# A Pragmatic Approach to DHT Adoption

Jeffrey Considine* Michael Walfish† David G. Andersen†

## Abstract

Despite the peer-to-peer community's obvious wish to have its systems adopted, specific mechanisms to facilitate incremental adoption have not yet received the same level of attention as the many other practical concerns associated with these systems. This paper argues that ease of adoption should be elevated to a first-class concern and accordingly presents HOLD, a front-end to existing DHTs that is optimized for incremental adoption. Specifically, HOLD is *backwards-compatible*: it leverages DNS to provide a key-based routing service to existing Internet hosts without requiring them to install any software. This paper also presents applications that could benefit from HOLD as well as the trade-offs that accompany HOLD. Early implementation experience suggests that HOLD is practical.

## 1 Introduction

The community has invested enormous energy in practical considerations for peer-to-peer systems, including performance, maintenance, deployment, and management. Yet despite this practical focus and despite the community's obvious desire to have its systems adopted, specific mechanisms to attract end-users have not received the same level of attention.

We argue that in the particular case of peer-to-peer systems, ease of adoption should be elevated to a first-class concern, for two principal reasons (that are certainly not novel but bear repeating here): (a) these systems become most useful when a large number of people use them, and (b) peer-to-peer systems are expressly designed for populations of this size, so understanding how a given system behaves when massively deployed is especially important.

We believe that a significant practical barrier to adoption is asking users to install software. We therefore describe a *backwards-compatible* front-end to existing Distributed Hash Table (DHT) and Distributed Object Location and Routing (DOLR) [3] designs that:

- Exposes the *key-based routing (KBR)* component [3] of these systems to Internet users without requiring them to install any software or modify their host in any way; and
- Permits end-users who desire a full feature set to become first-class participants in the DHT or DOLR (which does require software installation).

Specifically, our approach, which we call Hashing Over Legacy DNS (HOLD), leverages DNS to provide a single service: allowing *any* Internet host to map an identifier to the responsible node in a given DHT.[1] Rather than try to expose to legacy users the many other capabilities of DHTs and DOLRs, we instead pragmatically concentrate on solutions that do not require any end-host modification.

We have identified a useful class of applications that use key-based routing alone and work transparently over legacy protocols. Two examples are: (1) a DNS-based block list for guarding against spam that would be fortified against the recent DDoS attacks that plagued centralized versions of an identical service [6, 14]; and (2) the back-end of a file sharing system for finding replicas of desired files.

Our hope is that by exposing these and other applications to existing Internet users *along with* a clear path by which they can become full-fledged members of the DHT, HOLD can facilitate adoption of DHT-based systems. Using HOLD, researchers could seed a DHT with a small initial deployment and begin to gather data about their system driven by a large pool of potential users who might be attracted to these DHT-based applications while being unwilling or unable to install separate software.

HOLD achieves backwards-compatibility via an

*jconsidi@cs.bu.edu, Boston University
†{mwalfish,dga}@lcs.mit.edu, MIT

[1] "Structured peer-to-peer overlays" [3] precisely refers to the academic peer-to-peer systems whose lowest layer performs KBR. For brevity, we follow the convention in [7] and use the overloaded "DHT" to refer to these systems, which include DOLRs, proper DHTs, and CAST abstractions [3]. HOLD's focus is on the common, lower-level function of key-based routing.

efficient and effective mapping from DHT lookups to DNS queries: Internet hosts actually perform lookups in the same iterative fashion as full DHT members, but they do so using DNS queries to contact each DHT member along a routing path. We find that many popular DHTs can be mapped this way, allowing researchers to use this approach as a front-end to existing systems.

By pushing the burden of iteration to clients, HOLD nodes use less state and bandwidth compared to traditional proxy-based solutions. This difference is similar to the scale difference between a local recursive name server—which typically handles thousands of clients—and the root name servers, which scale to millions of clients by providing only stateless, iterative responses and by depending on client caching. We believe the same effect will apply to the HOLD approach, making it more scalable, load-balanced and resilient than a proxy-based solution.

HOLD maintains a pragmatic focus to facilitate incremental DHT deployment, obviously trading some of the benefits of a pure peer-to-peer approach for easier adoption. The purpose of this paper is to investigate HOLD, to discuss applications that could use HOLD, and to understand the trade-offs that accompany HOLD. We find, perhaps surprisingly, that HOLD is practical and that, for certain applications, it can provide many peer-to-peer features, such as robustness and load balancing. In light of these benefits, we believe that the hybrid approach employed by HOLD, with its attendant compromise of the strictly "P2P" model, merits further consideration by the wider peer-to-peer community.

# 2 HOLD: Mapping DHTs into DNS

Our imagined scenario is as follows: researchers set up a DHT that supports HOLD. They then establish one or more domain names like `mydht.net` that point to a subset of the DHT participants. External clients perform ordinary DNS lookups (perhaps resulting from following an HTTP URL) on domain names ending in `mydht.net` to locate a DHT node responsible for the given identifier.

Before continuing, we emphasize that HOLD uses DNS only as an adoption mechanism for DHTs. HOLD does not provide traditional DNS service (in contrast with [2]), and we do not claim that HOLD inherits DNS's usual features or its performance.

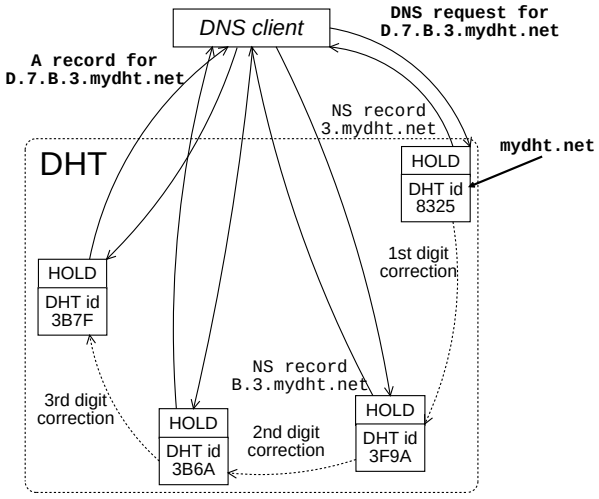


Figure 1: Finding the node responsible for identifier $x$ = `3B7D` in a tree-based geometry.

## 2.1 Architecture

The software on the DHT nodes includes a DNS name server that provides the interface between the DHT and non-participant clients. The DNS server replies to queries by consulting the DHT's neighbor list and dynamically returning an answer that points the requester to the next routing hop (via an `NS` record) or to the final owner node (via an `A` record). To identify the DHT node responsible for an identifier $x$, an Internet client first does a DNS lookup on an appropriate name $n(x)$ (we explain the mapping $n(\cdot)$ below); is then iteratively routed through the DHT via NS records; and ultimately receives an A-record with the IP address of the DHT host responsible for $x$. Figure 1 illustrates this process.

## 2.2 Mapping Identifiers to Domain Names

The map from an identifier $x$ to a domain name $n(x)$ explicitly separates the identifier into digits of a given radix; each digit corresponds to a sub-domain. For example, the identifier 3B7D (in radix 16) would be encoded as `D.7.B.3.mydht.net`. Although any DHT can in principle use HOLD, those using tree-based geometries [7], not surprisingly, map quite simply into DNS's strict hierarchy. Two such popular DHTs are Tapestry [16] and Pastry [13], both based on work by Plaxton *et al.* [12]. In these topologies, each DHT routing step corresponds to retrieving an NS record for a sub-domain. DHTs based on de Bruijn graphs [4], such as Koorde [10], also map simply into DNS.

Routing under tree-based geometries reduces to finding a longest prefix match with the identifiers of known peers. After $k$ steps starting from any DHT node, the current node and the node responsible for a given identifier must match in at least the $k$ most significant digits. This yields a simple mapping into DNS names: a node $n$ with identifier $I$ (where $I$ is, *e.g.*, the hash of $n$'s IP address) can be responsible for the sub-domain $d_k.\cdots.d_1$`.mydht.net` if and only if $d_1 \ldots d_k$ is a prefix of $I$. The embedding of de Bruijn graphs in the DNS hierarchy is nearly identical, except now $N$ is responsible for the same sub-domain if and only if $d_1 \ldots d_k$ is a *suffix* of $I$.

When a client resolves `D.7.B.3.mydht.net` in a tree or de Bruijn geometry, it sends a query to the `mydht.net` name servers for `D.7.B.3.mydht.net`. `mydht.net` responds with an NS record associating `3.mydht.net` to the IP address of any DHT node whose identifier begins with 3. The particular node returned depends on the contents of `mydht.net`'s DHT routing table. The client then iteratively continues to resolve its query via the `3.mydht.net` node, and so on.

We have designed similar mappings for other topologies, including those of Kelips [9] and the "one-hop" scheme proposed by Gupta *et al.* [8]; they are elided for space. In addition, owing to the one-to-one correspondence between returned NS records and next hops in the DHT, existing DHT routing optimizations work without modification. We have not conducted an exhaustive study, but these compatibilities suggest that the HOLD approach is flexible.

### 2.3 Implementation Experience

To understand the practical aspects of HOLD, we built a rudimentary prototype with a routing discipline similar to Pastry's. Using existing components for DNS packet processing, we built a small server that handles requests for a fictitious domain. The server uses the DNS protocol for all of its operations, including topology maintenance. The HOLD prototype either directly listens to the well-known DNS port (53) or relies on a server like BIND that can be configured to forward traffic to the correct HOLD process after examining only the suffix of the queried domain name. Thus, multiple HOLD applications and an existing DNS server can share a port. Our implementation uses radix 16. Small radices result in longer domain names; *e.g.*, radix two implies 320 bytes to encode the identifier alone, which rapidly approaches DNS's 512 byte maximum packet size. Our implementation also depends on DNS's name compression to fit each message into one packet.

Given our experience with this prototype, we hypothesize that developing HOLD for other DHT designs would be straightforward and quick, with the caveat that designing any DNS-intensive application requires some care—each DNS resolver has its own idiosyncracies that must be handled. We plan to validate this hypothesis by building a production version of HOLD for several existing DHT designs and by implementing the applications we describe next.

## 3 Case Studies

Although the single service that HOLD exports to legacy users—key-based routing—is only one of many DHT features, this function nonetheless supports several worthwhile applications.

### 3.1 Blocking Unsolicited Bulk Email

DNS-based block lists (DNSBLs) are highly effective for combating spam. DNSBL publishers collect IP addresses from known spam sources and respond to DNS queries like `8.7.6.5.bl.spamcop.net`. In this case, the DNSBL server returns an `A` record if it believes the IP address 5.6.7.8 is a source of spam. Mail servers (*e.g.*, sendmail) and client-side software (*e.g.*, spamassassin) are easily configured to use one or more DNSBLs.

Because DNSBLs are so effective and because their policies for "collateral damage" are controversial (*e.g.*, listing any addresses belonging to an ISP that hosts a spammer), DNSBLs are frequently victimized by Distributed Denial of Service (DDoS) attacks. For example, in a well-publicized departure, the Monkeys.com DNSBL went permanently off-line in September 2003 due to sustained DDoS attacks [6]; the Spamhaus project was also the target of a worm-born DDoS attack [14].

Maintaining DNSBLs in a DHT is an obvious solution to the single point of failure in conventional DNSBLs. Moreover, by their very nature, DNSBLs respond to simple queries using DNS, exactly the query mechanism that HOLD uses. Hence, a DHT-based DNSBL (DBBD) is a perfect application for HOLD. We now walk through an example of a `sendmail` server performing a lookup with DBBD.

- When a `sendmail` server receives an e-mail from IP address 1.2.3.4, it checks whether an `A` record or an `MX` record (Mail eXchange) for `4.3.2.1.dbbd.net` exists.
- To improve load-balancing, the root server for `dbbd.net` hashes 1.2.3.4 and responds that `4.3.2.1.dbbd.net` is actually an alias for the canonical name $4.3.2.1.h_0.\cdots.h_{d-1}$.`dbbd.net`.
- The client automatically queries this canonical name using HOLD until it reaches the name server for $h_0.\cdots.h_{d-1}$.`dbbd.net`.
- If this name server has 1.2.3.4 in its DHT, it returns an `A` record for the canonical name. Otherwise, it replies that no such record exists.

The above scheme works transparently for legacy clients over existing protocols, but it does not provide the full resilience of a DHT; in particular, the `dbbd.net` nodes themselves are points of failure and possible attack targets. Because *any* node can act as the root, however, a site wanting the full resilience provided by the underlying DHT could have a local node join the DHT and redirect all DNS requests within `dbbd.net` to that node.

### 3.2 File Sharing

File sharing typically involves two steps: (1) Given a set of search keywords, locate a file identifier. (2) Find replicas of the specific identified file within the network—a common problem in peer-to-peer systems. HOLD can export this second function in a backwards-compatible manner to Web clients.

Content distributors could expose a Web URL like: `http://`$d_0.\cdots.d_{k-1}$`.cdn.net/get/file`. This URL allows any Internet user to locate copies of the object with identifier $d_{k-1}.\cdots.d_0$ by simply clicking on a standard Web link. Unlike approaches that use external software to download the link, this technique works entirely within the existing browser. This URL is a persistent object identifier and thus invariant under changes in DHT membership.

## 4 Discussion

Ideally, HOLD would provide the best of both worlds—backwards compatibility with existing clients, and the resilience and performance benefits of a full DHT solution. This section discusses some of the benefits and drawbacks arising from design compromises that make HOLD work in practice.

**HOLD inherits the DHT's self-configuration** Unlike conventional DNS systems that require manual configuration when nodes are added or removed from a domain, HOLD is more resilient: it inherits the self-healing and load-balancing properties of the DHT and automatically reconfigures itself when DHT nodes join and leave.

**Mapping to a hierarchical system involves low overhead** Lookups using HOLD use almost the same number of lightweight DNS queries as corresponding DHT lookups would have used RPCs. HOLD introduces an additive constant because uncached HOLD lookups begin at the DNS root.

**Using DNS requires DHT software changes** The HOLD front-end that allows DHT nodes to provide DNS services naturally requires additional code. Because DNS uses a privileged and often firewalled port, not all DHT members will be able to export the HOLD front-end. Existing peer-to-peer systems handle similar problems due to NATs and firewalls; changes to the HOLD mapping to adapt to firewalls are possible but beyond this paper's scope.

**DNS caching improves performance but may slow fail-over** A root server for the DHT sheds load quickly via DNS caching. In a radix 16 system, the root need only hand out delegations to 16 different servers, one for each digit; these answers, even if they have low TTL, will be rapidly cached and reused by busy clients. A downside to DNS's caching is that inappropriate TTLs could make the system slower to respond to failures, since clients would continue trying cached `NS` records. Determining appropriate TTLs in this milieu is future work.

**HOLD is weakly centralized** HOLD's main (and unavoidable) weakness is that, from a client's perspective, the DHT front-end is attached to the DNS at a static and possibly vulnerable address. Fortunately, the symmetry of the underlying DHT means that each member running the HOLD software can be the "root" for client requests. This permits, for example, an administrator to establish a standard `NS` record for `mydht.net` containing, say, 10 different IP addresses referring to 10 different DHT nodes (which are hopefully relatively reliable). As we saw in the spam block list application in Section 3.1, organizations or individuals can easily run their own "root" by simply installing the HOLD software. The

ability to have multiple roots greatly improves reliability and load balancing.

**Security** The security of HOLD-based systems is primarily dependent on open research issues of security in DHTs. Because it does not expose the DHT's security features to clients, HOLD inherits some of the weaknesses of the underlying DNS system. We would like to explore the applicability of DNSSEC to improving HOLD but must leave it, and the underlying DHT trust issues, for future work.

# 5 Related Work

Akamai [15] also uses DNS, but does so to direct clients to nearby caches rather than to expose a key-based routing service. Zhou *et al.* [17] apply peer-to-peer techniques to combating spam by identifying characteristics of the *message body*. We instead attempt to provide a traditional DNSBL service, identifying spam *sources*. Several projects use DHTs as the storage and routing of a content distribution network of proxies; examples include CoDeeN [11] and Coral [5]. These projects expose a front-end proxy permitting Web clients to access CDN-based storage and retrieval. HOLD compliments these approaches by exporting key lookup to clients in cases when copying data through the CDN nodes is undesirable. Freenet [1] also exposes its file sharing services via a Web interface to legacy clients.

# 6 Summary and Future Work

HOLD takes a conceptually impure approach to building adoptable peer-to-peer systems, sacrificing some benefits to provide a practical migration path for users. With HOLD, people can use a peer-to-peer system before installing software. Our positive experience with the HOLD prototype, and its applicability to two example systems, makes us believe that the HOLD approach is worthwhile. We plan to augment the prototype and to implement these HOLD-based applications. We hope that HOLD's techniques, and its attendant pragmatic compromises, can facilitate the adoption of other peer-to-peer applications.

# References


[1] I. Clarke, O. Sandbert, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[2] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a peer-to-peer lookup service. In *Proc. 1st IPTPS*, Cambridge, MA, Mar. 2002.

[3] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured peer-to-peer overlays. In *Proc. 2nd IPTPS*, Berkeley, CA, Feb. 2003.

[4] N. de Bruijn. A combinatorial problem. *Proc. Koninklijke Nederlandse Akademie van Wetenschappen*, 49:758–764, 1946.

[5] M. J. Freedman, E. Freudenthal, and D. Mazières. Coral: The NYU Distribution Network. `http://www.scs.cs.nyu.edu/coral/`, 2003.

[6] R. F. Guilmette. ANNOUNCE: MONKEYS.COM: Now retired from spam fighting. newsgroup posting: news.admin.net-abuse.email, Sept. 2003.

[7] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003.

[8] A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Proc. 9th Workshop on Hot Topics in Operating Systems*, Lihue, Hawaii, May 2003.

[9] I. Gupta, K. Birman, P. Linka, A. Demers, and R. van Renesse. Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proc. 2nd IPTPS*, Berkeley, CA, Feb. 2003.

[10] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proc. 2nd IPTPS*, Berkeley, CA, Feb. 2003.

[11] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the Web: An open proxy's view. In *Proc. 2nd Workshop on Hot Topics in Networking*, Nov. 2003.

[12] C. G. Plaxton, R. Rajaraman, and A. W. Richea. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. SPAA*, June 1997.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-s cale peer-to-peer systems. In *Proc. Middleware*, Nov. 2001.

[14] The Spamhaus Project. Spammers Release Virus to Attack Spamhaus.org. `http://www.spamhaus.org/news.lasso?article=13`, Nov. 2003.

[15] Akamai. `http://www.akamai.com`, 1999.

[16] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE JSAC*, Nov. 2003.

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