Distributed Hash Tables: Design and Applications

C.-F. Michael Chan and S.-H. Gary Chan

Abstract The tremendous growth of the Internet and large-scale applications such as file sharing and multimedia streaming require the support of efficient search on objects. Peer-to-peer approaches have been proposed to provide this search mechanism scalably. One such approach is the distributed hash table (DHT), a scalable, efficient, robust and self-organizing routing overlay suitable for Internet-size deployment. In this chapter, we discuss how scalable routing is achieved under node dynamics in DHTs. We also present several applications which illustrate the power of DHTs in enabling large-scale peer-to-peer applications. Since wireless networks are becoming increasingly popular, we also discuss the issues of deploying DHTs and various solutions in such networks.

1 Introduction

The Internet has grown to an enormous size, with nearly 600 million hosts as of early 2008 [1]. Coupled with this intense growth in network size is the proliferation of large-scale applications such as file sharing and multimedia streaming. These applications require the support of fast search on objects. Since traditional server-client model is no longer scalable to large group of hosts, peer-to-peer approaches have been proposed. One of such approaches is the distributed hash table (DHT), a scalable, efficient, robust and self-organizing overlay routing infrastructure for millions of hosts. DHTs have the potential to enable large-scale peer-to-peer applications, such as distributed file systems and on-demand video streaming. This chapter introduces DHTs and discusses their design issues and applications.

C.-F. Michael Chan
Stanford University, Stanford, CA, USA, e-mail: mcfchan@stanford.edu

S.-H. Gary Chan
Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, e-mail: gchan@cse.ust.hk

X. Shen et al. (eds.), *Handbook of Peer-to-Peer Networking,*
DOI 10.1007/978-0-387-09751-0_10, © Springer Science+Business Media, LLC 2010

DHT provides a unified platform for managing application data. Due to the specific nature of application data, traditionally a protocol designed for one type of application may not work well for other types of applications. DHTs break this barrier by providing a flat identifier-based routing and location framework, and a very simple API to applications.

DHT maps application data to keys, which are m-bit identifiers drawn from the identifier space. Nodes participating in the DHT are distinguished by unique identifiers drawn also from the same identifier space. Each node is responsible for a subset of the space, i.e. stores a subset of keys. Typically, a value is associated with a key, and is also stored at the node responsible for the key. Depending on the specific application, the value could be the address of the node storing the data or the data itself.

A DHT scheme defines how the overlay is structured, how node state is maintained and how routing is carried out. Regardless of the details, DHTs provide a two-method interface for applications:

- *insert(k, v):* Insert a data item with key-value pair (k,v) into the DHT.
- *lookup(k):* Retrieve the value v associated with key k. Return *null* if k is not found.

In contrast to IP routing, DHT routing is identifier-based. Each node stores about $O(\log n)$ overlay neighbors and employs a deterministic algorithm to route queries from requestor to the node storing the target key in $O(\log n)$ overlay hops. In Section 3, we discuss various DHT schemes. In particular, we shall see how the overlay is structured for efficient and scalable routing. We also outline techniques used to improve a DHT's query response time and robustness.

There are many applications making use of DHT. We present several examples that demonstrate how DHTs enable large-scale peer-to-peer services.

Another interesting DHT topic is its deployment in wireless networks which are becoming increasingly popular nowadays. Applications are being developed for mobile ad-hoc networks (MANETs), wireless sensor networks (WSNs) and wireless mesh networks (WMNs). It is desirable to have a DHT-like framework for wireless networks to support object location. We describe some major challenges in deploying DHT in wireless networks, and outline some solutions.

This chapter is organized as follows. We present the performance characteristics and design considerations of DHTs in Section 2. We discuss some examples of DHTs and how DHT is used in large-scale applications in Sections 3 and 5 respectively. In Section 6, we highlight the challenges of applying DHTs in wireless networks and present some solutions. We conclude in Section 7.

2 Performance Characteristics and Design Considerations

In this section, we describe some common performance characteristics of DHTs, and discuss some design issues (Section 2.1), followed by design considerations of DHT (Section 2.2).

2.1 Common Performance Characteristics

A DHT usually has the following desirable properties:

1. *Efficiency in routing:* A DHT overlay is structured so that queries for keys may be resolved quickly. Typical DHT schemes have a $O(\log n)$ bound on the length of search path (in overlay hops). To reduce routing delay, in recent years, locality-aware DHTs have been proposed so that routing hops are of short Internet length by taking advantage of locality information.
2. *Scalability:* Node storage and maintenance overhead grows only logarithmically with the number of nodes in DHT. This leads to its high scalability to large number of users.
3. *Self-organization:* A DHT protocol is fully distributed. Node joins, departures and failures are handled automatically without the need of any central coordination.
4. *Incremental deployability:* A DHT overlay works for arbitrary number of nodes and adapts itself as the number of nodes changes. It is functional even with one node. This is a highly desirable feature as it enables deployment without interrupting normal operations when new nodes join the overlay.
5. *Robustness against node dynamics:* Queries are resolved with high probability even under node dynamics. Further optimizations may further increase system robustness.

2.2 Design Considerations

In designing a DHT to achieve the above desirable properties, several issues need to be considered:

1. *Node dynamics:* Peers may join and leave at any rate. Since it is not possible for a central server to record the status of each peer, a DHT must be able to dynamic node departure or failure. In other words, the DHT structure must be maintained to ensure correct and efficient routing in the presence of node dynamics.
2. *Overlay path stretch:* Compared with IP routing, overlay routing to a certain node has in general higher latency, since overlapping traversals of some physical links is inevitable. The path stretch is defined as the ratio of the overlay route's latency to the underlying IP route's latency. In order to avoid large path stretch, DHT routes need to be optimized for low response time and such optimization techniques should be scalable to large groups.
3. *Hotspots:* Application data is generally skewed in terms of access probability. For instance, in video streaming, some segment of a video may be more popular than the other. A DHT scheme needs to consider the high lookups for a particular key (i.e., popular segment) by load balancing request processing among many nodes. The load balancing algorithm should be scalable.

3 DHT Schemes

We describe several DHT schemes in this section. Note that keys and node IDs are drawn from the same m-bit identifier space and that keys are typically obtained by hashing meta-data, while node IDs are hashes of IP addresses or public keys. For a good hashing functions, nodes and keys are uniformly distributed in the overlay. Consequently, each node stores a similar share of keys. In our discussion below, we focus on operations including how keys are inserted, stored and looked up, and how the overlay is constructed and maintained. We assume m-bit identifiers are used, i.e. the identifier space is $[0, 2^m - 1]$. We denote a node with ID i as N_i and a key with ID j as K_j.[1]

3.1 Chord

The Chord DHT places nodes and keys in an identifier ring [23]. It is a simple DHT with great flexibility, which allows optimizations such as load balancing to be readily added as extensions to the basic scheme. We first describe the basic Chord, and then briefly discuss some simple but important extensions. The basic components and operations of Chord are as follows:

- *Key placement:* A key K_j is stored by the node N_i immediately following j in the identifier ring. In other words, N_i is chosen such that there is no node $N_{i'}$ where $j \leq i' < i$. We also call N_i the *successor* of j, denoted by $successor(j)$. Note that key and node IDs may coincide, in which case the key (K_i) is stored at the node with the same ID (N_i).
- *Successor Links:* Each node N_i maintains a link to its successor, the node N_j immediately following it. Such definition of successors for keys is also applicable to nodes. Denote N_j as $successor(i)$. As long as successor links are correct, a lookup is guaranteed to reach the key's successor, albeit via a long path going around the identifier ring one node at a time. A node achieves this by periodically running the following $stabilize()$ procedure. It asks its successor N_s for N_s's predecessor N_j. If $N_j \neq N_i$, N_i sets N_j as its successor. This happens if N_j is a newly joined node. Suppose $i < j < s$ and N_i and N_s are existing nodes in the DHT. When N_j first joins the overlay, it looks up j and gets N_s's address. It then sets N_s as its successor. Finally, N_s transfers keys in $(i, j]$ to N_j. It is shown in [24] that all successor links always converge correctly for any sequence of node joins and stabilizations.
- *Fingers:* A node N_i also maintains a *finger table*, which contains m entries, where m is the identifier length. The j-th finger stores the address of $successor(i+2^{j-1})$. An example of finger tables with 6-bit identifiers is shown in Fig. 1. Nodes periodically run a $fix\_fingers()$ procedure to refresh the finger table entries. The

[1] For simplicity, we also use i in place of N_i and j in place of K_j when there is no ambiguity.

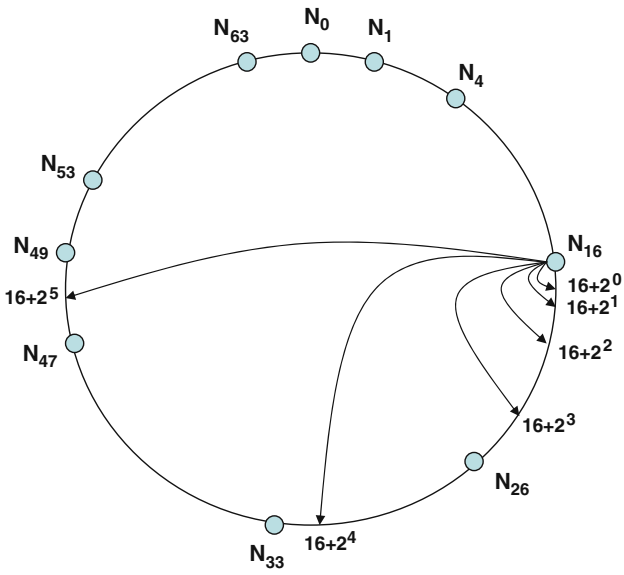


Fig. 1 Example of Chord fingers

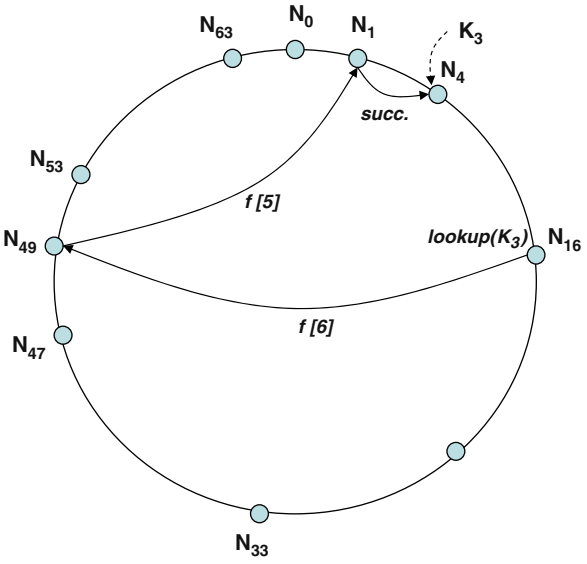


Fig. 2 Example of Chord routing with fingers. Each search hop is labeled with the finger table entry used in forwarding the lookup message. *succ* means the key lies within the segment between the current node and its successor

table is iterated in a round-robin fashion, where each call to the procedure refreshes the next finger. A refresh is achieved by looking up the finger's successor.

- *Key lookup:* The simple lookup algorithm forwards the request hop by hop using successor links until the key's successor is reached. The search path's length is $O(n)$ hops. Fingers drastically shorten the lookup path to $O(\log n)$ hops. In general, to look up key k, node i does the following. First, it checks if k is in $(i, successor(i)]$. If so, simply forward the request to the successor. Otherwise, forward the request to the largest finger $i + 2^{j-1}$ immediately preceding k. The same procedure is carried out at each intermediate node. Figure 2 shows an example of Chord routing with fingers. Notice that each hop (via a finger) reduces the distance to the key's successor by approximately half, thus giving the $O(\log n)$ path length. A more formal proof is given in [23].

The basic Chord is extended for greater robustness and load balancing. Two main ideas are key replication and employing virtual nodes. For key replication, instead of storing key k at only $successor(k)$, it is replicated on the r successors of k. This way, even if $j = successor(k)$ fails, the $successor(j)$ is still available for answering lookups for k. Setting $r = O(\log n)$ allows for robustness against very high degrees of node dynamics [23]. A node may also run multiple virtual nodes depending on its processing power and bandwidth. This way, heterogeneous peers may fully contribute their spare and varied resources to enhance the DHT's quality.

3.2 Pastry

In Pastry, an identifier is made of D digits. For example, a 128-bit identifier is broken up into 32 4-bit digits. To simplify the exposition, we assume 2^{bD}-bit identifiers, where b is the number of bits per digit. The components of Pastry are as follows:

- *Key placement:* A key k is placed at the node whose ID i is numerically closest to k.
- *Leafset:* Each node i keeps a list of L neighbors, where L is an even number. $L/2$ of those have IDs numerically closest to and smaller than i. The other $L/2$ have IDs numerically closest to and larger than i. This is similar to Chord's successor links in that a node may employ correct leafset information to resolve lookups in $O(n)$ hops.
- *Routing table:* A Pastry routing table consists of D rows, one for each digit. A row consists of 2^b entries. The j-th entry of the i-th row stores the address of a neighbor whose ID shares the same i significant digits with the current node's ID, but with the $i+1$-th digit equal j. Note there may be no node matching the requirements of some entries. In this case, the entry is left empty (Fig. 3).
- *Key lookup:* To look up a key k, a node i first checks its leafset. If a neighbor is numerically closest to k, the request is forwarded to that neighbor, and the lookup is complete. Otherwise, the routing table is checked for a node whose ID shares one more digit with k than i. If there is no such node (i.e., the route entry is empty), the message is forwarded to the node j matching as many digits with

Routing table of node 3123

Level *i*	*i+1* digit			
	0	1	2	3
0	0xxx	1xxx	2xxx	–
1	30xx	–	32xx	33xx
2	310x	311x	–	313x
3	3120	3121	3122	–

Fig. 3 Pastry routing table of node 3123. 8-bit identifiers are divided into 4 2-bit digits. All numbers are in base 4. "xxx" is an arbitrary string of base-4 numbers

k as i does, but is numerically closer to k. As shown in [22], node j exists with high probability given that L is reasonably large. A lookup fails, however, if $L/2$ nodes with consecutive IDs fail at the same time. Note that each hop effectively brings the lookup one-digit closer to the target key, thus the $O(\log n)$ bound on the search path length.

- *Node dynamics:* Node joins are handled similarly as in Chord. A new node looks up its own ID i. Suppose node j is responsible for the key i. Node i asks node j for its leafset and turns it into its own leafset by adding j and removing the node farthest away in terms of ID distance. Up to L nodes in j's leafset will need to be contacted so that they could update their leafsets to include i. j also updates its leafset to include i. The two adjacent neighbors in i's leafset then transfer keys to i.

  Node i's routing tables are populated by route entries from intermediate nodes along the search path it initiated when it looked up its own identifier. For each route entry, there may be multiple candidates. The entry with the lowest distance is chosen. The distance metric reflects the end-to-end delay between i and the neighbor. Once the route entries are filled, i asks for those neighbors for their routing tables, and updates the route entries again by replacing route entries with lower-distance entries. This way, locality-awareness is built-in during overlay construction. Reference [22] gives a formal proof that locality is preserved by using routing entries from physically close nodes.

  A node handles neighbor departures and failures lazily. Replacement of a failed neighbor occurs when the node forwards a message in vain to the neighbor. A failed leafset neighbor is replaced as follows. The node contacts the live leafset neighbor P with the smallest ID or the largest ID depending on which side of the node the failed neighbor resides in the leafset. P returns its leafset to the node, which then updates its own leafset by removing the failed neighbor and adding a new neighbor from P's leafset. The j-th entry of the i-th row is handled as follows. The node successively asks neighbors in the other i-th row entries for their j-th route entry. This way, a number of candidates for this particular entry is found for replacing the failed neighbor. The one closest in terms of the distance metric is chosen.

3.3 Kademlia

Kademlia is a widely implemented DHT [12, 16, 25]. It stands out among other DHTs by using an uncommon metric – the XOR metric. The distance between two nodes is defined as the bit-wise XOR of their identifiers. For instance, the distance between nodes with IDs 1100 and 0010 is 1110.

The following description of the Kademlia protocol is facilitated by a binary-tree view of the network. Each node is a leaf and the (labeled) path from the root to the leaf is the unique prefix of the node's ID. Figure 4 shows an example of this binary-tree representation of a Kademlia network.

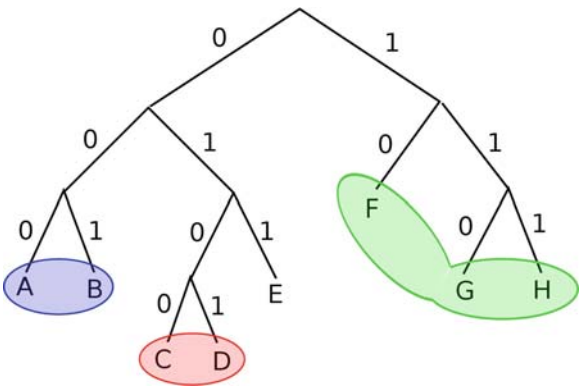


Fig. 4 Binary tree abstraction of a Kademlia network

- *Node state:* As in Chord and Pastry, Kademlia nodes keep track of peers in certain ID ranges. Peers are put into *k-buckets*, where bucket i is for peers at a distance between 2^i and 2^{i+1}. Each node must know of at least one node in each bucket, if there is some node in that ID range. For example, node E in Fig. 4 needs information of at least one node from each circled group. The size of a bucket is limited by the system parameter k. If a bucket is full, a new node will not be added unless the least-recently-used node in the bucket fails to respond. This modified LRU replacement scheme has the benefit of defending against malicious attempts to flush node state with new (and bogus) information.
- *Key placement:* A key is placed at the k nodes whose IDs are closest to the key.
- *Key lookup:* A key x is looked up as follows. The querying node n first scans stored information to locate the k nodes closest to x. It then sends the query to α ($< k$) nodes from the set. Upon receiving replies, n chooses again the k closest nodes to x, and then sends queries to α of them. The formal analysis of this algorithm is involved, but the idea is like walking down the binary tree from the root, where at each internal node, n queries some node it knows about in that subtree, receives information about a node further down the subtree and closer

to the destination, and repeats the process by moving down the correct edge. The algorithm is also similar to that of Pastry's prefix-based routing, where with each step, n gets closer to peers storing x by "correcting" significant bits in the nodes queried. The system parameter α governs the degree of parallelism in lookups. By employing parallel look ups, Kademlia sacrifices some (constant) increase in bandwidth for flexibility in selecting low-latency paths.

- *Node dynamics:* Unlike other DHTs, nodes in Kademlia learn about each other through queries. When a node joins the network, it looks up its own ID, so that peers along the query paths know about it. Similarly, there is no explicit messaging when a node leaves. The departure is only discovered when other nodes attempt to ascertain the node's presence before evicting it from a bucket. There is also the need for republishing keys to ensure that they are stored at the k closest nodes. Periodically, a key is republished (by a lookup for the key and data transfer to new holders). To reduce overhead, a node receiving this republication will not republish the key in the next period. This way, as long as some node first republishes the key, other nodes who were also obliged to do so would not need to waste the bandwidth to perform the same operations.

In practice, further optimizations are employed to speed up the protocol. When a bucket is full, new information for that bucket is cached. If some node in the bucket fails afterwards, the cached information can be used to fill in the gap immediately. Lookups are sped up by expanding the routing table so that multiple bits may be matched in each step. This is similar to setting $b > 1$ in Pastry.

3.4 Other DHTs

Several DHTs based on other types of overlay structures. The Content Addressable Network (CAN) constructs a d-dimensional hypercube [20]. Identifiers are divided into d digits. Each node owns a subset of the coordinate space and maintains a set of $d-1$ neighbors. A neighbor shares the same range of values with the node in $d-1$ dimensions, but not in the remaining dimension. Lookups are resolved by forwarding along the correct dimensions such that each hop matches one more digit in the key. A lookup is thus resolved in $O(d)$ hops. Note that by setting $d = \log n$, we get the $O(\log n)$ bound as in Chord and Pastry.

Tapestry is similar to Pastry in that it constructs a prefix-based routing table [29]. The main difference is that there is no leafset.

Viceroy maintains a butterfly structure [15]. Key management is similar to that in Chord. Nodes are distributed on an identifier ring. The difference is in the overlay neighbor selection. A node joins one of $\log n$ level rings. Each level ring is connected by level-ring links between nodes with adjacent identifiers on the same level. A node at level i maintains some pointers to neighbors at levels $i-1$ and $i+1$ in a way that lookups are resolved in $O(\log n)$ hops. Viceroy is most characteristic in its constant node state.

4 Design Fundamentals

In this section, we discuss some fundamental design issues. With so many DHT protocols, it is natural to ask how they fair against one another in various scenarios. A detailed comparison between the various DHT schemes is presented in [9], which studies the effect of the overlay's structure on system performance in terms of static resilience, path latency and local convergence. We also briefly discuss interesting findings concerning tradeoffs between the network diameter of a DHT and the amount of routing information a node stores [26].

4.1 Static Resilience

The robustness of a DHT depends on both its recovery mechanisms and structural properties. The study of static resilience reveals how well a DHT's structure copes with node failures without any recovery operations, and sheds light on how such mechanisms should be provisioned. For instance, if the structure is inherently resilient against node failures, then recovery mechanisms need not be frequently carried out.

Intuitively, a DHT is more robust when there are more alternative routes a query can take to the key holder. If some nodes fail, it is still possible to route around the failures to the destination. Clearly, the DHT's routing algorithm dictates how flexible queries may be forwarded. The more flexible the DHT's structure, the more robust the system.

Take Chord and Pastry as examples. In the original Chord proposal, a query is resolved by halving by the distance to the destination at each step. However, this is really a restricted version of the more general algorithm in which the distances covered by each step need not be in decreasing order. In other words, we may take the steps in any order, as long as there is one step for each distance (2^i) to be covered. The prefix-based phase of Pastry's routing algorithm is vastly different. At each step, we must fix the most significant unmatched bit. This means only one route entry can be used. If the node in that entry has failed, the query has to be redirected to leafset neighbors. Clearly, if not for the leaf sets, Pastry's routing algorithm results in little flexibility in routing, and hence lower static resilience.

4.2 Path Latency

The structure's flexibility also affects the latency of query paths. Once the routing table is fixed, there is limited choice in a query's next hop. A flexible structure enables selection of neighbors with better latency, and hence better choices in choosing next hops. As a result, the overall path latency can be reduced.

In Chord, a node n's i-th finger is originally defined to be the successor of $n+2^i$. However, this requirement can be relaxed to include nodes between $[n+2^i, n+2^{i+1})$, making neighbor selection very flexible. In Pastry, a route entry in the i-th row can be chosen from a maximum of $2^{b(D-i-1)}$ peers. A Kademlia node also has great flexibility in filling its k-buckets. For the i-th bucket, there are up to 2^i choices. On the other hand, CAN requires a neighbor to only differ by a single-bit, thereby restricting the selection to one neighbor.

4.3 Local Convergence

Consider two queries from two nodes that are nearby in the physical network. A DHT exhibits good local convergence properties if the queries converge at a common peer that is also close to the two nodes. The most significant advantage of a high degree of local convergence is that caching can be made more effective. Frequently, an application on top of DHT caches lookup results along the query path in hopes that future queries would hit the cached entries early in their lookup paths. The Cooperative File Storage (CFS) to be discussed in Section 5.1 is an example of such an application.

It is discovered that a structure with flexible neighbor selection exhibits good local convergence properties. Chord, Pastry and Kademlia with low-latency neighbor preference are desirable designs in this regard.

4.4 Network Diameter and Node State Tradeoffs

Another interesting observation is that for most of the DHTs we have discussed, the size of a node's routing table is either $O(\log n)$ (Chord, Pastry, Kademlia, Tapestry) or $O(d)$ (e.g. CAN), with network diameters of $O(\log n)$ and $O(n^{1/d})$ respectively. (The exception is Viceroy, which has constant node state with high probability and $O(\log n)$ diameter). The question asked in [26] is whether these are actually lower bounds on the network diameter, i.e. $\Omega(\log n)$ and $\Omega(n^{1/d})$. The answer will give us an idea of whether the tradeoff has been optimized by existing DHTs, and whether there is room for improvement.

It is, in fact, possible to go beyond the $\Omega(\log n)$ lower bound. The idea is to construct a (directed) $\log n$-ary tree. Each node except the root has a edge pointing back to the root. This way, all nodes are reachable from one another (thus satisfying routing correctness) and any path between two nodes is at most $h+1$, where $h = \log_{\log n} n = \frac{\log n}{\log \log n}$ is the height of the tree. Obviously the longest path (i.e., the network diameter) is $O(\frac{\log n}{\log \log n})$, which is asymptotically smaller than $O(\log n)$. Also, each node only maintains $O(\log n)$ neighbors. Unfortunately, this structure puts the root under heavy congestion, an unacceptable state of affairs in peer-to-peer systems.

Another discovery is that DHT routing algorithms can be classified into two major categories – uniform and others. Uniform routing algorithms treat every query without discrimination against the source of the query and the location of the processing node. For instance, a Chord node would process a query by looking up its finger table no matter where the query came from or where the node resides in the identifier ring. In this case, where nodes have $O(\log n)$ state, $\Omega(\log n)$ is indeed the lower bound on the network diameter. The reason is that the DHTs aim to provide a uniform load across nodes (i.e., to avoid congestion), therefore they cannot achieve the lower bound of the $\log n$-ary tree for network diameter, but only the larger lower bound of $\Omega(\log n)$.

It turns out that a butterfly network can be (deterministically) designed to achieve a $O(\frac{\log n}{\log \log n})$ diameter with $O(\log n)$ node state and a non-uniform routing algorithm such that no extra load is imposed on any node. However, this bound is known so far to be possible with static nodes, and that some links would still have $O(\log n)$ extra load. It remains unknown whether the bound can be achieved deterministically in the presence of node dynamics.

5 Applications

In this section, we present several applications of DHTs. We first describe the application briefly, and then discuss their properties with reference to those of DHTs.

5.1 Cooperative File Storage (CFS)

CFS is a large-scale distributed storage system supporting multiple file systems [7]. Files are divided into blocks, which are stored in CFS servers. Blocks are distinguished by unique IDs obtained by hashing block contents and public keys of block publishers. Clients read a file by looking up and retrieving blocks that make up the file. File publishers insert and periodically refresh files so that the blocks do not get deleted by CFS servers after a fixed expiry time. Each file system is a tree rooted at a *root block*, which is signed by the publisher with its private key. The root block's ID is the publisher's public key. Clients then name a file system according to the system publisher's public key.

CFS consists of three layers. The (lowest) *Chord layer* is responsible for looking up blocks given their IDs. A *DHash layer* is built on top of the Chord layer. It manages block storage, replication and caching. The *file system layer* converts blocks obtained from the DHash layer to files and provides users and applications a file system interface. We focus on how the Chord and DHash layers collaboratively gives CFS various desirable performance qualities:

- *Highly scalable:* CFS inherits scalability from the underlying Chord layer. Node state and control overhead in searching and routing table maintenance

grows logarithmically with the number of CFS servers. Since files are broken into blocks, a large file will not exhaust a particular server's storage, as will be the case in file-based storage systems. Also, since block IDs are hashes of block contents, and servers are distributed approximately uniformly around the Chord ring, it is expected that each server stores around the average number of blocks.

- *Robust to massive server failures:* With the original Chord protocol, a block is stored at the successor of its ID. The DHash layer improves robustness by replicating the block in the k successors of the ID. Since consecutive servers on the overlay are likely to be distributed in the physical network, their failure rates are likely to be indepedent. If a server fails with probability p, then a block is inaccessible (removed from CFS due to failure of servers storing it) with probability approximately equal to p^k, which could be made small with modest values of k.
- *Effective load balancing:* A popular file may put excessive load on its server. CFS proposes two approaches to balance the load for popular files. First, for large popular files, CFS inherently balances the load by breaking them into blocks. Furthermore, block replication allows lookups to be directed to a number of servers instead of the immediate successor of the block ID. Second, for small popular files (and blocks), caching is employed to reduce the load on the block successors. In a nutshell, the queried blocks are cached at intermediate nodes along the overlay search path. Since the Chord lookup algorithm halves the distance to the target node every hop, cached copies near the target node tend to overlap, thereby boosting the cache hit rate.
- *Fast file access:* The Chord searching algorithm is improved to provide server selection in CFS. At every hop during lookup, a node may choose among the set of successors and fingers to forward the query to. Two potentially conflicting considerations are latency to the next hop and the ID space covered by this forwarding. A hybrid metric balances the latency and overlay progress (see Section 4.3 of [7] for details.) With server selection enabled, CFS retrieval rates are comparable to and has lower variance than that of direct TCP-based transfers such as FTP.
- *Secure against data change and flooding attacks:* Two major security concerns of distributed file systems are unauthorized data modification and flooding attacks. CFS servers mandates node ID to be the SHA-1 hash of the node's IP address, which is in turn authenticated via challenges with random nounces sent to the claimed address. An attacker wishing to modify a specific data block would have to control a large set of IP addresses to be able to store the target block. However, entities owning many IP addresses are more easily identified (such as big organizations) than individual attackers owning a small number of addresses. This way, the system is quite secure against intentional deletion of data. CFS also limits the amount of data publishers can put into the system to guard against flooding attacks. Suppose the quota is f, a (small) fraction of the total storage space in the CFS system, then an attack would have to employ $1/f$ hosts to exhaust the system's storage.

5.2 Scribe

Enabling large-scale multicast in the Internet is an important topic of research. Traditional (IP) multicast has been proposed long ago, but has seen limited deployment on a large scale due to network management issues and other deployment concerns. To support large-scale multicast, all networks involved must have multicast support enabled, which is often not possible. There are also issues of assigning (unique) multicast addresses to groups and authorization of operations such as group creation, sending messages to a group and receiving messages. The lack of multicast support has stirred interest in application-layer multicast (ALM) where end-hosts provide multicast by relaying messages at the application layer.

Scribe is an ALM scheme based on the Pastry DHT and solves several important issues with IP multicast [5]. While ALM is in general less efficient in terms of packet delivery time and link stress, Scribe provides scalable multicast with acceptable delay penalty and link stress as compared to IP multicast. In Scribe, a group ID (multicast addresses) is generated by hashing the creator address and group name. The creator then sends a create message to the node responsible for the group ID via Pastry routing. This node is the rendezvous point (RP) for node joins and message sending for the group. A joining node sends a join message with the target group ID to the RP. Intermediate nodes on the path become forwarders in the resultant multicast tree. They store a children list so that future multicast messages may be forwarded to the downstream. Senders forward packets to the root for dissemination throughout the tree. We focus our discussion on Scribe's scalability, efficiency in propagating multicast messages and fault tolerance.

- *Scalability in group size:* An important hurdle in scaling multicast to large groups is the node joining process. In particular, it is essential to enable nodes to join with low control overhead. This is obviously not possible with a centralized server storing a (sub)set of existing nodes in the group. Scribe enables efficient node joins by exploiting Pastry's properties. First, Pastry's routing mechanism allows the join overhead to be distributed evenly among nodes. Second, the RP need not handle all joins, as join messages may hit an existing node (either a forwarder or receiver) along the path and processed there locally. Third, the uniform distribution of nodes in Pastry ensures a balanced multicast tree. Furthermore, since Pastry employs prefix-based routing, the multicast structure is guaranteed to be loop-free.
- *Scalability in number of groups:* Two problems that hinder scalability in supporting many multicast groups are the assignment of group addresses and load balancing control information of groups among participating nodes. The address assignment problem is solved by employing DHT keys as group IDs, which are obtained by hashing the group creator and group's name. With a suitable hash function, the probability of group ID collision is low. This way, group IDs can be assigned in a totally distributed manner. Hashing also distributes the group roots uniformly in the Pastry overlay. Since nodes are also distributed uniformly in the overlay, the control information for all groups, i.e. parent and children pointers,

is likely to be distributed uniformly among the nodes. Indeed, for fairly large networks (100000 nodes and 1500 groups), Scribe nodes have on average less than 10 children pointers. Additionally, nodes may offload children connections by asking some of them to connect to their siblings instead. This so-called *bottleneck remover* algorithm dynamically adapts node stress to network conditions, and effectively reduces the maximum number of children connections under the same network conditions.

- *Multicast efficiency:* Scribe achieves low delay penalty compared to IP multicast for two reasons. First, the paths from receivers to the RP are short – with DHT routing, the path length is logarithmic in number of nodes in the network. Second, since Pastry enforces locality in choosing overlay neighbors, each overlay hop in the multicast tree is likely to be short in terms of latency. These two factors contribute to an overall low delay in disseminating messages in the multicast trees.
- *Fault tolerance:* Scribe exploits efficient DHT routing to localize repairs of the multicast trees in case of node failures. If a node fails, its children can simply route a join message to the group ID to discover a new parent. Failure of the RP has a more detrimental effect on the multicast group since it stores, apart from children points, other control information such as authorization credentials, identity of the group creator and senders. Scribe replicates the state at the RP to its k closest nodes in the Pastry overlay. If the RP fails, its children use Pastry to find the new root, which is the closest overlay node of the failed RP. Since this new root already possesses the group state, normal operation may resume quickly.

5.3 VMesh

Multimedia streaming is one of the most popular services in the Internet. A particular type of multimedia streaming is video-on-demand (VoD), in which users may request for any video at any time. An important feature of VoD is user interactivity, i.e. users should be able to start viewing the video at any point, and may jump forward and backwards as they wish. VMesh provides scalable and efficient VoD with rich user interactivity based on DHT [27].

In VMesh, a video is divided into segments initially stored at the video server. A peer downloads some segments from the server to its local storage and searches for segments it does not own via a DHT and a video mesh. The stored segments are not removed even if the peer is not viewing them. Instead, the peer streams these segments to other peers.

A DHT is constructed to facilitate segment location. Each peer registers the segments they store by inserting keys into the DHT. A segment's key comprises three parts – the video ID, segment ID and segment owner's network location, in decreasing order of significance. The network location field encodes the node's location in the network via space filling curves, which provide a one-dimensional proximity

metric for parent selection during segment location streaming. Since the video and segment IDs occupy more significant bits, keys for the same segment from different peers are stored in the same region in the overlay (e.g., in a (small) arc of the Chord ring). With key replication, a node may answer segment queries with a list of parents owning the segment. Once the requesting peer receives this list, it could ask nearby parents to stream the segment.

Peers maintain in addition to the DHT a *video mesh*. In a nutshell, peers storing segment i maintains pointers to peers storing the next ($i+1$-th), previous ($i-1$-th) and the same (i-th) segments. These pointers are obtained by locating the segments using DHT and storing the addresses returned. Whenever a peer is about to exhaust its current segment, it asks its parents for the addresses of the peers holding the next segment and starts buffering it for better video continuity. When a peer wants to jump to a new (far-away) segment, it queries the DHT for the segment's owners. If the segment ID is close to the current segment's ID, the parent pointers are followed instead.

VMesh also addresses non-uniform segment popularity. Typically, certain segments of a movie are more popular and thus accessed more frequently. It is desirable to have more replica of popular segments for load balancing purposes. VMesh peers employ a distributed averaging algorithm to estimate the popularity and number of segment replica in the network, and adjust the segments they store accordingly [17].

The following are desirable properties of VMesh:

- *Scalability:* VMesh scales well to large number of peers. By having peers store and stream segments to other peers, the video server's workload is greatly reduced. The peer-to-peer segment location algorithm is locality-aware. This allows peers to find physically nearby parents, thereby reducing streaming overhead and latency. Popularity-aware adjustment in segment storage balances streaming load among a suitable number of peers, thereby avoiding hotspots for popular segments.
- *Robustness to parent failure:* Since a peer receives a list of parents as an answer to a DHT query, it may stream the segment in parallel from multiple (nearby) parents. If a parent fails, the other parents may share the failed parent's load while the peer searches for a new parent.
- *Efficient bootstrap and jumping:* VMesh provides low startup delay for newly joined peers and low delay in jumping to arbitrary positions in the video. Since segment location is performed as a DHT search, queries are resolved efficiently. Also, a new peer is given a list of parents with their location in the network. Such locality awareness allows the peer to choose nearby parents to reduce streaming delay. The same principle applies for jumping peers.

5.4 Internet Indirection Infrastructure (i3)

Unicast routing has been the main service provided by the network layer in the Internet. The need for large-scale multicast, anycast and host mobility services

has sparked much interest in the research community. The *Internet Indirection Infrastructure* (*i*3) is an overlay framework supporting very general routing services. While it does not require any particular structure in its implementation, a DHT is a natural and desirable candidate.

Routing in *i*3 is entirely identified-based. Instead of an address, peers send messages to identifiers. Receivers interested in those messages place triggers in the overlay. Basic triggers are the ordered pair $< id, addr >$, where *id* is the identifier messages are sent to, and *addr* is the receiver's IP address. Triggers with the same *id* are stored in the same DHT node. Messages from the sender are routed to the node storing the triggers with the given *id*, who then forwards it to each registered address. Such indirection allows for natural multicast, anycast, host mobility and service composition.

- *Multicast:* A multicast group is identified by a group id *G*. Nodes join the group by inserting triggers $< G, addr >$ into the DHT. Senders simply have to send messages to the id *G*. A scalable multicast scheme which extends this basic idea is presented in [13].
- *Anycast:* Servers insert triggers of the form $< S, addr >$. Nodes then locate a server by sending a request to id *S*. The service id *S* is separated into a prefix and the suffix. The latter is used for load balancing or locality-awareness in selecting servers. For instance, location information may be enocded into the ID suffix and the suffix of the requested *ID* to achieve locality-awareness. On the other hand, servers could insert multiple triggers with random suffixes proportional to their capacity to achieve load-balancing.
- *Host mobility:* When a node moves, it inserts a trigger $< ID_{old}, addr_{new} >$, where ID_{old} is the ID of its previous address and $addr_{new}$ is its new address. One may consider them as the home address and foreign address respectively in IP mobility. Changes in the node's address is reflected by inserting new triggers with updated $addr_{new}$. The reader is referred to [30] for details.
- *Service composition:* In certain applications, messages may need to be processed in between the server and service requestor. *i*3 achieves this by making triggers more flexible. In its more powerful form, triggers allows the *addr* field to be an ordered list of identifiers (we may consider an address an identifier as well). This list behaves like a stack, with the top of the stack on the left end of the list. Suppose a packet from server *S* to requestor *R* needs to pass through a processing node *P*. Figure 5 shows how data is redirected to the processing node before reaching *R*. The advantage of this approach is that processing is receiver-driven. The server need only send one format of the application data, while different receivers may process the data individually for compatibility. Numerous examples of this redirection mechanism is given in [14].

From the above, we see the need to organize a large number of triggers and route messages to triggers efficiently. Thus, DHTs are a good choice for the overlay. DHTs can also provide robustness through trigger replication and reduce latency by locality-aware techniques. Given this scalable, efficient and robust overlay, the above routing services can be provisioned in large scale with desirable performance.

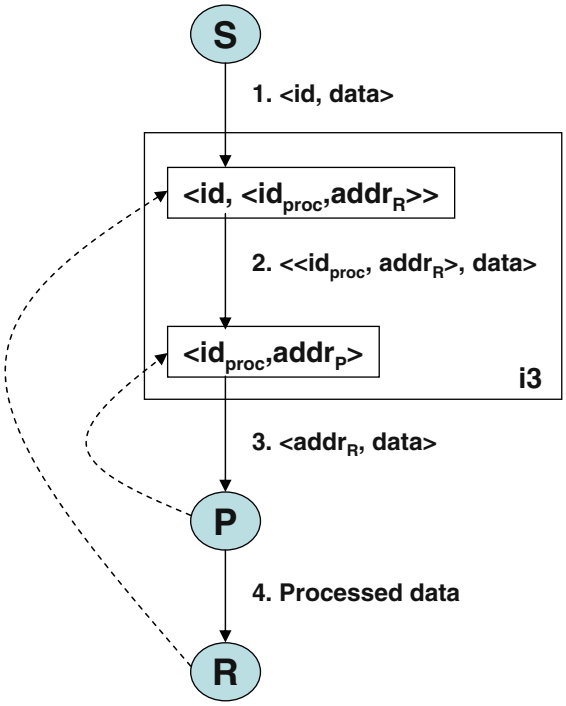


Fig. 5 Example of service composition in $i3$. The receiver R inserts the trigger $< id, < id_{proc}, addr_R >>$ and the processing node P inserts the trigger $< id_{proc}, addr_P >>$. The server S sends the data to id without knowing the data will be processed by P. The message is redirected to the trigger $< id_{proc}, addr_P >$ and then forwarded to P for processing. P obtains R's address from the message and finally sends the processed data to R

6 DHTs in Wireless Networks

Recent advances in mobile technology, such as mobile computing power and wireless communications capabilities, has sparked great interest in building scalable applications for large-scale wireless networks. In this section, we highlight the characteristics of wireless networks, how such characteristics hinder the deployment of DHTs and various approaches to adapt DHTs to such environments.

6.1 Characteristics of Wireless Networks

There are three major types of wireless networks:

- **Mobile ad-hoc networks (MANETs)** are characterized by high node mobility and lack of infrastructure support. Nodes of low processing power and move in random directions at random speeds, thereby making the topology unstable. An

example of MANETs is the vehicular ad-hoc network, where wireless devices in vehicles interact with each other while moving at high speeds.
- In **wireless sensor networks (WSNs)**, thousands of sensor nodes are scattered onto a large geographical area. The sensors are very low-profile devices with very limited processing power, memory and battery. The primary concern of a sensor network is its lifetime, therefore protocols for sensor nodes focus on power conservation. The network serves as a repository of sensed data, such as temperature and humidity. Queries are not directed towards a specific node. Instead, multiple nodes cooperate to reply with aggregated data that is sensible to the application.
- **Wireless mesh networks** provide an alternative way for network access in areas where infrastructure is difficult and/or costly to install. In its most general form, a wireless mesh network consists of a number of gateways, mesh points and end-hosts. The gatways provide access to the wired Internet. Mesh points are typically small devices with limited processing power and memory mounted on lamp posts or rooftops. They extend the network coverage of the gateway wirelessly. By associating with a default gateway, they forward Internet-bound traffic from associated users to the gateway. Routing is also conducted between different mesh points. While users are mobile and may switch their associated mesh points at any time, the gateway and mesh points are generally stationary.

While these wireless networks have different uses and greatly diversified node capabilities, they share some characteristics which are of paramount importance in considering the design of DHTs in such environments:

- *Limited shared bandwidth:* The (broadcast) wireless channel is shared by all nodes and has limited bandwidth. A DHT scheme has to be light-weight in terms of bandwidth consumption.
- *Lack of routing infrastructure:* Unlike the wired network, where routing is readily available at low cost, routing in wireless networks is non-trivial. The two problems associated with routing in this case are high maintenance overhead of routes and inefficient bandwidth utility due to long routes.
- *Low processing power and memory capacity:* Wireless devices are usually low in processing power and limited in memory capacity. A desirable DHT scheme should be light-weight in terms of node processing and storage.

In the following sections, we discuss two main issues with wired DHTs in wireless networks and outline solutions proposed in the literature.

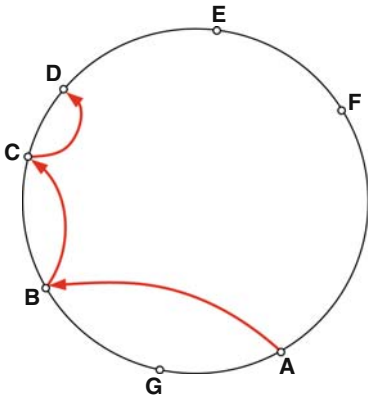
6.2 Challenges of Using DHTs in Wireless Networks

It is argued that DHTs designed for the wired network are not directly applicable to wireless networks. The two main issues are as follows.

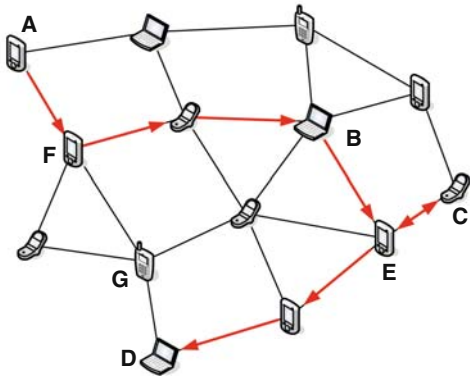
1. *Overlay mismatch problem:* In the DHT overlay, any pair of nodes is considered to be "one-hop" apart. While this abstraction is somewhat valid in the wired network where routing is efficient and low-cost, it is definitely not the case in

wireless networks. In particular, a single overlay hop may map to multiple physical links, resulting in inefficient bandwidth utility [19, 21, 28].

Figure 6 shows an example of the overlay mismatch problem. The overlay path from node *A* to node *D* is 3 hops, as perceived by the application. However, it actually spans 8 physical links, one of which is traversed twice. Notice that the shortest path via nodes *F* and *G* cannot be obtained by the overlay lookup algorithm since the algorithm is fixed with routing on identifiers only and with no consideration of physical proximity of nodes.



(a) Overlay path node *A* to node *D* as seen by the application.



(b) Actual path traversed in the physical network. This path is much longer than the perceived length of the overlay path and may traverse a link multiple times, e.g. the link between nodes *C* and *E*.

Fig. 6 Illustration of the overlay mismatch problem. The overlay path from node *A* to node *D* spans 8 links, whereas the shortest path is only 3 hops. It is impossible to tell by considering the overlay alone that the shortest path is $A \rightarrow F \rightarrow G \rightarrow D$

2. *High maintenance overhead:* DHT maintenance procedures ensure routing convergence and efficient routing (in terms of number of overlay hops). While the overhead incurred by such procedures is acceptable in the wired network, the same procedures are demanding for bandwidth-limited wireless networks. For example, in Chord, a node periodically run the *stabilize(·)* and *fix_fingers(·)* methods to ensure that its successor and finger table entries are up to date. Each of these operations may require a route discovery. For reactive routing protocols, this incurs up to $O(n)$ overhead, where n is the number of nodes [10, 18]. Proactive routing requires periodic flooding of topology control, which is particularly costly in wireless sensor networks where power is of utmost importance [6]. It is also difficult to achieve convergence in MANETs as the topology changes quickly. Multiple route discoveries may be needed in the presence of mobility. Furthermore, overlay mismatch exacerbates this issue, since much bandwidth is spent obtaining routes that are unnecessarily long. The situation could be even worse than simple flooding in resolving requests for data items [19, 28].

6.3 Search Approaches for Wireless Networks

The overlay mismatch problem and high maintenance overhead incurred by traditional DHTs must be resolved in order to make DHTs feasible for wireless networks. In this section, we discuss optimizations and alternate approaches that aim to resolve these two issues.

- *Adding location awareness:* The overlay mismatch problem is caused by blindly layering the DHT layer directly on top of the routing protocol. An intuitive idea is to let nodes choose overlay neighbors that are physically close. This is achieved by cross-layering the two overlay routing and underlay routing. Several methods have been proposed.

  In Ekta, Pastry nodes collect DSR routes by overhearing control packets [19]. A node only maintains routes to physically close overlay neighbors as indicated by hop counts in DSR routes. The correctness of overlay routing is maintained because in prefix-based routing that Pastry employs, choosing any one of the possible nodes for a given bit position would ensure route convergence. The key idea here then is to choose the node that is closest in the underlay.

  MADPastry employs clustering to enforce physical proximity of overlay neighbors [28]. The identifier space is divided into m (equal) partitions, where m is the number of clusters. Each cluster's identifier space starts with a different prefix. A landmark heads a cluster and floods beacons throughout the cluster so that new nodes may join the appropriate cluster. A node identifiers is composed of its cluster's prefix and a suffix obtained via hashing its address or public key. The Pastry routing table is stripped down to contain only m entries, one for each cluster. When resolving a query, the first overlay hop is taken to the cluster housing the target key. After that, leaf-set routing is used to reach the reference node. As

long as clusters are relatively small, the search path is confined within a small geographical region, thus solving the overlay mismatch problem.

If nodes know about their geographical locations, e.g. by means of GPS, it is possible to structure their identifiers to match the underlay topology. A common technique is to apply a specialized hash function to map node positions (in 2D) to a 1D identifier space. For instance, the hash function employed by Georoy and MeshChord maps node locations to identifiers on a unit ring in a way that nodes in the same geographical region will also be in the nearby each other (in the same segment) in the unit ring [3, 8]. An illustrative example is given in Eq. (3.1) and Fig. 3 of [8]. The concept of cell-addressing in Cell Hash Routing is of similar spirit to Georoy and MeshChord [2].

Another technique is to replace traditional DHT overlay routing with geographical routing. The Geographic Hash Table (GHT) and CHR use Greedy Perimeter Stateless Routing (GPSR) to route queries to reference nodes [21]. GPSR greedily forwards queries to the target location in the physical network [11]. There is no need for overlay routing, thereby solving the mismatch problem.

- *Reducing maintenance overhead:* Various approaches have been proposed to curb maintenance overhead. One method is to reduce the amount of overlay routing information. This may involve reducing the number of overlay neighbors and/or lowering the frequency of refreshing overlay connections. Ekta tries to reduce route discoveries by overhearing DSR routes to overlay neighbors. MADPastry achieves low overhead by storing a degenerate Pastry routing table with only as many entries as there are clusters. Virtual Ring Routing (VRR) takes this approach one step further by only storing routes to successors and predecessors in a Chord-like identifier ring [4].

  Another method is to eliminate the need for overlay routing. For example, GHT and CHR employ geographical routing (GPSR), which requires local exchange of node locations only. No flooding for route discovery (as in reactive routing) or of topology information (as in proactive routing) is needed.

7 Conclusions

We introduced in this chapter the distributed hash table (DHT), a scalable, efficient, self-organizing and robust peer-to-peer routing infrastructure. Data is inserted into the DHT in the form of key-value pairs, where the key is an identifier uniquely distinguishing the data. DHT nodes store these key-value pairs and conduct efficient data lookup using a fully distributed algorithm. Scalable routing is realized by limiting lookups to $O(\log N)$ hops, where N is the number of nodes in the DHT. Important extensions such as locality-aware neighbor selection and key replication reduces response time, improves robustness against node dynamics and provides load balancing.

Several applications based on DHTs are presented, such as distributed storage systems (CFS), application-layer multicast (Scribe), peer-to-peer video-on-demand

with rich user interactivity (VMesh) and a framework for general routing services (*i*3). These applications exploit the benefits provided by the underlying DHT for large-scale operation in the Internet.

DHT deployment in wireless networks is also discussed. The characteristics of wireless networks, such as bandwidth scarcity and limited node processing capacity, make it difficult to apply DHT schemes directly. Two main issues, namely the overlay mismatch problem and excessive maintenance overhead, hinder DHT deployment in such networks. Various approaches have been highlighted to address theses issues.

The role of DHTs in future networks is an open issue. While we have focused on DHT design on wired networks and wireless networks, the future of networking is most probably a hybrid of both. With millions of mobile devices participating in networked applications across the Internet, it is interesting to see how DHTs can be employed to achieve the required scalability and communications latency, while keeping up with high degrees of node dynamics.

References

1. Internet Domain Survey. www.isc.org/ds
2. Araujo, F., Rodrigues, L., Kaiser, J., Liu, C., Mitidieri, C.: CHR: A distributed hash table for wireless ad hoc networks. In: Proc. of IEEE Distributed Computing Systems Workshops (ICDCSW), pp. 407–413 (2005)
3. Burresi, S., Canali, C., Renda, M.E., Santi, P.: MeshChord: A location-aware, cross-layer specialization of Chord for wireless mesh networks (concise contribution). Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on pp. 206–212 (2008)
4. Caesar, M., Castro, M., Nightingale, E.B., O'Shea, G., Rowstron, A.: Virtual ring routing: Network routing inspired by DHTs. SIGCOMM Computer Communication Review **36**(4), 351–362 (2006)
5. Castro, M., Druschel, P., Kermarrec, A.M., Rowstron, A.: Scribe: a large-scale and decentralized application-level multicast infrastructure. Selected Areas in Communications, IEEE Journal **20**(8), 1489–1499 (2002)
6. Clausen, T., Jacquet, P.: Optimized link state routing protocol. In: IETF RFC 3626 (2003)
7. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles, pp. 202–215. ACM, New York, NY, USA (2001)
8. Galluccio, L., Morabito, G., Palazzo, S., Pellegrini, M., Renda, M.E., Santi, P.: Georoy: A location-aware enhancement to Viceroy peer-to-peer algorithm. Computer Network **51**(8), 1998–2014 (2007)
9. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of dht routing geometry on resilience and proximity. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 381–394. ACM, New York, NY, USA (2003)
10. Johnson, D., Hu, Y., Maltz, D.: The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4. In: IETF RFC 4728 (2007)
11. Karp, B., Kung, H.T.: GPSR: greedy perimeter stateless routing for wireless networks. In: MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking, pp. 243–254. ACM, New York, NY, USA (2000)

12. Khashmir: http://khashmir.sourceforge.net
13. Lakshminarayanan, K., Rao, A., Stoica, I., Shenker, S.: Flexible and robust large scale multicast using i3. Tech. Rep. CS-02, University of California, Berkeley (2002)
14. Lakshminarayanan, K., Stoica, I., Wehrle, K.: Support for service composition in i3. In: MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia, pp. 108–111. ACM, New York, NY, USA (2004)
15. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A scalable and dynamic emulation of the butterfly. In: Proceedings of the 21st annual ACM symposium on Principles of distributed computing (2002)
16. Maymounkov, P., Mazires, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS), pp. 53–65 (2002)
17. Mehyar, M., Spanos, D., Pongsajapan, J., Low, S.H., Murray, R.M.: Asynchronous distributed averaging on communication networks. IEEE/ACM Transactions on Networking **15**(3), 512–520 (2007)
18. Perkins, C., Royer, E.: Ad-hoc on-demand distance vector routing. Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on pp. 90–100 (1999)
19. Pucha, H., Das, S.M., Hu, Y.: Ekta: An efficient DHT substrate for distributed applications in mobile ad hoc networks. In: Proc. of IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), pp. 163–173. IEEE (2004)
20. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 161–172. ACM, New York, NY, USA (2001)
21. Ratnasamy, S., Karp, B., Shenker, S., Estrin, D., Govindan, R., Yin, L., Yu, F.: Data-centric storage in sensornets with GHT, a geographic hash table. Mobile Networks and Applications **8**(4), 427–442 (2003)
22. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, pp. 329–350. Springer-Verlag (2001)
23. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149–160. ACM, New York, NY, USA (2001)
24. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.C.: Chord: A scalable peer-to-peer lookup service for internet applications. Tech. Rep. TR819, Laboratory for Compuer Science, Massachuseets Institute of Technology (2001)
25. Vuze: http://wiki.vuze.com/index.php/distributed_hash_table
26. Xu, J.: On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE **3**, 2177–2187 (2003)
27. Yiu, W.P., Jin, X., Chan, S.H.: VMesh: Distributed segment storage for peer-to-peer interactive video streaming. Selected Areas in Communications, IEEE Journal **25**(9), 1717–1731 (2007)
28. Zahn, T., Schiller, J.: MADPastry: A DHT substrate for practicably sized MANETs. In: Proc. of IEEE Workshop on Applications and Services in Wireless Networks (ASWN) (2005)
29. Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., Kubiatowicz, J.: Tapestry: a resilient global-scale overlay for service deployment. Selected Areas in Communications, IEEE Journal **22**(1), 41–53 (2004)
30. Zhuang, S., Lai, K., Stoica, I., Katz, R., Shenker, S.: Host mobility using an internet indirection infrastructure. Wireless Networks **11**(6), 741–756 (2005)