness of a quarantine approach for P2P systems [30].

III. D1HT System Design

A D1HT system is composed of a set $\mathbb{D}$ of n peers, and the keys are mapped to peers based on *consistent hashing* [16], where both peers and keys have IDs taken from the same identifier ring $[0 : N]$, with $N >> n$. The key and peer IDs are, respectively, the hashes (e.g., SHA1 [33]) of the key values and the peer IP addresses.

In D1HT, each peer has a full routing table, and so any lookup can be solved with just one hop, provided that its routing table is up to date. However, if the origin peer is unaware of an event that has happened in the vicinity of the target peer (e.g., a node has joined or left the system), the lookup may be initially addressed either to a wrong peer or to a peer that has already left the system. In both cases, the lookup will eventually succeed, provided that each peer has a pointer to its correct successor in the ring [47], but it will take longer than expected. To completely avoid those *routing failures* (as the lookup will eventually succeed, we do consider it as a *routing failure* instead of a *lookup failure*), D1HT would have to immediately notify all its n peers about the occurrence of any event in the system, which is simply infeasible. In practice, single-hop DHTs must try to keep the fraction of routing failures below an acceptable maximum by implementing mechanisms that can quickly notify all peers in the system about the events as they happen. These event dissemination mechanisms represent the primary distinction among the single hop DHTs, and in the next section, we will present the EDRA mechanism introduced with D1HT.

As in other works (e.g., [14], [30]), we will assume that the systems are churned with an event rate (or churn rate) r proportional to the system size, according to Equation III.1 below, where S_{avg} is the peer average session length.

$$r = 2 \cdot n/S_{avg} \qquad \text{(III.1)}$$

We refer to the *session length* as the amount of time between a peer's join and its subsequent leave; thus, Equation III.1 simply assumes that each peer generates two events per session (one join and one leave). As the average session lengths of a number of different P2P systems have already been measured (e.g., [6], [45], [46]), the equation above allows us to calculate event rates that are representative of widely deployed applications. In Sections VII and VIII, we will present experimental and analytical results with different session lengths, which will allow us to evaluate its effect on the maintenance overheads.

In D1HT, any message should be acknowledged to allow for retransmissions in the case of failures, which can be done implicitly by a protocol like TCP or be explicitly implemented over an unreliable protocol like UDP. We assume that the maintenance messages are transmitted over UDP to save bandwidth, but we consider the choice of the transport protocol for all other messages as an implementation issue. We also consider that the details of the joining protocol should be decided at the implementation level. In Section VI, we will discuss how we ensure message delivery in our D1HT implementation and what joining protocol we used.

D1HT is a pure P2P and self-organizing system, but its flat topology does not prevent it from being used as a component of hierarchical approaches aiming to exploit the heterogeneity of the participant nodes in a system. For example, the popular FastTrack network [23] has two classes of nodes: the super nodes (SN) and ordinary nodes (ON). SNs are better provisioned nodes, and each SN acts as a central directory for a

number of ONs, while flooding is used among the SNs. As measurements [23] have shown that FastTrack should have less than 40K SNs with an average session length of 2.5 hours, the analysis that we will present in Section IV shows that we could use a D1HT system to connect the SNs with maintenance costs as low as 0.9 kbps per SN. This overhead should be negligible, especially if we consider that the SNs are well-provisioned nodes and that we would avoid the flooding overheads while improving the lookup performance.

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

IV. EDRA

As each peer in a D1HT system should know the IP address of every other peer, any event should be *acknowledged* by all peers in the system in a timely fashion to avoid stale routing table entries. Here we say that a peer *acknowledges* an event when it either detects the join (or leave) of its predecessor or receives a message notifying it of an event. Then, to efficiently propagate any event to all peers in a system, D1HT makes use of the Event Detection and Report Algorithm (EDRA), which can announce any event to the whole system in logarithmic time with a pure P2P topology and provides good load-balance properties coupled with a very low bandwidth overhead. Additionally, EDRA is able to group several events into a single message to save bandwidth, yet it ensures an upper bound on the fraction of stale routing table entries.

At first glance, grouping several event notifications per message seems to be an obvious and easy way to save bandwidth, as any peer can locally buffer the events that occur during a period of time and forward them in a single message. However, such a mechanism imposes delays in the event dissemination, which in turn will lead more stale entries in the routing tables; thus, the difficult question is the following: *For how long can each peer buffer events while assuring that the vast majority of the lookups (e.g., 99%) will be solved with just one hop?* This problem is especially difficult because the answer depends on a number of factors that vary unpredictably, including the system size and churn rate. EDRA addresses this issue based on a theorem that will be presented in this section, which allows each peer to independently adjust the length of the buffering period while assuring that at least a fraction 1-f of the lookups will be solved with a single hop (f is typically 1%, but it can be tuned according to the application)

In this section, we will formally describe EDRA by means of a set of rules, prove its correctness and load balance properties, and present its analysis. Before we begin, we will define 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 $succ(p,i\text{-}1)$ for $i > 0$. Note that for $i \geq n$, $succ(p,i)=succ(p,i\text{-}n)$. 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 $pred(p,i\text{-}1)$, for $i > 0$. As in [26], 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 $stretch(p,n\text{-}1)=\mathbb{D}$ for any $p \in \mathbb{D}$.

A. The EDRA Rules

In this section, we will first present a brief description of EDRA and then formally define it. To save bandwidth, each peer buffers the events acknowledged during intervals of Θ seconds (Θ intervals), where Θ is dynamically tuned (as it will be seen in Section IV-D). At the end of a Θ interval, each peer propagates the events locally buffered by sending up to $\rho=\lceil \log_2(n) \rceil$ maintenance messages, as shown in Figure 1. Each maintenance message $M(l)$ will have a Time-To-Live (TTL) counter l in the range [0:ρ) and will be addressed to $succ(p,2^l)$. To perform event aggregation while assuring that any event will reach all peers in the system, each message $M(l)$ will include all events brought by any message $M(j), j > l$, received in the last Θ seconds. To initiate an event dissemination, the successor of the peer suffering the event will include it in all messages sent at the end of the current Θ interval.

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

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

Rule 2: Each maintenance message $M(l)$ will have a distinct TTL l, $0 \leq l < \rho$, and carry a number of events. All events brought by a message $M(l)$ 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: Messages 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 P does not receive any message from its predecessor p for T_{detect} sec, P will probe p to ensure that it has left the system and, after confirmation, P will acknowledge p leaving.

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 $TTL=\rho$ (so it 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 $stretch(p,k)$.

Rules 4 and 5 should allow each peer to maintain pointers to its correct successor and predecessor even in the case of peer failures. Moreover, to improve robustness, any peer p should run a local stabilization routine, similar to that used in Chord, whenever it does not receive a reply to a msg with TTL=0 or when it receives a msg with TTL=0 (or TTL=1) from others than its predecessor (or $pred(p,1)$).
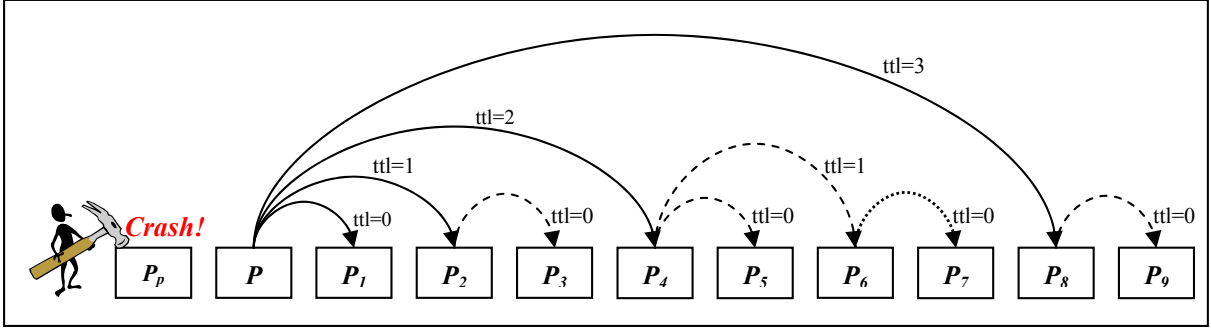
Fig. 1. This figure shows a D1HT system with 11 peers, where peer p crashes and this event is detected and reported by its successor P. In the figure, peers P_i are such that $P_i = succ(P, i)$. The figure also shows the TTL of each message sent.

Figure 1 shows how EDRA disseminates information about one event to all peers in a system according to the rules just presented, and it illustrates some properties that we will formally prove in the next section. The figure presents a D1HT system with 11 peers ($\rho = 4$), where peer p crashes and this event ε is detected and reported by its successor P. The peers in the figure are shown in a line instead of a ring to facilitate the presentation. Note that P acknowledges ε after T_{detect} sec (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. As P_2, P_4 and P_8 will acknowledge ε with $TTL > 0$, they will forward it to $P_3 = succ(P_2, 2^0)$, $P_5 = succ(P_4, 2^0)$, $P_6 = succ(P_4, 2^1)$ and $P_9 = succ(P_8, 2^0)$, as represented by the dashed arrows. Because 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 from forwarding ε to $succ(P_8, 2^1)$ and $succ(P_8, 2^2)$, which in fact are P and P_3, saving these two peers from having to acknowledge ε twice.

B. EDRA Correctness

The EDRA rules ensure that any event will be delivered to all peers in a D1HT system in logarithmic time, as we will shortly show in Theorem 1. For this theorem, we will ignore message delays and consider that all peers have synchronous intervals, i.e., the Θ intervals of all peers start at exactly the same time. The absence of message delays means that any message will arrive immediately at its destination, and because 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. We will also assume that no new event happens until all peers are notified about the previous event. All these practical issues will be addressed in Section IV-C.

Theorem 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 such a way that ε will be acknowledged exactly once by all peers in $stretch(p, 2^l\text{-}1)$ and by no other peer in the system. The average time T_{sync} for a peer in $stretch(p, 2^l\text{-}1)$ to acknowledge ε will be at most $l \cdot \Theta/2$ after p acknowledged ε.*

Proof: By strong induction in l. For $l=1$, the EDRA rules imply that p will only forward ε through a message with $TTL=0$ 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 sec after p acknowledged it, making the average time for peers in $stretch(p, 1) = \{p, succ(p, 1)\}$ to be $T_{sync} = (\Theta + 0)/2 = \Theta/2$ (at most). Thus, the claim holds for $l=1$.

For $l > 1$, the EDRA rules imply that p will forward ε through l messages at the end of the current Θ interval, each one with a distinct TTL in the range [0 , l). Then, after Θ sec (at most) 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 deduce that each acknowledgment made by a peer p_k implies that all peers in $stretch(p_k, 2^k\text{-}1)$ will acknowledge ε exactly once. Accounting for all l-1 acknowledgments made by the peers p_k, and that Rule 8 will prevent ε from being acknowledged twice by any peer in $stretch(p, 2^\rho\text{-}n)$, we conclude that ε will be acknowledged exactly once by all peers in $stretch(p, 2^l\text{-}1)$. By the induction hypothesis, none of those peers will forward ε to a peer outside this range, so ε will not be acknowledged by any other peers in the system. The induction hypothesis also ensures that the average time for the peers in each $stretch(p_k, 2^k\text{-}1)$ to acknowledge ε will be (at most) $k \cdot \Theta/2$ after the respective peer p_k acknowledged it, which will lead to $T_{sync} = l \cdot \Theta/2$ (at most) for $stretch(p, 2^l\text{-}1)$. ∎

Applying Theorem 1 and the EDRA rules to a peer join (or leave) that is acknowledged by its successor p, we can conclude that this event will be further acknowledged exactly once by all peers in $stretch(p, 2^\rho\text{-}1) = \mathbb{D}$. Moreover, the upper bound on the average acknowledge time will be $\rho \cdot \Theta/2$. We can thus formally ensure three very important EDRA properties. First, any event will be announced to all peers in a D1HT system, ensuring that they will receive the necessary information to maintain their routing tables. Second, each peer will be notified of any event just once, avoiding unnecessary bandwidth overheads and ensuring good income load balance. Third, for each event, the average notification time is bounded by $\rho \cdot \Theta/2$, and this result will be used in Section IV-D to develop a mechanism that will allow each peer in a D1HT system to dynamically find the optimal value for Θ based on the current system size and behavior.

C. Practical Aspects

In Theorem 1, we did not consider the effects of message delays and asynchronous Θ intervals; thus, we will turn to them in this section. To compute those effects, we will assume that each maintenance message will require an average delay of δ_{avg} to reach its target, and it will typically arrive at the middle of a Θ interval. Therefore, under those more realistic assumptions, each peer in the event dissemination path should add an average of $\delta_{avg}+\Theta/2$ to the event propagation time, leading to the adjusted value $\rho \cdot (2 \cdot \delta_{avg} + \Theta)/4$. Note that we have not yet considered the time to detect the event, which we will assume to be $T_{detect}=2 \cdot \Theta$, reflecting the worst-case scenario in which, after one missing message with $TTL=0$, a peer will probe its predecessor for up to Θ sec before reporting its failure. Thus, the upper bound on the average acknowledge time for any event will be

$$T_{avg} = 2 \cdot \Theta + \rho \cdot (\Theta + 2 \cdot \delta_{avg})/4 \; sec \qquad \text{(IV.1)}$$

Equation IV.1 overestimates T_{avg} because it only considers the worst-case of peer failures, whereas we should have set $T_{detect} = 0$ for joins and voluntary leaves.

In Theorem 1, we also considered that no new event would happen until all peers had been notified of a previous event, which is not a reasonable assumption for real and dynamic systems. While the admission of new peers should be correctly handled by the joining protocol, peer leaves are more complicated, and we may not expect that all peers in a system will have identical routing tables. For instance, when a peer fails before forwarding the locally buffered events, the propagation chain for these events will be partially broken. However, because this problem may occur only once per peer session (at most), it should not have a significant effect, as the duration of the buffering period (a few tens of seconds at most [30]) is typically orders of magnitude smaller than the average session length (e.g., almost three hours for KAD and Gnutella). In fact, in Section VII, we will see that D1HT was able to solve more than 99% of the lookups with just one hop in all experiments, even under a high rate of concurrent joins and leaves, which is a strong experimental evidence that the routing failures due to those practical issues should not be relevant in relation to f.

Besides the problems discussed so far, there are a number of practical situations that can lead to stale routing table entries in D1HT and other DHT systems, and we will not be able to completely remedy all of them. For that reason, as in many other systems (e.g., [14], [15], [18], [20], [27]), any D1HT implementation should allow the peers to learn from the lookups and maintenance messages to perform additional routing table maintenance without extra overhead. For example, a message received from an unknown peer should imply its insertion in the routing table. In the same way, routing failures will provide information about peers that have left or joined the system. In addition, many other known mechanisms that are commonly used in other DHT systems could be implemented on top of our base D1HT design, such as event re-announcements [48] and gossip [41] to improve routing table accuracy, or parallel lookups (as in [18], [20]) to mitigate the latency penalties

caused by timeouts due to missed leave notifications. We should point out that even with parallel lookups, the D1HT lookup bandwidth demands would be much smaller than those of multi-hop DHTs.

D. Tuning EDRA

In this section, we will show how to tune EDRA to ensure that a given fraction 1-f of the lookups will be solved with one hop, where f can be statically defined (e.g., f=1%) or dynamically adjusted.

As the lookups are solved with just one hop, to achieve f it is enough to ensure that the hops will fail with probability f, at most. Assuming, as in many other studies (e.g., [14], [19], [29], [47]), that the lookup targets are random, the average fraction of routing failures will be a direct result of the number of stale routing table entries. In that manner, to satisfy f, it suffices to assure that the average fraction of stale routing table entries is kept below f [14].

Given that the average acknowledge time is at most T_{avg}, the average number of stale routing table entries will be bounded by the numbers of events occurred in the last T_{avg} seconds, i.e., $T_{avg} \cdot r$. Then, we should satisfy the inequality $T_{avg} \cdot r/n \leq f$, and thus, by Equations III.1 and IV.1, the maximum Θ value should be

$$\Theta = (2 \cdot f \cdot S_{avg} - 2 \cdot \rho \cdot \delta_{avg})/(8 + \rho) \; sec. \qquad \text{(IV.2)}$$

The equation above requires each peer to know the average message delay; to ease the implementation, we will simply assume that $\delta_{avg}=\Theta/4$, which is an overestimation according to previously published results [30], [45]. Then

$$\Theta = (4 \cdot f \cdot S_{avg})/(16 + 3 \cdot \rho) \; sec. \qquad \text{(IV.3)}$$

As all D1HT peers know about any event in the system, Equations III.1 and IV.3 allow each peer to dynamically calculate the optimal value for Θ based on the event rate that is observed locally, without the need for further communication or agreement with other peers. This allows each peer in a D1HT system to independently adapt to the environment dynamics to maximize the buffering period without penalizing latency, even for large real systems whose size and peer behavior typically change over time. In contrast, all other P2P single hop DHTs are unable to calculate the length of event buffering periods, even for hypothetical systems with fixed size and peer behavior.

To make D1HT more robust to sudden bursts of events, we extended the original D1HT analysis to allow each peer to overestimate the maximum number of events it may buffer (E) according to Equation IV.4 below. This equation was derived from Equation IV.3 with the assumption that peers in a D1HT system observe similar event rates (which led us to assume that $r=E/\Theta$).

$$E = (8 \cdot f \cdot n)/(16 + 3 \cdot \rho) \quad events \qquad \text{(IV.4)}$$

E. Maintenance Traffic and Load Balance

While we have proven that EDRA ensures a good income load balance, it does not seem at first glance to provide good

balance in terms of outgoing traffic. For instance, in Figure 1, peer P sent four messages reporting p crash, while P_1 did not send a single message. But we should not be concerned with the particular load that is generated by a single event, as it should not exceed a few bytes per peer. Nevertheless, we must guarantee good balance with respect to the aggregate traffic that is necessary to disseminate information about all events as they happen, and this outgoing maintenance load balance will rely on the random distribution properties of the hash function used. The chosen function is expected to distribute the peer IDs randomly along the ring, which can be accomplished by using a cryptographic function such as SHA-1 [33]. Then, as in many other studies (e.g., [14], [19], [26], [47]), we will assume that the events are oblivious to the peer IDs, leading to a randomly distributed event rate r. Thus, the average number of messages each peer sends per Θ interval will be (including message acknowledgments)

$$(N_{msgs} \cdot (v_m + v_a) + r \cdot m \cdot \Theta)/\Theta \quad \text{bit/sec} \qquad \text{(IV.5)}$$

where m is the number of bits to describe an event, and v_m and v_a are the bit overheads (i.e., headers) per maintenance message and per message acknowledgment, respectively. As in a D1HT system, no peer will exchange maintenance messages with any node outside $\mathbb{D}$, Equation IV.5 will reflect both the incoming and outgoing average maintenance traffic.

F. Number of Messages

Equation IV.5 requires us to determine the average number of messages a peer sends, which is exactly the purpose of the following theorem.

Theorem 2. *The set of peers S for which a generic peer p acknowledges events with $TTL \geq l$ satisfies $|S|=2^{\rho-1}$.*

Proof: By induction on j, where $j=\rho$-l. For $j=0$, Rule 2 ensures 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), so $S=\{pred(p,1)\}$, which leads to $|S|=1=2^0=2^{\rho-l}$.

For $j > 0$, $l = \rho$-$j < \rho$. As S is the set of peers for which p acknowledges events with $TTL \geq l$, we can say that $S=S1 \cup S2$, where $S1$ and $S2$ are the sets of peers for which p acknowledges events with $TTL=l$ and $TTL > l$, respectively. By the induction hypothesis, $|S2|=2^{\rho-(l+1)}$. As $l < \rho$, the predecessor p will not be in $S1$ (Rule 6). Thus, as Rule 7 implies that p only receives messages with $TTL=l$ from a peer k, where $k=pred(p,2^l)$, we have that $S1$ will be the set of peers for which k forwards events through messages with $TTL=l$. By Rule 3, $S1$ is the set of peers for which k acknowledges events with $TTL > l$, and as the induction hypothesis also applies to the peer k, it follows that $|S1|=2^{\rho-(l+1)}$. By Theorem 1, we know that any peer p acknowledges each event only once, ensuring that $S1$ and $S2$ are disjoint, and thus, $|S|=|S1| + |S2|=2^{\rho-(l+1)} + 2^{\rho-(l+1)}=2^{\rho-l}$. ∎

The EDRA Rules 3 and 4 ensure that a peer p will only send a message with $TTL = l > 0$ if it acknowledges at least one event with $TTL \geq l + 1$. Then, based on Theorem 2, we can state 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 suffering an event in a Θ interval is $\Theta \cdot r/n$, the probability $P(l)$ of a generic peer sending a message with $TTL = l \neq 0$ at the end of each Θ interval is

$$P(l) = 1 - (1 - 2 \cdot r \cdot \Theta/n)^k, \quad \text{where } k = 2^{\rho-l-1}. \quad \text{(IV.6)}$$

As the messages with TTL=0 are always sent, the average number of messages sent by each peer per Θ interval will be

$$N_{msgs} = 1 + \sum_{l=1}^{\rho-1} P(l). \qquad \text{(IV.7)}$$

V. 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 departs quickly from the system. While ideally all peers in a DHT system should be able to solve lookups with a single hop at any time, in extremely large and dynamic systems the overheads caused by the most volatile peers can be excessive. Moreover, P2P measurement studies [9], [45], [46] have shown that the statistical distributions of session lengths are usually heavy tailed, which means that peers that have been 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, in which a joining peer will not be immediately allowed to take part in the D1HT overlay network, but it will be able to perform lookups at any moment. In this way, the most volatile peers will cause insignificant overheads to the system, while the other peers will be able to solve lookups with just one hop most of the time (typically, during more than 95% of their session lengths).

To join a D1HT system, a joining peer p retrieves the keys and IP addresses from a set of peers $\mathbb{S}$ (which can include just one peer, e.g., the successor of the joining peer). With Quarantine, the peers in $\mathbb{S}$ will wait for a Quarantine period T_q (which can be fixed or dynamically tuned) before sending the keys and IP addresses to p, postponing its insertion into the D1HT ring. While p is in Quarantine, its join will not be reported, and it will not be responsible for any key. The overhead reductions attained can be analytically quantified based on the Quarantine period and the statistical distribution of the session lengths, as in a system with n peers, only the q peers with sessions longer than T_q will effectively take part of the overlay network and have their events reported.

To be able to perform lookups during its Quarantine, a quarantined peer p will choose the nearest (in terms of latency) and best provisioned peers from $\mathbb{S}$ and will forward its lookups to those *gateway* peers. To avoid excessive loads, each gateway peer may limit the rate of lookups it will solve on behalf of quarantined peers, even though the experimental results that we will show in Section VII-C, where each D1HT peer used less than 0.1% of the available CPU cycles, indicate that the load imposed on the gateway peers should not be high. Anyway, the extra load imposed to the gateway peers
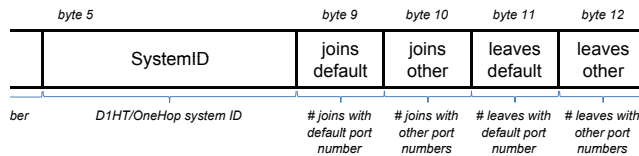
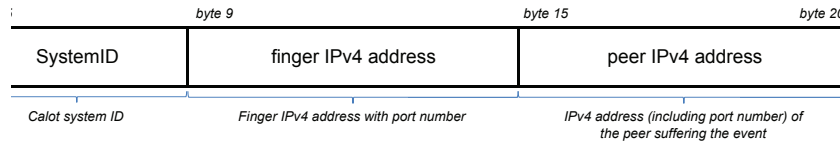Fig 2.a: D1HT and OneHop message headers

Fig 2.b: 1h-Calot message headers.

Fig. 2. Message headers used in our implementations and analyses. The SeqNo field is necessary to assure message delivery over UDP, and the SystemID field allows any peer to discard unsolicited messages received from other DHT systems. Each 1h-Calot maintenance message has a fixed size of 48 bytes (v_c=384 bits, including 28 bytes for the IPv4 and UDP headers). Each D1HT and OneHop message has a fixed part with 40 bytes (v_m=320 bits, including IPv4 and UDP headers), followed by the IPv4 addresses (without port numbers) of the peers that have joined and left the system in the default port (m=32 bits), and the IPv4 addresses (with port numbers) of the peers that have joined and left the system using others ports (m=48 bits). All acknowledgment and heartbeat messages for the three systems have just the four first fields shown (Type, SeqNo, PortNo and SystemID), and so $v_a=v_h$=288 bits (including IPv4 and UDP headers).

should be much inferior than those handled by superpeers (or supernodes) in hierarchical systems like FastTrack [23], OneHop [14] or Structured Superpeers [29].

With the Quarantine mechanism, we avoid the join and leave overheads for peers with session lengths smaller than T_q, but newly incoming peers will have their lookups solved in two hops while they are in Quarantine. We believe that this extra hop penalty should be acceptable for several reasons. First, the additional hop should have low latency, as it should be addressed to a nearby peer. Second, this extra overhead will only be necessary during a short period (e.g., 5% of the average session length). Third, the Quarantine mechanism should have beneficial effects even to the volatile and gateway peers, as they will not incur the overhead of transferring the keys and routing tables. Fourth, the Quarantine mechanism should significantly reduce the maintenance overheads of all peers in the system (as will be confirmed by the results presented in Section VIII).

Furthermore, the Quarantine mechanism can also be used for other purposes. For instance, we can improve the system robustness against malicious attacks if we allow T_q to be dynamically tuned in a way that suspicious peers will take longer to be fully accepted in the system, while their behaviors can be monitored during their Quarantine. We can also use Quarantine to minimize sudden overheads due to flash crowds, as we can increase T_q whenever the event rate reaches the upper limit that can be comfortably handled by the system.

VI. D1HT IMPLEMENTATION

We implemented D1HT from scratch, resulting in more than 8.000 lines of dense C++ code, even though we did not yet implement the Quarantine mechanism. This implementation is fully functional and was tested on thousands of nodes running Linux, and its source code is freely available [11].

Our implementation uses a variant of the Chord joining protocol [47], with a few important differences. First, any join is announced to the whole system by EDRA. Second, the new peer p gets the routing table from its successor p_s. Third, to prevent p from missing events while its joining is notified to the system, p_s will forward to p any event it knows until p receives messages with all different TTLs.

To save bandwidth and minimize latency, the maintenance and lookup messages are sent with UDP, and TCP is used for all other types of communications (routing table transfers, stabilizations, etc.). Each D1HT instance has a default IPv4 port, but any peer may choose an alternative port when joining the system. Thus, we expect that most events will be identified only by the peer's four byte IPv4 address (as most peers should use the default port), which led us to propose the message header layout as shown in Figure 2. Then, for Equation IV.5, we expect that the average m value will be around 32 bits.

Each D1HT peer stores its routing table as a local hash table indexed by the peer IDs in such a way that any peer needs only to store the IPv4 addresses of the participant peers (including the port number), leading to a memory overhead of about $6n$ bytes in each peer (plus some additional space to treat eventual hash collisions). In this way, for environments such as HPC and ISP datacenters, each routing table will require a few hundred KBs at most. For a huge one million Internet wide D1HT deployment, each routing table would require around 6 MB, which is negligible for domestic PCs and acceptable even for small devices, such as cell phones and media players.
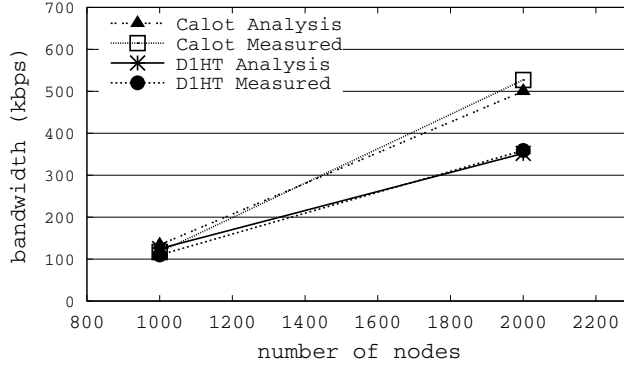
Fig. 3. Experimental and analytical outgoing maintenance bandwidth demands for D1HT and 1h-Calot in the PlanetLab.

VII. EXPERIMENTAL EVALUATION

In this section, we will present our D1HT and 1h-Calot experimental results, which will be complemented by our analytical evaluations presented in Section VIII.

It is worth noting the extensive experimental results we present in this section. First, we used two radically distinct environments, specifically an HPC datacenter and a worldwide dispersed network. Second, our DHT evaluation used the largest experimental testbed set up so far, with up to 4,000 peers and 2,000 physical nodes. Finally, we report the first latency comparison among DHTs and a directory server.

A. Methodology

The D1HT implementation used in our experiments was presented in Section VI, which includes only the base D1HT proposal without any extension. In this way, we should better evaluate the unique D1HT contributions, but we expect that our experimental results will reflect a worst case scenario in relation to production grade and better tuned D1HT implementations, which would probably include a number of well known optimizations (e.g., peer re-announcements, parallel lookups, etc.), even though our implementation has been already thoroughly tested.

Because 1h-Calot was not implemented by its authors, we had to develop a real implementation of that system for our experiments. To allow for a fair comparison, we implemented 1h-Calot after our D1HT code, and both systems share most of the code, in an effort to ensure that differences in the results are not due to implementation issues. Because each 1h-Calot maintenance message carries just one event, it does not make sense to include counters in its message headers, which will then have the format shown in Figure 2.

As in 1h-Calot each event incurs $2n$ maintenance messages (including acks), and each peer sends four heartbeats per minute (which are not acknowledged), the analytical average 1h-Calot peer maintenance bandwidth will be given by

$$B_{Calot} = (r \cdot (v_c + v_a) + 4 \cdot n \cdot v_h/60) \text{ bps}, \qquad \text{(VII.1)}$$

where v_c, v_a and v_h are the sizes of the maintenance, acknowledgment and heartbeat messages, respectively (as shown in Figure 2).

Each experiment evaluated both systems with a specific session length S_{avg} and a given network size n. In all experiments, we used S_{avg}=174 min, as this value is representative of Gnutella [45] and it was used in other studies (e.g., [14], [30]). In some experiments, we also used S_{avg}=60 min to exercise the systems under more dynamic scenarios. The bandwidth results considered only the traffic for routing table maintenance and peer failure detection, as the other overheads involved, such as lookup traffic and routing table transfers, should be similar for all single-hop DHTs. For all experiments, we defined the routing tables with 6K entries (around 36KB).

Each experiment had two phases, where the first one was used to grow the system up to the target size and the second phase was used for the measurements. In the first phase, each system started with just eight peers, and one peer joined per second until the target size was reached, resulting in a very steep growth rate (the systems doubled in size in just eight seconds, with an eightfold growth in less than one minute), which should stress the joining protocols. The second phase always lasted for 30 min, while each peer performed random lookups. We ran each experiment three times and reported the average results.

In both phases of all the experiments, the systems were churned according to Equation III.1 and the chosen S_{avg} (60 or 174 min), and the peer leaves were random. Half of the peer leaves were forced with a POSIX SIGKILL signal, which does not allow the leaving peer to warn its neighbors nor to flush any buffered event. To maintain the system size, any leaving peer rejoined the system in three minutes with, unless otherwise stated, the same IP and ID, which allowed us to evaluate both systems in a scenario with concurrent joins and leaves.

Even though our experiments stressed the joining protocols and imposed a high rate of concurrent joins and leaves, both D1HT and 1h-Calot were able to solve more than 99% of the lookups with a single hop in all experiments, showing that in practice, the issues discussed in Section IV-C should not have a relevant impact.
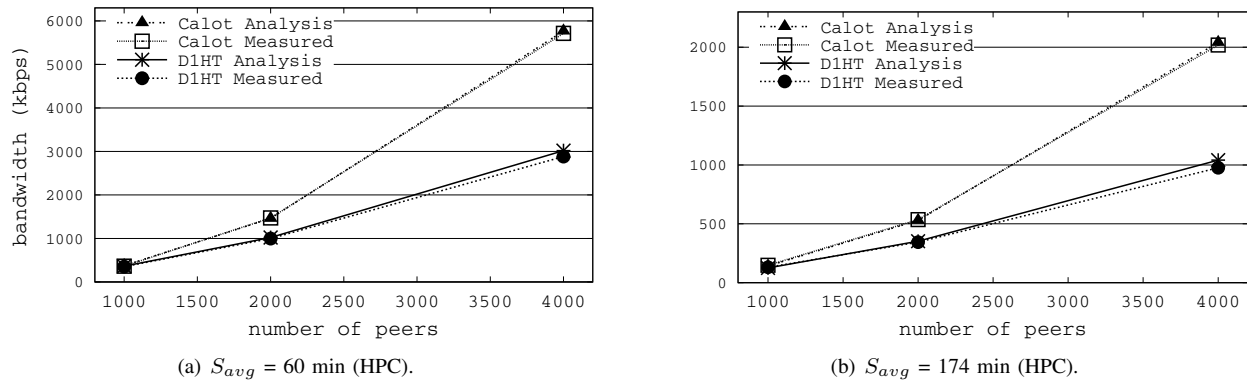
(a) S_{avg} = 60 min (HPC).

(b) S_{avg} = 174 min (HPC).

Fig. 4. Experimental and analytical outgoing maintenance bandwidth demands for D1HT and 1h-Calot in the HPC datacenter.

Cluster	# nodes	CPU	OS
A	731	Intel Xeon 3.06GHz single core	Linux 2.6
B	924	AMD Opteron 270 dual core	Linux 2.6
C	128	AMD Opteron 244 dual core	Linux 2.6
D	99	AMD Opteron 250 dual core	Linux 2.6
F	509	Intel Xeon E5470 quad core	Linux 2.6

TABLE I

CLUSTERS USED IN OUR EXPERIMENTS. EACH NODE HAS TWO CPUS.

B. PlanetLab Bandwidth Experiments

To evaluate the system overheads in a worldwide dispersed environment, we ran experiments using 200 physical PlanetLab [5] nodes, with either 5 or 10 D1HT and 1h-Calot peers per node, leading to system sizes of 1K or 2K peers, respectively. Each peer performed one random lookup per second during the second phase of our PlanetLab experiments.

The measured and analytical maintenance bandwidth demands are plotted in Figure 3, showing that both DHTs had similar overheads for the smaller system size, while with 2K peers the demands of 1h-Calot were 46% higher than those of D1HT. The more extensive experiments and analyses presented in Sections VII-C and VIII will show that this difference will significantly increase with larger system sizes. Figure 3 also shows that the analyses of both systems were able to predict their bandwidth demands, which differ to some extent from previous results [31], where the D1HT analysis overestimated its overheads by up to 25%. We credit those differences to a few factors. First, D1HT had an increase in bandwidth demands due to the implementation of the mechanisms to close Θ intervals based on Equation IV.4, which was not used in the experiments reported in [31]. Additionally, because the D1HT analysis is strongly dependent on $\rho=\lceil\log_2(n)\rceil$, it leads to more precise predictions when ρ is close to $\log_2(n)$ (i.e., when n is slightly smaller than a power of 2, which is the case with all experiments presented here) and to overestimated results when ρ is significantly greater than $\log_2(n)$ (as with the results presented in [31]).

C. HPC Bandwidth Experiments

We also performed experiments on a subset of five clusters at a Seismic Processing HPC datacenter [34] (see Table I). In that network, each node has a Gigabit Ethernet connection to

an edge switch, while each edge switch concentrates 16 to 48 nodes and has a 2 Gbps or 10 Gbps Ethernet connection to a non-blocking core switch.

Each peer performed one random lookup per second during the second phase of these experiments, which were conducted with the clusters under normal datacenter production, where typically most of the nodes were experiencing 100% CPU use, as imposed by the Seismic Processing parallel jobs. Nevertheless, we were able to run all our experiments smoothly, without any single interference in the normal datacenter production, confirming that it is absolutely feasible to run these DHT systems in heavily loaded production environments. In fact, in all our HPC bandwidth experiments the average CPU use per D1HT peer was below 0.1% (including the cycles used by the joining mechanism and the lookups).

Figure 4(a) and 4(b) show the sum of the outgoing maintenance bandwidth requirements of all peers for each system for different churn rates. We plotted the measured and analytical requirements, showing that, as in the PlanetLab results, the analyses for both systems were precise. The figures also show that D1HT had lower maintenance bandwidth requirements for all cases studied, once more confirming in practice that D1HT can provide a more lightweight DHT implementation than 1h-Calot.

We also ran our biggest case (4,000 peers) with the leaving peers rejoining with new IPs and IDs to evaluate whether the reuse of IDs caused any relevant bias in our results. In fact, without reusing the IDs, the fraction of the lookups solved with one hop dropped by less than 0.1%, but it remained well above our 99% target, which allowed us to conclude that the reuse of IDs did not cause any significant bias in our results.

D. HPC Latency Experiments

In this section, we will present our latency experiments performed in the HPC datacenter. As the lookup latencies are sensitive to the network load, we used 400 idle nodes from Cluster A (see Table I) but, as we expect that DHTs should be able to be used in heavy loaded production environments, we measured the latencies with those nodes both in the idle state and under 100% CPU load (by running two burnP6 [10] instances per node). Because we used dedicated nodes, we could increase the lookup rate during the second phase of the
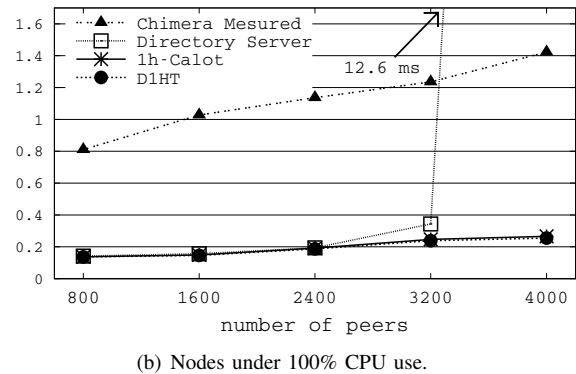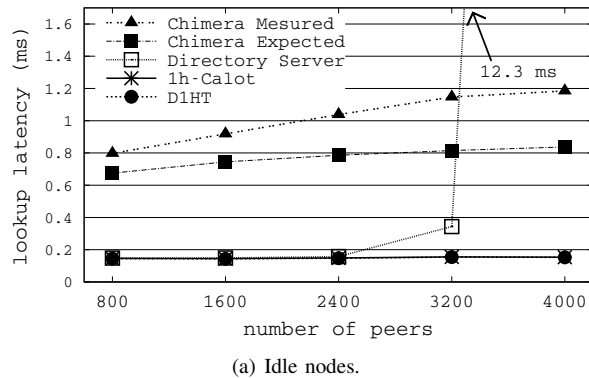
Fig. 5. Lookup latencies measured in the HPC environment with idle and busy (100% CPU use) nodes.

experiments to 30 lookups/sec per peer, which allowed us to evaluate the systems under an intense lookup load.

In addition to D1HT and 1h-Calot, we also ran a multi-hop DHT and a directory server (Dserver). In an effort to avoid inserting bias due to implementation issues, Dserver was essentially a D1HT system with just one peer. We first ran Dserver in a Cluster B node, which reached 100% CPU load when serving lookups from 1,600 peers, thus providing a first indication of the scalability issues of this client/server approach, after which we picked up a dedicated node from Cluster F. For the multi-hop DHT, we used Chimera [8] not only because it implements Pastry (using base 4), which is one of the most prominent multi-hop DHTs, but also because it did not require any prerequisites to be installed in the HPC clusters (e.g., Java, Python, etc.).

As our time windows with the dedicated cluster were limited, we ran the four systems concurrently in each experiment. To study different system sizes, we varied the numbers of DHT peers and Dserver clients per node from two to ten. For example, when running six peers per node, we concurrently ran six D1HT peers, six 1h-Calot peers, six Chimera peers and six Dserver clients in each cluster A node.

The D1HT and 1h-Calot peers were churned with S_{avg}=174 min, while Dserver and Chimera were not churned. To verify whether the base latencies of the studied systems differ due to implementation issues, we first ran the four systems with just two peers and the observed one-hop latencies were quite similar (around 0.14 ms).

Figure 5(a) and 5(b) show the latencies measured with the nodes in the idle state and under 100% CPU load, respectively. As the measured Chimera latencies were higher than expected, we also plotted the expected Chimera latencies assuming that each hop takes 0.14 ms. We believe that the differences between the measured and expected Chimera latencies were due to either implementation issues or measurement artifacts, but even the expected latencies are much higher than those for the single-hop DHTs, which confirms that a multi-hop DHT solution is less suitable for latency-sensitive applications. While Chimera latencies could be improved by using a larger base (e.g., 16), its performance would still be worse than that of D1HT.

We can see from Figure 5(a) that all systems, except for Chimera, presented very similar latencies with idle nodes and
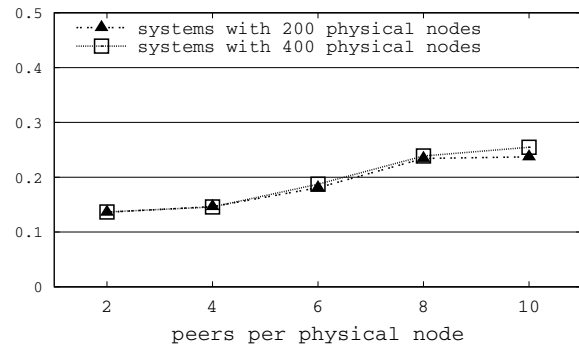

Fig. 6. Lookup latencies measured in the HPC environment with busy (100% CPU use) nodes.

smaller system sizes, which was expected because D1HT and 1h-Calot solved more than 99% of the lookups with one hop, while Dserver ran similar code. However, Dserver started to lag behind the single-hop DHT systems at 3,200 peers (120% higher latencies), and at 4,000 peers it provided latencies more than one order of magnitude higher, revealing its scalability limitations.

We may observe in Figure 5(b) that the latencies of all systems degraded with busy nodes and that, quite surprisingly, the D1HT and 1h-Calot latencies increased slightly with the system size when running on busy nodes. To verify whether this unexpected behavior was related to the extra load generated by the artifact of running several peers under an intense lookup rate and four different systems per node, we ran the 100% CPU load experiments with just 200 physical nodes, varying again the number of peers per node from two to ten. The results are plotted in Figure 6 along with the latencies measured with 400 nodes. For simplicity, in Figure 6, we only plot the D1HT results, even though we also ran 1h-Calot, Chimera and Dserver for both the 200 and 400 node experiments. Confirming our hypothesis, the figure indicates that the latency degradation observed was related to the number of peers per physical node (and the overload they imposed on the already 100% busy nodes), as the latencies measured with 200 and 400 nodes and the same number of peers per node were quite similar, even though the 400-node systems had twice the size. For instance, with four peers per node, the average latencies measured with 200 nodes (total of 800 peers) and 400 nodes (total of 1,600 peers) were both 0.15 ms. With eight peers per node, the results with 200 nodes

(a) S_{avg}=60 min.

(b) S_{avg}=169 min (KAD dynamics).

(c) S_{avg}=174 min (Gnutella dynamics).

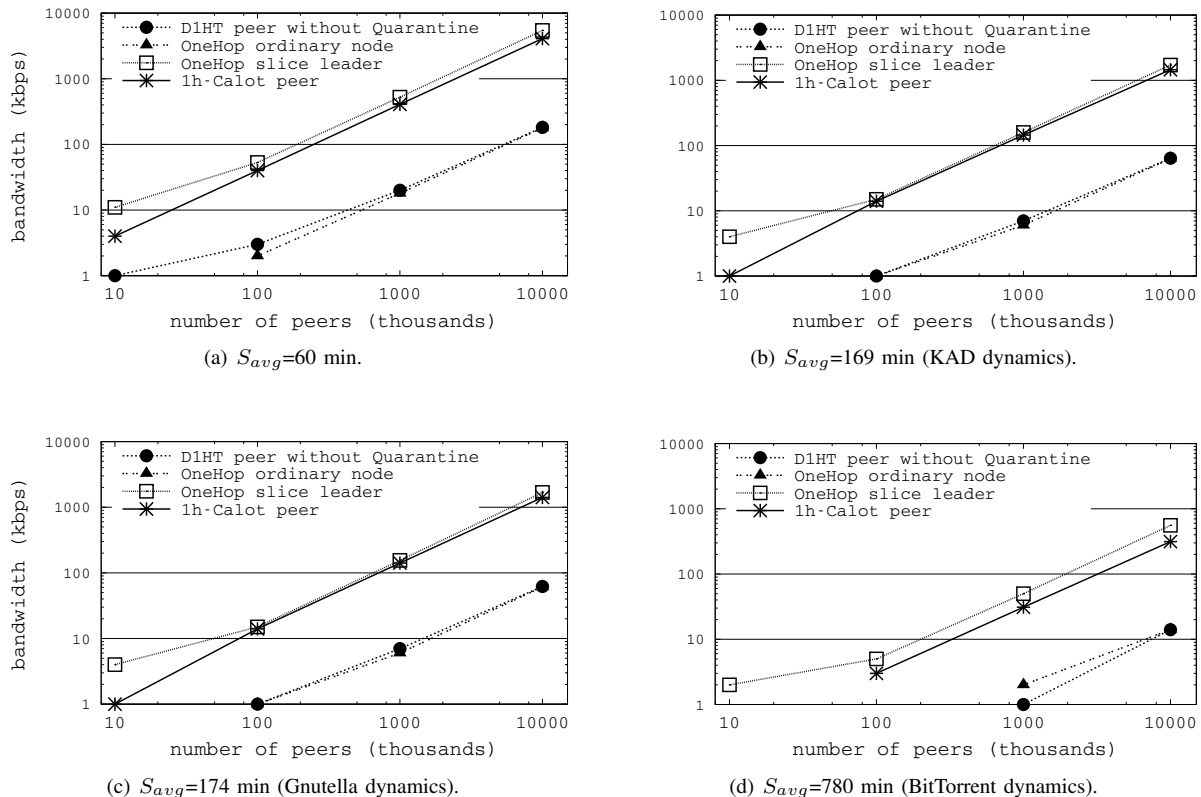(d) S_{avg}=780 min (BitTorrent dynamics).

Fig. 7. Log-log plots showing the analytical outgoing maintenance bandwidth demands for D1HT, 1h-Calot and OneHop (we do not show values below 1 kbps).

(total of 1,600 peers) and 400 nodes (total of 3,200 peers) were 0.23 ms and 0.24 ms, respectively. These results indicate that the D1HT lookup latencies should not vary with the system size, but they can degrade with overloaded peers, while they are still similar to or better than those provided by Dserver.

VIII. ANALYTICAL RESULTS

As our experiments have validated the 1h-Calot and D1HT analyses, and the OneHop analysis had already been validated in a previous work [14], we will now compare those three systems analytically. As discussed in Section II, the 1h-Calot results presented in this section should also be valid for the 1HS [40] and SFDHT [21] systems. In a previous work [30], we have already provided an extended D1HT analysis, studying the variation of the D1HT overheads and Θ intervals for different values of f, churn rates and system sizes; thus, here we will focus on comparing the overheads of the systems being studied.

As in our experiments, our analytical results compute only the traffic for routing table maintenance, we used f=1%, and we assumed random events and lookups. The OneHop analysis is available from [14], for which we will consider the same message formats used in our D1HT implementation, as shown in Figure 2, because they have been shown to be realistic in practice. Besides, the OneHop results always considered the optimal topological parameters and did not account for the failure of slice and unit leaders. The OneHop and 1h-Calot

results do not consider message delays, while for D1HT we used δ_{avg}=0.25 sec, which is an overestimation compared to the Internet delay measurements presented in [45].

We varied the system size from 10^4 to 10^7, which are representative of environments ranging from large corporate datacenters to huge Internet applications, and studied average sessions of 60, 169, 174 and 780 min, where the latter three were observed in KAD [46], Gnutella [45] and BitTorrent [2] studies. This range of session lengths is more comprehensive than those used in most DHT evaluations (e.g., [14], [19], [20], [29]–[31]) and is representative of widely deployed P2P applications.

Figure 7(a) to 7(d) show log-log plots comparing the analytical bandwidth demands of D1HT and 1h-Calot peers against those of the best (ordinary nodes) and worst (slice leaders) OneHop cases. We can see that the OneHop hierarchical approach imposes high levels of load imbalance between slice leaders and ordinary nodes. Moreover, a D1HT peer typically has maintenance requirements one order of magnitude smaller than OneHop slice leaders, while attaining similar overheads compared to ordinary nodes. Compared to D1HT, the 1h-Calot overheads were at least twice greater and typically one order of magnitude higher for the cases studied. The requirements for a D1HT peer in systems with n=10^6 and average sessions of 60, 169, 174 and 780 min are 20.7 kbps, 7.3 kbps, 7.1 kbps and 1.6 kbps, respectively. In contrast, the overheads for the OneHop slice leaders and 1h-Calot peers for systems with n=10^6 and KAD dynamics were above 140 kbps.

(a) Quarantine gains with KAD dynamics ($q=0.76n$).

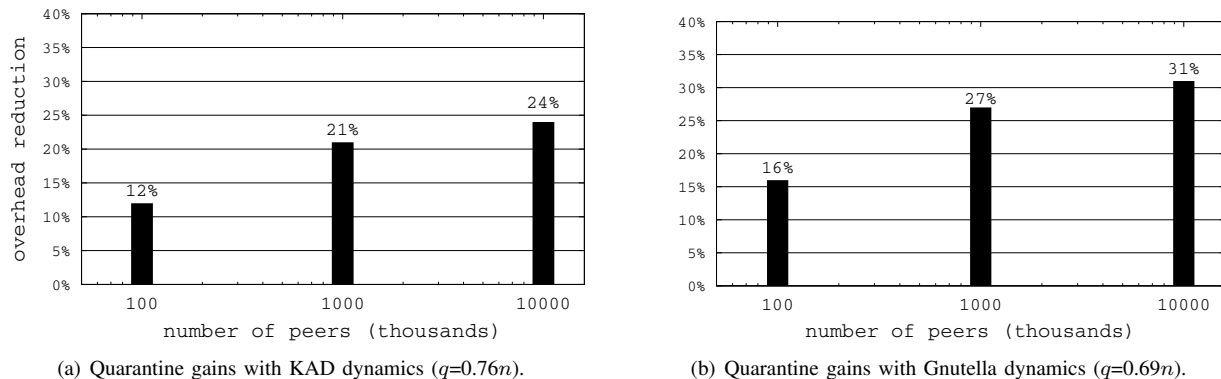(b) Quarantine gains with Gnutella dynamics ($q=0.69n$).

Fig. 8.   Estimated overhead reductions brought by Quarantine for systems with KAD and Gnutella behaviors.

The Quarantine analysis will be based on data from studies that observed that 31% of the Gnutella sessions [9] and 24% of the KAD sessions [46] lasted less than 10 minutes, which is a convenient value for the Quarantine period T_q. Then, Figure 8(a) and 8(b) show the overhead reductions provided by Quarantine for D1HT systems with dynamics similar to KAD and Gnutella, with T_q=10 min. We can see that the maintenance bandwidth reduction grows with the system size, as for very small systems the overheads were dominated by messages with TTL=0, which are always sent even when there are no events to report. Although the length of the Quarantine period studied was less than 6% of the average session length for both systems, the overhead reductions with $n=10^7$ for KAD and Gnutella dynamics reached 24% and 31% respectively, showing the effectiveness of the Quarantine mechanism.

IX. DISCUSSION

In addition to validating the D1HT analysis, our experiments confirmed that D1HT was able to solve more than 99% of the lookups with a single hop and minimal overhead, even with nodes under high CPU load or peers widely dispersed over the Internet. For instance, in all our HPC bandwidth experiments, the average CPU usage per peer was less than 0.1%, and the memory sizes for routing table storage were around 36 KB per peer.

Our results also showed that D1HT had the lowest overheads among all single-hop DHTs that support dynamic environments, with typical reductions of one order of magnitude. D1HTs extraordinary performance advantage was due to its ability to group events for dissemination with a pure P2P approach, even for large and dynamic environments where the peer behaviors change over time. In contrast, other single-hop DHTs either do not provide means for their peers to group events [21], [40], [41], [43], [48] or use a hierarchical approach with high levels of load imbalance and other intrinsic issues [14].

It is worth noting that while pure P2P topologies are commonly used to achieve high levels of robustness and better load balancing, they are usually associated with an increase in the overall bandwidth cost. In contrast, D1HT is both a self-organizing pure P2P system and has lower maintenance overheads than all other single-hop DHTs.

Compared to a directory server, D1HT achieved similar latencies for small systems while attaining much better scalability, which allowed it to provide latencies up to one order of magnitude better for the larger systems studied, even with nodes under full CPU load, revealing that D1HT is also an attractive solution for large-scale latency-sensitive applications.

Considering that, back in 2004, the BitTorrent peer average download speed was already around 240 kbps [36], we may assume that the D1HT with 1.6-16 kbps maintenance overheads should be negligible for systems with one to ten million peers and BitTorrent behavior. Moreover, as other studies have found that most domestic connections have at *least* 512 kbps of downstream bandwidth with very low occupation [13], [25], we argue that we should not penalize lookup latencies to save fractions below 10% of the available bandwidth. Thus, even systems with up to ten million nodes with KAD or Gnutella dynamics could benefit from the lowest latencies provided by D1HT with 50 kbps maintenance overheads, whereas they are not supported by either 1h-Calot or OneHop due to their overheads in excess of 1,400 kbps.

While 1h-Calot could also be used in HPC and ISP datacenters, its use would require the development and maintenance of a DHT dedicated to those environments. In contrast, the unique D1HT ability to provide both low latency and small overheads allows it to support a wide range of environments, in such a way that D1HT can act as a commodity DHT, which makes D1HT a very attractive option for these corporate datacenters, specially as they are preferably built on commodity hardware and software [4], [49].

X. CONCLUSION

While latency issues should become more critical than bandwidth restrictions over time, the first DHT proposals have opted to trade off latency for bandwidth, and recent single-hop DHTs typically have either high overheads or poor load balance. In this work, we presented D1HT, which has a pure P2P and self-organizing approach and is the first single-hop DHT combining low maintenance bandwidth demands and good load balance, along with a Quarantine mechanism that is able to reduce the overheads caused by volatile peers in P2P systems.

We performed the most complete and representative set of DHT comparative experiments published so far, which validated the D1HT analysis and was complemented by analytical studies. Specifically, by using an experimental environment that was at least 10 times greater than those of all previous DHT comparative experiments, we became the first to assess five key aspects of DHT behavior in such practical settings. Concretely, we were the first to i) report DHT experiments in two different environments; ii) compare DHT lookup latencies; iii) perform experiments with two different single-hop DHTs; iv) compare the latencies of multi and single hop DHTs; and v) compare DHTs to central directories.

Overall, our results showed that D1HT consistently had the lowest maintenance costs among the single-hop DHTs, with typical overhead reductions of one order of magnitude, and indicated that D1HT could be used even for huge systems with one million peers and dynamics similar to those of popular P2P applications.

Our experiments also showed that D1HT provides latencies comparable to those of a directory server for small systems, while exhibiting better scalability for larger ones, which should allow its use in very large latency-sensitive environments.

As a consequence of our extensive set of results, we may conclude that D1HT can potentially be used in a multitude of environments, ranging from HPC and ISP datacenters to huge P2P applications deployed over the Internet. This ability to support such a wide range of environments, which is unique among DHTs, should allow D1HT to be used as an inexpensive and scalable commodity software substrate for distributed applications. As one step in that direction, we have made our D1HT source code available for free use [11].

We believe that D1HT may be a key component in several Internet and datacenter distributed applications, since the improvements in both bandwidth availability and processing capacity that we should continuously get will bring performance expectations to users and applications, which can be frustrated by latency constraints. In addition, trends in High Performance Computing, ISP and Cloud Computing environments indicate significant increases in the system sizes, which will challenge the scalability and fault tolerance of client/servers solutions.

Acknowledgments

References

[1] I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron. Practical locality-awareness for large scale information sharing. In *Proc. of IPTPS*, Feb 2005.

[2] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on cooperation in BitTorrent communities. In *Proc. of the 3rd SIGCOMM Workshop on Economics of P2P Systems*, Aug 2005.

[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[4] L. Barroso and U. Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2008.

[5] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI*, 2004.

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

[7] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Elsevier Future Generation Computer Systems*, 25(6), Jun 2009.

[8] Chimera. http://current.cs.ucsb.edu/projects/chimera/, Jun 2011.

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

[10] CPUburn. http://pages.sbcglobal.net/redelm/. Jun 2011.

[11] D1HT. Source code available from http://www.lcp.coppe.ufrj.br/D1HT. Jun 2011.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *In Proc. SOSP*, 2007.

[13] M. Dischinger, K. Gummadi, A. Haeberlen, and S. Saroiu. Characterizing residential broadband networks. In *Proc. of IMC*, Oct 2007.

[14] P. Fonseca, R. Rodrigues, A. Gupta, and B. Liskov. Full information lookups for peer-to-peer overlays. *IEEE Transactions on Parallel and Distributed Systems*, 20(9), Sep 2009.

[15] 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.

[16] 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.

[17] V. Kindratenko and P. Trancoso. Trends in High-Performance Computing. *IEEE Computing in Science & Engineering*, 13(3), Jun 2011.

[18] B. Leong, B. Liskov, and E. Demaine. EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management. In *Proc. of ICON*, Nov 2004.

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

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

[21] L. Li, Z. Chunhong, W. Mi, Y. Zhang, T. Ma, and J. Yang. SFDHT: A DHT Designed for Server Farm. In *Proc. of GLOBECOM*, Nov 2009.

[22] Q. Lian, W. Chen, Z. Zhang, S. Wu, and B. Zhao. Z-ring: Fast prefix routing via a low maintenance membership protocol. In *ICNP*, 2005.

[23] J. Liang, R. Kumar, and K. Ross. The fasttrack overlay: A measurement study. *Computer Networks*, 50(6), Apr 2006.

[24] W. Litwin, R. Moussa, and T. Schwarz. LH*RS—a highly-available scalable distributed data structure. *ACM Trans. Database Syst.*, 30, 2005.

[25] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *IMC*, 2009.

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

[27] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS*, Mar 2002.

[28] K. McKusick and S. Quinlan. GFS : Evolution on Fast-forward. *ACM Queue*, 7(7), 2009.

[29] 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.

[30] L. Monnerat and C. L. Amorim. D1HT: A Distributed One Hop Hash Table. In *Proc. of IPDPS*, Apr 2006. Previously published as TR ES-676/05, UFRJ, May 2005 (available from http://www.lcp.coppe.ufrj.br).

[31] L. Monnerat and C. L. Amorim. Peer-to-Peer Single Hop Distributed Hash Tables. In *Proc. of IEEE GLOBECOM*, Nov 2009.

[32] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *INFOCOM*, 2003.

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

[34] J. Panetta, P. Souza, C. Cunha, F. Roxo, S. Sinedino, I. Pedrosa, A. Romanelli, L. Monnerat, L. Carneiro, and C. Albrecht. Computational characteristics of production seismic migration and its performance on novel processor architectures. In *International Symposium on Computer Architecture and High Performance Computing*, Nov 2007.

[35] D. Patterson. Latency lags bandwith. *Comm. ACM*, 47(10), 2004.

[36] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In *IPTPS*, 2005.

[37] 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.

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

[39] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The Oceanstore Prototype. In *Proc. of the USENIX Conference on File and Storage Technologies (FAST)*, Mar 2003.

[40] J Risson, A Harwood, and T Moors. Stable high-capacity one-hop distributed hash tables. In *Proc. of ISCC*, Jun 2006.

[41] J Risson, A Harwood, and T Moors. Topology dissemination for reliable one-hop distributed hash tables. *IEEE Transactions on Parallel and Distributed Systems*, May 2009.

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

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

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

[45] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. of MMCN*, Jan 2002.

[46] M. Steiner, T. En-Najjary, and E. Biersack. Long term study of peer behavior in the KAD DHT. *Trans. on Networking*, Oct 2009.

[47] 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. *Trans. on Networking*, Feb 2003.

[48] C. Tang, M. Buco, R. Chang, S. Dwarkadas, L. Luan, E. So, and C. Ward. Low traffic overlay networks with large routing tables. In *Proc. of SIGMETRICS*, Jun 2005.

[49] TOP500. http://www.top500.org.

[50] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *JSAC*, Jan 2004.