MINEMA

Survey on Distributed Hash Tables[1]

Filipe Araújo and Luís Rodrigues
Univ. de Coimbra, Univ. de Lisboa
filipius@dei.uc.pt, ler@di.fc.ul.pt

Abstract

In this paper we review distributed hash tables (DHTs) that work as overlay networks on top of the IP network. We compare these DHTs under a number of key aspects that concern performance, scalability and self-configurability. Additionally, since these networks are not suitable for wireless *ad hoc* networks, we shortly review techniques that can be used to take distributed hash tables to wireless environments.

Document Identifier	TR-05
Document Status	Final
Created	13 October 2005
Revised	24 November 2006
Distribution	Public

[1] This paper is based on a chapter of the PhD Thesis of Filipe Araújo [1].

Contents

1 Overview 1

2 DHTs in Overlay Networks 1

2.1 The Routing Scheme of a Generic DHT 2

2.2 Chord 2

2.3 Content-Addressable Network (CAN) 4

2.4 Expressways Content-Addressable Network 5

2.5 Pastry 6

2.6 Tapestry 7

2.7 Viceroy 7

2.8 D2B 9

2.9 Koorde 11

2.10 TOPLUS 12

2.11 Comparison of the DHTs 13

3 Position-Based DHTs for Wireless *Ad Hoc* Networks 17

4 Summary 17

1 Overview

A dictionary stores values which can be accessed by associated keys. A hash table is a dictionary in which keys are mapped to array positions by a hash function. Informally, a hash table stores (key, value) pairs[2]. A user of the dictionary must use a key to store or to retrieve the corresponding value. A centralized solution, where a single node holds the entire dictionary, is prone to a number of problems, like lack of tolerance to faults, high congestion and low scalability. The natural answer to these problems lies in the decentralization of data provided by a distributed hash table (DHT). In a DHT, nodes must cooperate to maintain the coherence of the data. Nodes use a consistent hash function [2] to determine the peer that holds a given value (often, some sort of file, a music, or a pointer to the location of those items). Usually, nodes self-organize to form a communication graph which has optimal or near-optimal path length/node degree trade-off (in wired networks, both are typically logarithmic with respect to the number of nodes). In this way, although nodes have a small number of neighbors (logarithmic or better) the DHT needs only a small number of hops (logarithmic or better) to satisfy requests from clients. When compared to a centralized solution this is a fair price to pay, because distribution offers some considerable advantages, like improved fault-tolerance. In fact, a carefully designed DHT may resist to single or even multiple node failures, loosing only the data that was in the departing nodes. Another advantage of a distributed solution is the reduced congestion. While a centralized server needs to reply to all requests, most DHTs need to reply only to a small fraction of the requests, like $O(\log n/n)$, where n is the number of nodes. Furthermore, DHTs scale better than a centralized solution, both in terms of communication and storage requirements.

2 DHTs in Overlay Networks

An overlay network is a network operated on top of another underlying network, but organized under an independent logic. For this reason overlay networks are also deemed as “logical” or “virtual” networks. The growing interest of users in peer-to-peer applications, like Napster[3] and Gnutella[4] ignited the creation of many peer-to-peer overlay networks that implement DHTs, like Pastry [3], Tapestry [4], Chord [5] and CAN [6]. Although different in their details, most of these DHTs share a number of common features. According to [7], in a DHT, one can often distinguish between the “routing scheme”[5] and the “routing geometry” of the overlay network. The routing scheme is the set of mechanisms that determines the neighbors and the next hops (this basically corresponds to the formal definition of *routing scheme*). Often, the routing scheme compels the nodes to create an underlying graph with a predetermined organization, which is called the “routing geometry”. The notion of *routing geometry* is quite useful, because it allows us to pinpoint some of the most relevant differences among existing DHTs. Depending on the particular geometry, nodes may have some flexibility to choose their neighbors and to choose paths for the messages, given their actual neighbors. As we shall see, some routing geometries give their nodes degrees of freedom in both aspects (neighbors and paths), others only in one, while the remaining give none. Flexibility to choose neighbors allows the DHT to select closer peers, thus improving routing latency, while flexibility to choose paths also impacts latency (less) and increases tolerance to faults caused by nodes that have departed. To shorten presentation of the DHTs, we will use a “generic DHT” with no specific routing geometry. We use this generic model to present the components of the routing scheme that are common to most DHTs. Then, we briefly overview and compare some of the most well-known DHTs that exist.

[2] See for example the web page of National Institute of Standards and Technology — Dictionary of Algorithms and Data Structures: http://www.nist.gov/dads.

[3] See http://www.napster.com.

[4] See http://www.gnutella.com.

[5] In [7], this is called the “routing algorithm”.

2.1 The Routing Scheme of a Generic DHT

A DHT stores (key, value) pairs, e.g., ("Bob", 32), where "Bob" is the key, while 32 is the value we want to store, e.g., Bob's age. The DHT must have a globally known hash function capable of converting the key to a pseudo-random value, e.g., an integer or a position in a virtual space. One of the crucial aspects is that the hash function should balance the distribution of keys throughout the space. We first consider that the DHT creates an underlying graph $\mathcal{H}$, which is a function of the set of existing nodes. Nodes of the DHT receive a (pseudo) random identifier that evenly spreads the nodes in the space of identifiers. The space of identifiers of the nodes must coincide with the output space of the hash function (to ensure this, some DHTs, e.g., Bamboo [8], hash the identifier of the node plus port of the application). If nodes and keys are evenly distributed in space, each node will store a similar share of the keys.

The routing scheme must create graph $\mathcal{H}$ in a way that a path between two arbitrary nodes always exists (at least in steady state conditions). To retrieve (or to store) the value corresponding to a key, the DHT must find the node that stores that key. Take again the example above and assume that the key "Bob" hashed to 81. In general, node 81 will not exist (because nodes are sparse in the identifier space) and some other node will become responsible for that key. Assume that it is node 90. Any node, say node 13 looking for the age of Bob will use the key "Bob", which hashes to 81. In general, nodes only have a partial view of the network and node 13 will not know whether node 81 exists or not. Nevertheless, the DHT will try to route the message to node 81 and will always end up in node 90. One of the problems in most DHTs is that their overlay network bears only a limited (if any) relation with the underlying network. Consequently, each hop in the DHT may represent a very long distance in the Internet and successive hops may make the message travel back and forth several times.

Entrance and departure of nodes from the DHT is often very similar from one DHT to the next. To illustrate the process, we will take as example Pastry [3]:

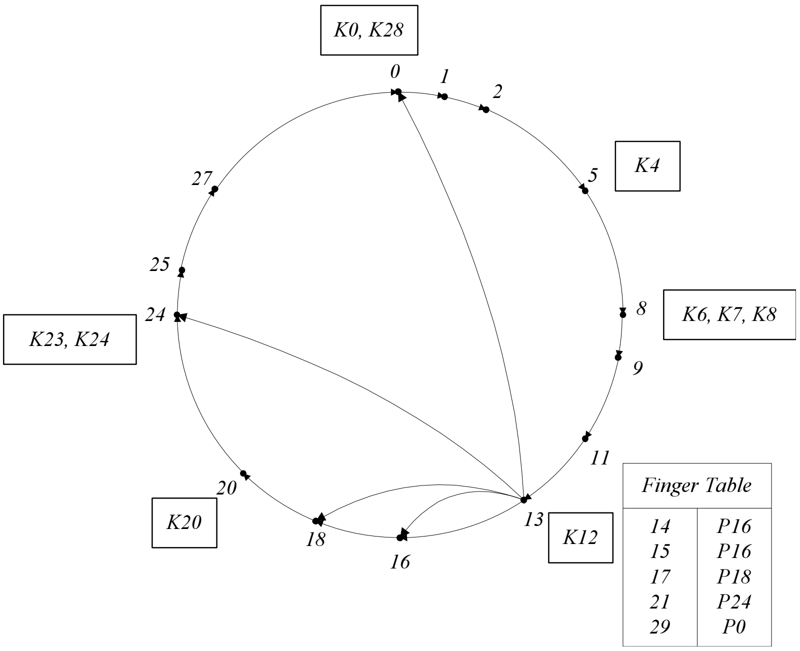
- to enter the network, a node with identifier N must ask some node P_0 already in the network to send a special JOIN message to node N (which hopefully does not exist in the network). This JOIN message will be routed to the node responsible for the identifier N in the network, say P_f. At this moment, P_f will know that there is a new neighbor coming in and it divides its own space and its keys with the newcomer. The remaining actions to take strongly depend on the concrete DHT;
- ideally, departure of nodes involves redistribution of the keys of the leaving node. However, in practice, nodes may leave abruptly, rendering this option impossible.

Next, we review some of the most important DHTs.

2.2 Chord

Overview

In Chord [5], nodes organize into a logical ring ordered by increasing order of identifier. To close the ring, the smallest node follows the largest one. To maintain the ring, each node keeps a pointer to the node that follows it. The ring allows to define the notion of *successor node of a key.* For some key k, successor(k), is the node of the ring with smallest identifier, not smaller than k or, if all nodes have identifiers smaller than k, successor(k) is the node with smallest identifier (0 is successor of $2^m - 1$). To improve routing performance, Chord nodes use "finger tables" with m entries, where 2^m is the number of possible identifiers. This scheme requires $O(\log n)$ memory at



Finger Table	
14	P16
15	P16
17	P18
21	P24
29	P0

Figure 1: Chord ring

each node, but ensures delivery of messages in $O(\log n)$ hops with high probability [6]. The i-th entry of the finger table at node P keeps a pointer to the first node S that succeeds P by at least 2^{i-1}, i.e., $S = \text{successor}(P + 2^{i-1} \textit{ modulo } 2^m)$. These pointers correspond to long range contacts (LRCs) carefully selected along the ring. If Chord reaches a steady state, where nodes have updated finger tables, each hop successively eliminates half of the possible remaining identifiers.

Figure 1 depicts an example of a Chord ring with $m = 5$. Nodes hold the keys represented as Kx. We also show the finger table of node 13 (the rows of the first column result from the computation $13 + 2^{i-1}$ *modulo* 32, while the second column is the successor of this identifier). As an example consider that destination node is $D = 27$ and forwarding node is $P = 13$. In this case, the first hop of the message will go through node successor(21) = 24.

In [5], Stoica *et al.* claim that simplicity, provable correctness and provable performance are the main features of Chord. This is further explored in [9].

Routing Geometry

The routing geometry of Chord is a ring. This ring is augmented with LRCs that nodes store in their finger tables. When populating their routing table, Chord nodes have many options. Except for their immediate neighbors, Chord nodes can select their LRCs in a rather flexible way (however the original specification of Chord does not allow this flexibility). The more distant the LRC is, the more options there are. Routing also has a large flexibility, because there are many possible routes with $O(\log n)$ path lengths. Ideally, there are $O((\log n)!)$ different ways of arranging the paths with length $O(\log n)$. To see this, consider an average path with a sequence of hops like $n/4, n/8, n/16, \ldots, 4, 2, 1$. We can rearrange this sequence in an arbitrary way and still get the same length. In fact, for k terms, we can have $k!$ different orderings.

[6] With high probability, this means that an event will occur with a probability of at least $1 - O(1/n)$.

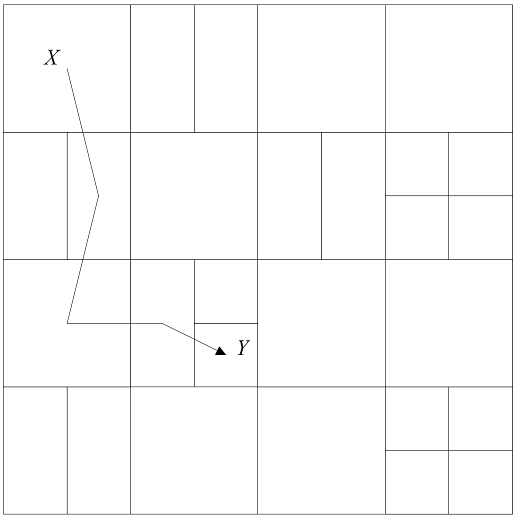


Figure 2: Content-Addressable Network (CAN)

2.3 Content-Addressable Network (CAN)

Overview

Content-Addressable Network (CAN) divides a virtual (imaginary) d-dimensional torus into d-dimensional zones. There is a one-to-one correspondence between CAN nodes and the d-dimensional zones. The hash function deterministically maps the keys to coordinates of the virtual torus and each process manages all the keys that hash inside its own zone. A visual representation of CAN for anything above two dimensions is not very intuitive and therefore, Figure 2 represents a CAN square, for only two dimensions. In fact, this is not a square, but a torus, because coordinates wrap and the virtual space has no borders. Also, note that physical coordinates bear no relation to virtual space.

To route messages, CAN uses a greedy algorithm, where each node sends the message to the neighbor that is closest to destination. In CAN, nodes are only aware of neighbors with which they share a common border of the space. Figure 2 depicts an example that illustrates the routing process between X and Y.

In [6], Ratnasamy *et al.* claim that average path length in a d-dimensional CAN with n nodes is $(d/4)n^{1/d}$, i.e., $O(dn^{1/d})$, and that individual nodes maintain only $2d$ neighbors. However, we notice that it suffices to look at Figure 2 to see that the number of neighbors may be higher than $2d$. Additional methods are required to avoid very unfavorable space partitions where some node might need to store information of an arbitrary number of neighbors. Authors also state that CAN can achieve $O(\log n)$ hops by setting $d = (\log_2 n)/2$. However, for typical configurations of CAN, where the number of nodes in the system is not known beforehand, the path length/node degree trade-off of CAN is very unfavorable, as paths tend to be very long. CAN can significantly benefit from some enhancements to its basic design [6].

Routing Geometry

The routing geometry of CAN becomes a hyper-cube instead of a torus if we consider that $d = \log n$ and that all possible identifiers are taken by existing nodes. In this case, each node will have precisely $\log n$ neighbors each one differing in a single bit. Therefore, routing will take $\log n$ hops,

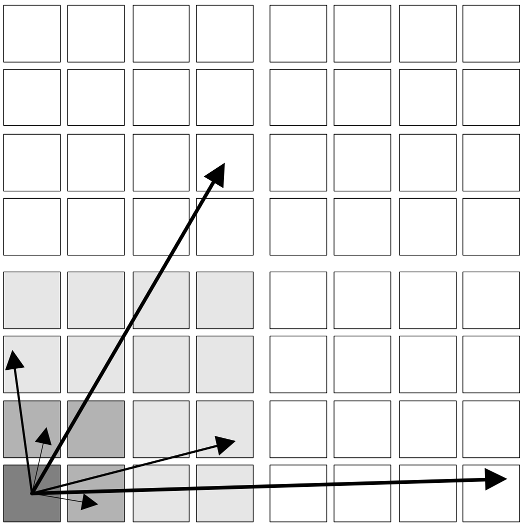


Figure 3: eCAN-like LRCs

as each one of the hops will correct one of the bits. However, a practical implementation of CAN can hardly correspond to this idealized hyper-cube. Not only the number of nodes of the system is unknown when it boots, but it is likely to vary. Consequently, either the dimensionality is too large or too small. If it is too large, the number of neighbors per node will be above $O(\log n)$. If it is too small, most of the nodes will not lie in the vertices of the hyper-cube and crossing each dimension needs more than a single hop, more than a logarithmic number of hops and in fact $O(n^{1/d})$.

From the two flexibility criteria, CAN only owns one, which is the routing flexibility. Since there is only one way of arranging the hyper-cube, there is no flexibility at all to select the neighbors. On the other hand, nodes can route messages using several of their neighbors. If the forwarding node and the destination have b different bits, the forwarding node can select any of the b neighbors that are closer to destination. This means that there are $O((\log n)!)$ possible paths from source to destination.

2.4 Expressways Content-Addressable Network (eCAN)

Overview

The CAN DHT is not very efficient in practice, because path lengths tend to be very long. To overcome this problem Xu and Zhang [10] proposed an extension to CAN, called “expressways CAN” (eCAN). Refer to Figure 3. To simplify presentation we will consider a two-dimensional space, but this mechanism works for an arbitrary number of dimensions. The idea in eCAN is to make a first level division of the entire space into four big squares[7]. Each node keeps LRCs to the two neighboring squares. Then, the four big squares are further divided into other four smaller squares. This time, nodes inside squares have a total number of four LRC (above, below, right and left). This process is repeated for as many levels as wanted. Figure 3 illustrates the eCAN-like LRC scheme, for a node in a corner. Wrapping pointers are not shown.

[7] Division in 3 × 3, 4 × 4 or any other number of squares is also possible.

Routing Geometry

If the routing geometry of CAN were truly a hyper-cube with $O(\log n)$ dimensions, eCAN would be pointless. However, in practice, CAN works like a torus and eCAN reduces path lengths in that torus. The LRC mechanism of eCAN gives a lot of flexibility to nodes in what concerns selection of neighbors. Each LRC has to fall within a hyper-cube with possibly many nodes. However, eCAN is not flexible in route selection, because, in general, there is little gain in progressing until the message crosses the border of its current largest hyper-cube. This means that a node has only one good option to reach destination. Although eCAN has some similarities with the finger table of Chord, the division of the space in hyper-cubes makes it less worthwhile to do small steps forward.

2.5 Pastry

Overview

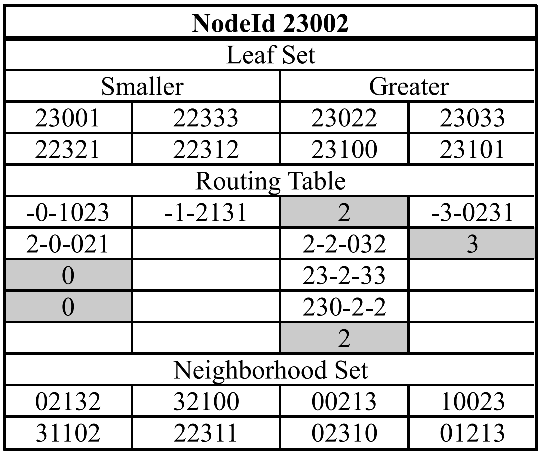
Pastry and Tapestry are two descendants from the work of Plaxton *et al.* [11]. Pastry routes messages as follows. Consider that some node $S = 14543_{10}$ sends a message to node $D = 84944_{10}$. To succeed, S must know some node that starts with an 8. Consider it to be $R_1 = 83135$. Now, R_1 must know about some node that starts with an 8 and has 4 in the second position, say $R_2 = 84899$. This reasoning goes on for nodes $R_3 = 84996$, $R_4 = 84945$ and finally D. Implementations of Pastry use a numeration base of 2^b for some b. b is a parameter of the system, typically set to 4, which means that the numeration base is 16. Under accurate routing tables and in the absence of recent node failures, $O(\log n)$ hops suffice for a lookup operation, w.h.p., while the number of entries in the routing information of each node is $O(\log n)$ [3]. n is the number of nodes in the system.

Each node divides its routing information in three parts. The first part is the "routing table", which includes information of peers needed to route messages according to the description made before. The "leaf set", L of node P includes a set of nodes with identifiers close to P. Nodes use this set to know exactly which keys belong to them and which keys belong to their neighbors. Nodes also use the leaf set to route to nodes with close identifiers. Additionally, nodes store a list of nodes that are physically close called "neighborhood set". The purpose of this list is to improve locality properties of routing. By forwarding messages to nodes that are topologically closer, routing becomes more efficient. Figure 4 [3] shows the information stored in some example Pastry node, for $b = 2$ and $|L| = 8$ in a system that uses 5 digits for the node identifier. Node identifiers are split in three parts: equal prefix, current digit and different suffix. First row keeps addresses of nodes that have no common prefix with current node. Second row keeps addresses of nodes that share the first digit with the current node and so on. At each row, the cell whose digit matches the node's digit has a gray background. The routing table of each node has an average of $\lceil \log_{2^b} n \rceil$ rows [3].

Rowstron and Druschel [3] claim that Pastry has good locality properties, in the sense that more often than not, nodes will select to neighbors nodes that are close to them. This property is ensured by the way that Pastry nodes build the network.

Routing Geometry

The routing geometry of Pastry is hybrid [7]. The routing tables of the nodes are organized as a tree, while the leaf set of the nodes forms a ring (with some redundancy). The ring part of the graph is easy to understand and, therefore, we shall focus on the tree. Each node is the root of its own tree. Any node that shares all but the last digit of that identifier is a candidate to be a child node. Exactly which nodes exist on that level of the tree depends on things like the nodes that the "root" node used to join the network. The grandchildren nodes share all but the two last



NodeId 23002			
Leaf Set			
Smaller		Greater	
23001	22333	23022	23033
22321	22312	23100	23101
Routing Table			
-0-1023	-1-2131	2	-3-0231
2-0-021		2-2-032	3
0		23-2-33	
0		230-2-2	
		2	
Neighborhood Set			
02132	32100	00213	10023
31102	22311	02310	01213

Figure 4: State of node 23002

digits of the root's identifier and so on, until only the first digit coincides in the lowest level of the tree. Figure 5 depicts such a tree for node 23002. Routing for node 23002 is straightforward. The path length between two nodes is determined by the subtree that connects them. Transient network states can cause the tree to be incomplete and force the ring to come into play.

Selection of neighbors in Pastry is quite flexible, especially for entries of the routing table that correspond to nodes which only share a few bits in their identifier. For instance, in Figure 5, node 21330 could choose many different nodes starting by $23xxx$, while node 23021 only has a few choices for nodes starting by $2300x$ (possibly only one). On the other hand, there is no routing flexibility in a tree, because there is only one neighbor that yields good routing results.

2.6 Tapestry

Routing Geometry

Pastry and Tapestry's main mechanisms are very similar and therefore, to conserve space, we omit the details of Tapestry. The main difference between the routing geometry of these DHTs is that the routing geometry of Tapestry is a pure tree, because Tapestry does not have any ring. Figure 5 also represents this structure, despite the fact that Tapestry resolves digits in the opposite order. Similarly to Pastry, Tapestry is very flexible in the construction of the routing table, while the same is not true for the route selection.

2.7 Viceroy

Overview

Viceroy [12] implements a DHT with constant out-degree and logarithmic diameter. Additionally, insertion or removal of a node requires a number of link changes that is constant in expectation and logarithmic with high probability. The Viceroy network is based on the butterfly network.

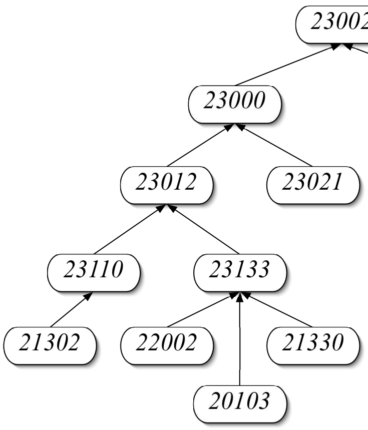


Figure 5: Pastry tree rooted at node 23002

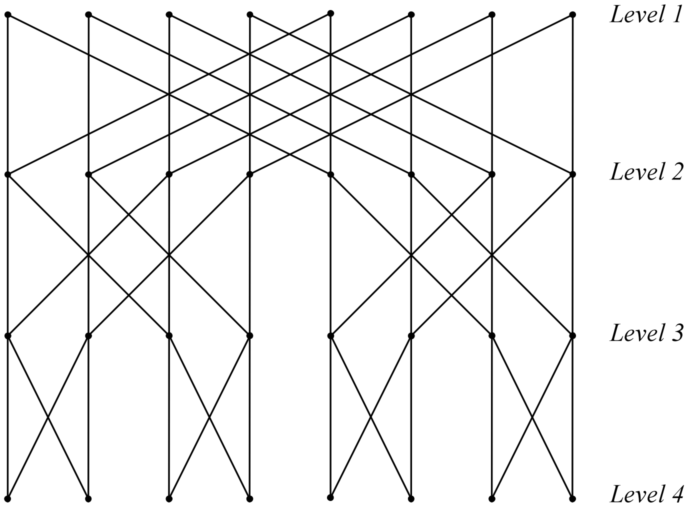


Figure 6: The butterfly network

Figure 6 illustrates a butterfly network comprised of thirty two nodes distributed by four different levels. Note that, despite having only a constant number of connections, nodes can reach each other in a logarithmic number of steps.

The principle of the Viceroy network is to enhance the Chord basic ring with carefully selected LRCs inspired in the butterfly configuration. Viceroy nodes try to divide themselves into $\log n$ different levels, n being the number of nodes. A Viceroy network is comprised of three different types of structures: i) a ring like Chord; ii) the butterfly, that connects nodes from the general ring using an emulation of the butterfly network; and iii) level rings that connect nodes from the same butterfly level in a ring structure. To organize the Viceroy network, nodes randomly select identifiers in the range $[0, 1)$, to determine their positions along the 2π radians of the general Chord ring. A node has the following constant number of connections: two connections with predecessor and successor nodes in the general ring; right and left connections to the next level of the butterfly, a connection to the upper level of the butterfly; and two connections to the predecessor and successor nodes in the level ring. The left and right connections of the butterfly to the next level intend to divide ring space, according to the following rule: for the left connection of node N of level l is selected the neighbor in level $l + 1$ that is closest to N in the clockwise direction (i.e., the "clockwise-closest" neighbor) around the general ring; for the right connection is selected the neighbor of level $l+1$ clockwise-closest to $N+1/2^l$. Connections from level l to level $l+1$ are expected to be about $1/2^l$ far apart in the ring. The reader may confront this with the perfect butterfly graph depicted in Figure 6, considering that the distance from the leftmost to the rightmost lines is 1. For the up-link, the clockwise-closest neighbor of level $l-1$ is chosen. All the butterfly connections are used for routing purposes.

The level of a node in the butterfly is randomly selected from the set $\{1, \ldots, \lfloor \log n \rfloor\}$, where n is the number of nodes. Being impossible to determine the exact number of nodes, node N estimates this to be $n_0 = 1/d(N, \text{successor}(N))$, where $d(N, \text{successor}(N))$ is the distance from N to N's successor in the general ring. For instance, if $N = 0.44$ and $\text{successor}(N) = 0.64$, N will assume that network has 5 nodes and will select its level between 1 and 2. If successor of N changes, N must re-select its level.

Routing is performed in three sequential steps, called "proceed to root", "traverse tree" and "traverse ring". In the first step, nodes send the message to the first level of the butterfly through their up-links. In the second step, nodes traverse the tree from the first level to the destination. At node N, level l, distance to destination D, $d(N,D)$, is at most $1/2^{l-1}$. Hence, if $d(N,D) < 1/2^l$, left link is chosen, otherwise, right link is chosen. This process will either reach some node without further down links or a node beyond destination D. Assume that $X \neq D$ is the node reached in the end of the second step. The last step will make use of the level ring to achieve destination in $O(\log n)$ steps w.h.p. [8]. In the last step of the routing algorithm, if $X < D$, nodes select either the successor in the level ring, Y, if $Y \leq D$, or the successor in the general ring if $Y > D$. If $X > D$, the reasoning is similar. For further details on Viceroy see [12].

Routing Geometry

The routing geometry of Viceroy is a variation of the butterfly graph represented in Figure 6. The Viceroy DHT offers no flexibility in the selection of neighbors or in the selection of paths.

2.8 D2B

Overview

One of the most interesting features of D2B [13, 14] is that nodes have constant node degree and, still, path lengths are logarithmic. D2B network is based on the *de Bruijn graph.* A de Bruijn graph, $B(2,k)$, for $k \geq 1$, has 2^k nodes with k-bit identifiers. In-degree and out-degree of nodes is 2 and diameter of the graph is k. Node with identifier $x_1x_2\ldots x_k$ has an arc directed to nodes $x_2\ldots x_k\alpha$, for $\alpha \in \{0,1\}$ and has incoming arcs from $\beta x_1 \ldots x_{k-1}$, for $\beta \in \{0,1\}$. Figure 7 depicts an example of $B(2,3)$. Routing from $x_1x_2\ldots x_k$ to $y_1y_2\ldots y_k$ is done by selecting intermediate nodes $x_2\ldots x_k y_1$, $x_3 \ldots x_k y_1 y_2$, etc., until destination is reached. For instance, routing from 101 to 000 is done through 010 and 100.

D2B must incorporate some additional features to *i*) support the DHT functionality, *ii*) support sparsity of nodes and *iii*) cope with the dynamic behavior of a network made of nodes that vary in number. The key space of D2B is $\mathcal{K} = \{0, \ldots, 2^m - 1\}$, for a binary string length of m. The key $\mathcal{k}_1\mathcal{k}_2\ldots\mathcal{k}_m$ is managed by existing node $x_1x_2\ldots x_k$ if and only if $x_1x_2\ldots x_k$ is a prefix of $\mathcal{k}_1\mathcal{k}_2\ldots\mathcal{k}_m$. This raises the notion of "universal prefix set", S, which is a set where for any possible integer with infinite length w there is one and only one prefix of w in S. For example, $\{0, 100, 1010, 1011, 11\}$ is a universal prefix set. The empty set is also a universal prefix set. D2B must ensure that at any

[8] In [12], Malkhi *et al.* present two versions of the routing algorithm. The simplest version uses the general Chord ring instead of the level rings, but may require $O(\log^2 n)$ steps (more precisely, the expected number of steps is $O(\log n)$ and it is $O(\log^2 n)$ w.h.p.).

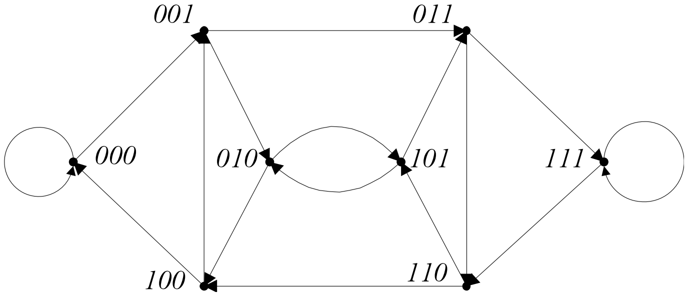


Figure 7: Example of a de Bruijn graph, $B(2, 3)$

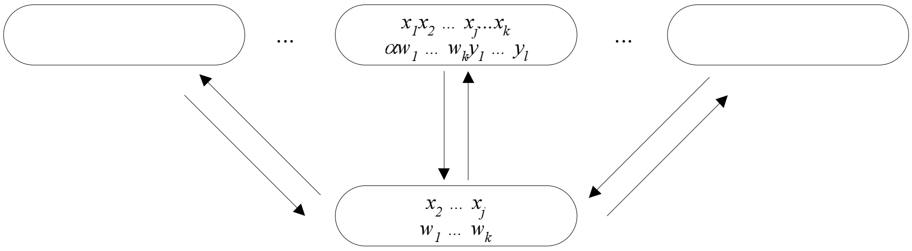

...
$x_1x_2 \dots x_j \dots x_k$
$\alpha w_1 \dots w_k y_1 \dots y_l$
...
$x_2 \dots x_j$
$w_1 \dots w_k$

Figure 8: Multiple parent may correspond to a single child

given time, for any identifier, there is one and only one node with a corresponding prefix in the network. In other words, identifiers of nodes of D2B must form a universal prefix set.

The sparsity of the identifier space implies that, unlike the static de Bruijn graph, node $x_1x_2 \dots x_k$ may have either a single child $x_2 \dots x_j$, $j \leq k$ or several children with identities $x_2 \dots x_k y_1 \dots y_l$, $1 \leq l \leq m - k + 1$. If several children exist, the set of sequences $y_1 \dots y_l$ forms a universal prefix set. For instance, node 110 may have more than one children with the following identifiers, 100, 10100, 101010, 101011, 1011, or may, otherwise, have a single child with identifier of 1 or 10. Given the childhood relation, the parenthood relation is symmetrical. If node A has a single child B with identifier $w_1w_2 \dots w_k$, then, B may have several parents with identifiers $\alpha w_1 \dots w_k y_1 \dots y_l$, where $\alpha \in \{0, 1\}$ and $y_1 \dots y_l$ forms a universal prefix set. On the other hand, if node A has several children including B, with identifier $w_1w_2 \dots w_k$, then B's single parent is A and the identifier of A is $\beta w_1 \dots w_j$, where $\beta \in \{0, 1\}$. Figures 8 and 9 illustrate childhood and parenthood relations. To keep coherency with the text we used two different identifiers for nodes in the figures.

Routing Geometry

The routing geometry of D2B is a variation of the de Bruijn graph represented in Figure 7. D2B is an inflexible architecture, both from the point of view of the neighbor selection and from the point of view of the route selection. This is quite evident for neighbor selection, due to the highly structured de Bruijn graph. Additionally, there is also only one optimal option to route the message and therefore this does also not meet the routing flexibility criterion.

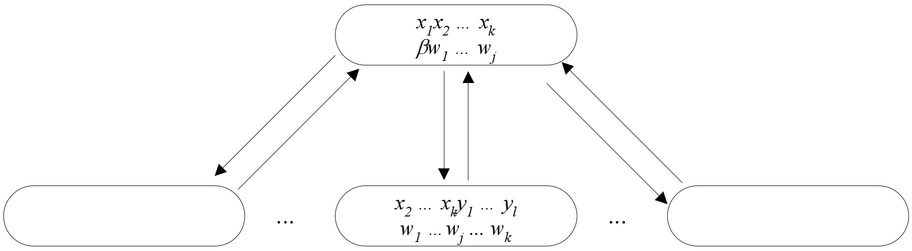


Figure 9: One parent may have several children

Algorithm 1 Lookup algorithm at node N

```
Function $lookup(k, kshift, I)$
  if $k \in (N, \text{successor}(N))$ then
    return successor(N)
  else
    if $I \in [N, \text{successor}(N))$ then
      return call function $D.lookup(k, kshift << 1, I \circ topBit(kshift))$
    else
      return call function $\text{successor}(N).lookup(k, kshift, I)$
    end if
  end if
```

2.9 Koorde

Overview

Koorde [15] is a DHT that augments the basic Chord ring with a de Bruijn graph. Koorde network achieves the following optimal results: *i)* for $O(1)$ node degree, Koorde diameter is $O(\log n)$ w.h.p.; *ii)* for $O(\log n)$ node degree (which enhances fault tolerance), Koorde diameter is $O\left((\log n)/\log\log n\right)$. Koorde node m requires information about 2 nodes: its successor in the ring and the predecessor of $2m$ *modulo* 2, known as D (unlike a pure de Bruijn graph, there is no pointer to $(2m+1)$ *modulo* 2, as this would probably be redundant).

The Koorde DHT uses the notions of "virtual node" I and "real node" N. I corresponds to the current routing node in the de Bruijn graph. Since it may happen for identifier I not to exist in the Chord ring, node N is a real node that represents I in the ring, such that $I \in [N, \text{successor}(N))$. To route toward key k, node X invokes the function *lookup*() in Algorithm 1 with arguments $X.lookup(k, k, X)$. We used the original notation of [15]: $kshift$ is used to extract successive bits, starting from the most significant, from key k; function $topBit(x)$ returns the most significant bit of x; finally, $x \circ b$, left-shifts x and introduces the bit b at the right of x. The algorithm first checks if key belongs to the successor. If it does not, but node N represents I, request is forwarded to node D and operator $\circ$ is used on I. Otherwise, request for key is forwarded to the successor in order to search for node I. In Figure 10, we depict the Koorde network corresponding to Figure 1. The pointer D at each node is shown inside a box. Figure 11 exemplifies the *lookup*() function of a request for key $k = 21 = 10101_2$, starting at node $N = 18 = 10010_2$.

The number of hops can be reduced if instead of using N as initial I, the most significant bits of the key k are inserted from the beginning in I, such that I does not overflow the range of N that stops at (not including) successor(N). For instance, in the previous example, we could immediately do $I = 18 = 10010_2$ and $kshift = 101$. Although there is no improvement in this particular case, Kaashoek and Karger [15] prove that this enables Koorde to achieve $O(\log n)$ path lengths with high probability.

Koorde has a second variant where it can use a base-$O(\log n)$ (say base-k) de Bruijn graph. In this

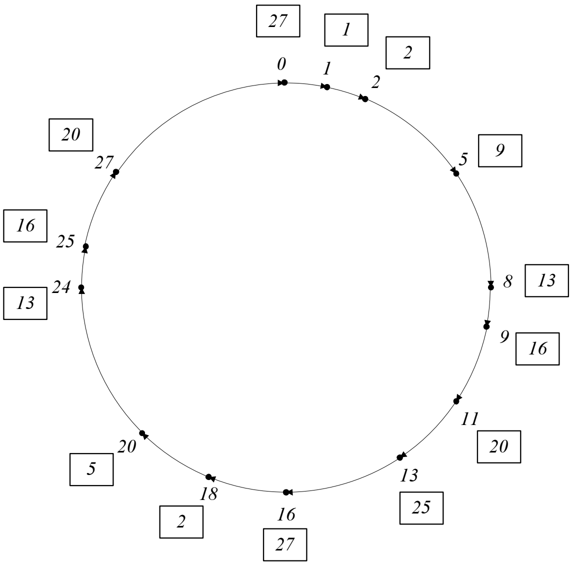


Figure 10: A Koorde network

$kshift$	10101	0101	0101	101	01	01	01
I_2	10011_2	00101_2	00101_2	01010_2	10101_2	10101_2	10101_2
I_{10}	19	5	5	10	21	21	21
N	18	2	5	9	16	18	20

Figure 11: Request for key $21 = 10101_2$ starting at node $18 = 10010_2$

variant, node m uses pointers to k predecessors of km. This represents a node degree of $O(\log n)$, but, on the other hand, it improves the path lengths to $O\left((\log n)/\log\log n\right)$, which is still optimal.

Routing Geometry

The routing geometry of Koorde comprises two parts: a basic Chord ring (without the finger tables) plus a variation of the de Bruijn graph represented in Figure 7. Neither of the variants of Koorde has any flexibility concerning either the selection of paths or the selection of neighbors. However, comparison might not be entirely fair. In the second variant of Koorde, with $O(\log n)$ neighbors, the effect of choosing alternative neighbors or alternative routes is not very clear. Although this would affect the optimality of Koorde, fact is that most remaining DHTs are already suboptimal.

2.10 TOPLUS

Overview

In most peer-to-peer systems there is a mismatch between the logical and the physical networks. As a consequence, paths selected by routing schemes in peer-to-peer networks may be considerably longer than the corresponding paths in the physical network. The TOPLUS DHT [16] addresses this problem. TOPLUS organizes peers in groups according to their IP addresses. Groups are then organized into a new higher-order group, which is then organized into a new even higher-order group, and so on until, at the highest level, there is only one group, which includes all possible nodes. To increase correlation between group and network structure, TOPLUS gets topological information from Border Gateway Protocol (BGP) tables.

Let $H_N(X)$ be the lowest-order group including node X; let $H_{N-1}(X)$ be the second lowest-order group including $H_N(X)$, until $H_0(X)$, which includes all groups and all nodes. Node X must know all the nodes of group $H_N(X)$; at level $N-1$, node X must know at least one node from each group that is sibling of $H_N(X)$ (i.e., some node Y such that $H_{N-1}(X) = H_{N-1}(Y) \wedge H_N(X) \neq H_N(Y)$); the same applies for all other levels until level 0. The collection of all IP addresses known to X is the routing table of X.

To route, nodes use a metric derived from a simple longest-prefix matching known as the "XOR metric", which is also used in Kademlia [17]. A distance between two identifiers j and k under the XOR metric is defined as $d(j,k) = \sum_{v=0}^{31} |j_v - k_v| \cdot 2^v$, where j_v is the v-th bit of j. The difference of the XOR metric to the longest-prefix match is that the former always breaks ties when prefixes have the same length (see [16] for details). Given the routing table structure of node X, it is trivial to prove that routing will always converge. In fact, node X always knows some node that is closer to destination D under the XOR metric (e.g., some node F belonging to some sibling group $H_1(F)$).

While capable of achieving a stretch as low as 1.17 compared to the IP routing, TOPLUS has a number of drawbacks [16]. The most obvious one is that consistent hashing is no longer able to balance load among the nodes, because nodes may not be evenly spread in the identifier space. As a consequence, some nodes may receive an unfair share of the load. Another practice that might be used in other peer-to-peer systems would be to assign several virtual nodes to the same powerful peer. This is also difficult in TOPLUS. Another issue is the possibility of correlated node failures that may bring down an entire set of related IP addresses, thus reducing the effectiveness of using peers with neighboring addresses to increase availability.

Routing Geometry

The routing geometry of TOPLUS is hybrid in the sense that TOPLUS creates a tree of groups of nodes. Nodes have pointers to groups at higher levels of the tree as well as pointers to sibling groups at each level. TOPLUS only requires nodes to have complete knowledge of the lowest level group of the node. Selection of neighbors in TOPLUS is very flexible. The same does not happen with route selection. The tree structure of TOPLUS allows only a single option for the optimal route.

2.11 Comparison of the DHTs

To compare the DHTs, we evaluate their most significant features:

performance and scalability as the network size increases up to thousands or millions of nodes, scalability depends heavily on the behavior of the following factors: ability of nodes to share the load, growth of path lengths versus degree of nodes and congestion at the nodes. While most DHTs assume that load sharing is ensured by hashing (and possibly randomization), figures for path lengths and node congestion may vary. To evaluate performance and scalability, we will use the theoretical bounds as they are presented by their authors;

self-configurability to evaluate the self-configurability of the DHTs, we focus on the ability of the network to choose better paths, on the tolerance to faults and on churn[9]. To evaluate the ability of the network to choose better paths, we outline conclusions of [7]. Resistance to churn can be seen as a particular case of tolerance to faults. However, it is interesting to

[9] The site http://searchcrm.techtarget.com defines the "churn rate" as the number of costumers who discontinue a service during a specified time period divided by the average total number of costumers over the same time period.

Table 1: Comparison between *expected* performance of several peer-to-peer systems

P2P system	Node degree	Network diameter	Node congestion
small-worlds	$O(1)$	$O(\log^2 n)$	$O\left((\log^2 n)/n\right)$
Chord	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
CAN	$O(d)$	$O(dn^{1/d})$	$O(dn^{1/d-1})$
Pastry	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
Tapestry	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
D2B	$O(1)$	$O(\log n)$	$O((\log n)/n)$
Viceroy	$O(1)$	$O(\log n)$	$O((\log n)/n)$
Koorde cfg. 1	$O(1)$	$O(\log n)$	$O((\log n)/n)$
Koorde cfg. 2	$O(\log n)$	$O((\log n)/\log\log n)$	$O((\log n)/(n\log\log n))$

isolate both concepts and analyze them under different perspectives. We base our evaluation on tolerance to faults on the interesting work of Gummadi *et al.* [7] that defines the concept of "static resilience" of a DHT. Although tolerance to faults depends on many different mechanisms, static resilience measures the ability of the DHT to resist to node failures prior to any attempt of reconstruction. Static resilience strongly depends on the particular DHT and on the routing geometry. We evaluate the static resilience of the different DHTs in Section 2.11. Other recovery mechanisms may be applicable to more than one DHT and consequently they are less prone to comparisons. On the other hand, churn has an interest of its own, because the patterns of utilization of existing peer-to-peer systems tend to show that participation in the network is highly dynamic. Therefore, DHTs need to deal with a high churn rate. However, there is little data available to compare the DHTs and additionally, many techniques are applicable to all the DHTs. For this reason, we will focus on generic measures to improve resistance to churn and, as a consequence, self-configurability. To infer behavior under churn and to mitigate this problem, we outline conclusions of [18, 19, 20].

Performance and Scalability

Path lengths are measured in terms of hops traveled by a message in the overlay network, while the degree of a node is the number of its overlay neighbors. These parameters are tightly connected and cannot be simultaneously reduced, as any n-node network meets its fundamental limits in the inequality $d^{h+1}/(d-1) \geq n-1$. $d^{h+1}/(d-1)$ is an upper bound for the number of nodes within h hops for a given maximum node degree d. In fact, for $O(1)$ node degree, expected path lengths can be, at best, $O(\log n)$, while for $O(\log n)$ node degree, expected path lengths cannot be shorter than $O(\log n/\log\log n)$ [15]. The path length/node degree trade-off is one of the central criterion used to evaluate a DHT.

Besides path lengths and node degree, it is also common to find a theoretical derivation of congestion at nodes in the papers where authors present their DHTs. Table 1 summarizes these figures for the DHTs presented before, when available (see also [13, 12]). For comparative purposes, we have also included networks with small-world characteristics. A small-world has a constant node degree and poly-logarithmic diameter. In [21, 22, 23], we can find some notable works that study the creation of small-worlds. One important aspect that we must be aware of is that there may be no correspondence between these theoretical bounds and the results observed in practice, because the $O()$ notation hides a constant factor that may vary widely. For instance, experimental results of Gummadi *et al.* [7] tend to demonstrate that, for instance, the butterfly network is very inefficient when compared to networks with $O(\log n)$ node degree.

Table 2: Flexibilities of the different DHTs architectures

Property	Optimal paths	Neighbor selection
Ring (Chord)	$O(\log n)$	$n^{\log n/2}$
Hyper-cube (CAN)	$O(\log n)$	1
Torus (eCAN)	1	$n^{\log n/2}$
Hybrid (Pastry)	1	$n^{\log n/2}$
Tree (Tapestry, TOPLUS)	1	$n^{\log n/2}$
de Bruijn (D2B, Koorde)	1	1
Butterfly (Viceroy)	1	1

Self-Configurability

Path Latency As we referred before, there is often a mismatch between the theoretical bounds for the trade-off between path lengths and node degrees and the observed performance. This difference may result from two facts. First, the constant hidden in the $O()$ notation may vary widely. Second, this trade-off does not convey any information about latency. In a DHT where nodes pick their identifiers in a random way, there are two main techniques to reduce latency [7]: *i*) Proximity Neighbor Selection (PNS): pick nearby neighbors when creating the routing tables (if possible) or *ii*) Proximity Route Selection (PRS): when more than a single path is available to a destination, try to select paths through neighbors with a small latency. In general, PNS consistently yields better results than PRS. According to [7], this is a consequence of the fact that PNS has more alternatives than PRS. Many DHTs allow a large number of candidates for a single routing entry[10], while, at routing time, the candidates for the next hop are restricted to the few options available in the routing table of the node. Interestingly, a combination of the two methods is also possible and yields good results.

Now, the question that we want to answer is which DHTs allow PNS and which do not. This is obviously related to the "neighbor selection" column of Table 2. Whenever more than one choice for the routing table is available a node may try to pick a neighbor with a small latency. Therefore, Chord, eCAN, Pastry, Tapestry and TOPLUS can select among multiple different neighbors, while the remaining cannot. One of the conclusions of [7] is that the routing geometry (i.e., tree, ring, etc.) does not seem to have any influence in the gains of PNS. The only thing that matters is whether or not it is possible to implement it. Concerning PRS, the DHTs that allow multiple paths, e.g., Chord or CAN, are also more prone to support "runtime" selection of neighbors.

Tolerance to Faults With respect to tolerance to faults, we make an analysis of the static resilience of a DHT, which depends on the routing geometry. If a given routing geometry offers more alternative paths to a destination, one may expect that a DHT based on this geometry will resist to a larger number of broken links. Note that this is largely independent of two mechanisms that can be included by virtually all DHTs: *i*) data replication to prevent data loss and *ii*) active recovery mechanisms to repopulate the routing tables. As noted in [7], if DHT A has better static resilience than DHT B, then DHT B needs to use quicker, more expensive, active recovery schemes, to offer a comparable resilience.

Table 2 summarizes the flexibility considerations that we made in the previous sections [7]. Experimental results of [7] tend to confirm the hypothesis that more paths provide better resilience. This result is consistent with the "optimal paths" column of Table 2. In particular, authors have observed that the tree and the butterfly have a low resilience, while the ring and the hyper-cube

[10] However, it is not possible to measure the latency to each one of the nodes and therefore, nodes must pick a neighbor from a small sampling subset.

have the highest resilience of the analyzed DHTs[11].

Resistance to Churn Churn is one of the most crucial problems of peer-to-peer systems. Informally, churn refers to the rapid membership changes that may occur. This was first observed in the context of peer-to-peer applications like Napster, Gnutella or FastTrack[12]. However, given the lack of widespread large-scale applications based on DHT [8], there is little or no consistent and thorough analysis of churn in DHTs. For this reason, we restrict ourselves to outline the conclusions of previous work [18, 19, 20].

The basic problem with churn is that there is a communication cost associated with the maintenance of the DHT. On the other hand, it has been observed that nodes can have very small up times (also known as "session times") in a DHT [24, 25, 26, 27, 28]. Some of these figures point to session times as low as a single minute for 50% of the nodes. This may occur if nodes that join a DHT supporting a file sharing system leave immediately after they find or give up finding a file the user was trying to retrieve. Hence, the basic trade-off is: either spending bandwidth updating outdated routing table entries or accepting a large latency resulting from not updating those entries. Li *et al.* [20] experimentally compared four DHTs (Chord, Pastry, Kademlia [17] and Kelips [29]) to find the feasibility limits of this trade-off between "average live bandwidth" and "average lookup latency". They concluded that given the right parameter settings this trade-off is identical for all the analyzed DHTs. Qualitatively, this trade-off curve looks like the function $y = 1/x+k$, where x is the live bandwidth, y is the lookup latency and k is the minimum achievable latency. This means that it is useless to spend more bandwidth beyond a certain point as this will not reduce latency below k. On the other hand latency will sharply increase if the system tries to save bandwidth below a critical point of the function.

Although interesting from a theoretical perspective, the basic problem is to make the system converge to the intended pace of substitution of stalled routing table entries. Rhea *et al.* [18] clearly show that some systems start failing most lookup requests when the churn rate increases. Additionally, they show that this results from an inadequate policy of replacing stalled entries. These systems replace entries in a *reactive* way, i.e., whenever a node detects that a given neighbor failed it tries to replace that neighbor. This may create a positive feedback cycle where the node congests its own access link trying to refresh its routing table and consecutively declaring more and more neighbors as dead as it cannot communicate with them. Hence the authors identify the three most important factors that may improve the response of a DHT to churn:

- periodic instead of reactive recovery from failures;
- calculation of message timeouts during lookups;
- choice of nearby over distant neighbors.

The first of these ideas is to recover at previously scheduled moments, instead of immediately reacting to failures. On the other hand, correctly estimating the timeouts is also a central aspect of resistance to churn. If a timeout is too short, a new request might be issued before the first one is able to reach the requester, thus congesting the network even more. On the other hand, an unnecessarily longer timeout will have a negative impact on latency. Finally, the third issue is nothing more than PNS. As we have seen, choosing nearby, instead of distant neighbors, is only possible with certain routing geometries. Although Rhea *et al.* [18] use the Bamboo [8] DHT[13] to support their claims, the use of mechanisms based on these conclusions may also increase the resistance of most other DHTs to churn.

[11] Gummadi *et al.* [7] have analyzed the following geometries: tree (Tapestry), hyper-cube (CAN), ring (Chord), butterfly (Viceroy), xor (Kademlia [17]) and hybrid (Pastry).

[12] See http://en.wikipedia.org/wiki/Fasttrack.

[13] The routing scheme and the routing geometry of Bamboo are identical to Pastry. Most of the differences lie in the maintenance mechanisms.

3 Position-Based DHTs for Wireless *Ad Hoc* Networks

All the DHTs presented before assume that there is a network with operational routing underneath. Then, most of these DHTs create a logical overlay network where nodes perform routing in a way that is strongly decoupled from the data network. This creates an overhead for every lookup operation of the DHT in a setting where resources are scarce. Additionally, routing can hardly be taken for granted in wireless *ad hoc* networks, which makes the problem even harder. As a result, if routing *per se* is such a complicated task, there is little sense in coming up with overlay networks that need an independent routing facility. Therefore, the problem of creating a DHT for wireless environments has deserved fewer efforts (there are, nevertheless, some examples, e.g., [30, 31, 32]).

Given these problems, we can use position to eliminate the mismatch between the data network and the logical network. Hence, like in the routing problem [33], construction of a DHT can strongly benefit from the use of positional information, whenever there is a strong relation between position and topology. While the most obvious application of this idea is to wireless *ad hoc* networks, wired networks can also benefit from this [34].

The idea of integrating routing with the DHT was first proposed in the Geographical Hash Table (GHT) of Ratnasamy *et al.* [30](see also [35]). Despite not being localized (see [36] for a definition of localized routing scheme), in most circumstances GHT will only need information of nearby nodes to operate correctly. In GHT there is a space of identifiers for nodes and keys. Unlike other DHTs, this space is not virtual, but physical. GHT relies on the Greedy Perimeter Stateless Routing (GPSR) protocol [37, 38] with little modifications to operate. GHT uses two additional related concepts: the "home node" and the "home perimeter". The *home node* of a point is the node which is geographically closest to that point in space. The *home perimeter* of a point is the set of edges that encloses that point[14]. Assume that the network supporting the DHT is routing to some destination D. In general, D will not correspond to any node. However, standard behavior of GPSR ensures that a packet always reaches the home node, thus making the operation of the DHT possible. Details on the operation and maintenance of the DHT can be found in [30].

One of the shortcomings of this DHT is that path lengths tend to increase as network density increases (since the underlying graph is planar, edges become shorter when node density increases). This problem is inherent to GPSR and we can use clustering to solve it. Perhaps the simplest way of clustering nodes in a position-based algorithm is to divide the space into cells of predictable size. This clustering can then be used to create a DHT. As a result, the cell becomes the addressing unit of the DHT and the nodes inside each cell must cooperate to manage the keys. Gupta *et al.* [39] suggest this form of clustering. Li *et al.* [40], take this idea a step further. They create a DHT on top of a logical structure of cells, which is part of a more general peer-to-peer information sharing architecture. A similar idea exists in [41], which presents a Two-tier Data Dissemination (TTDD). TTDD creates a grid of dissemination for each type of data. This grid also follows a geometrical arrangement. Then, TTDD uses flooding within the local area, while global dissemination is achieved through the grid, for the sake of efficiency. In [42], Araújo *et al.* present "Cell Hash Routing" (CHR), which is a DHT for wireless *ad hoc* networks, based in clusters of cells. CHR builds its clusters in a very simple way and due to the geometry of the cluster, routes packets directly using the logical clusters.

4 Summary

In this paper we surveyed some of the most well-known DHTs. These DHTs are implemented as peer-to-peer overlay networks working on top of IP. To make a detailed comparison, we evaluated the following aspects: performance and scalability (ability to share the load, path length vs. node

[14] The reader should keep in mind that GPSR needs a planar graph.

degree trade-off and node congestion) and self-configurability (ability to improve path latency, tolerance to faults and resistance to churn). Most if not all these DHTs are not suitable to wireless *ad hoc* networks for a number of reasons, including the mismatch between the overlay network and the topology. In this context, position-based DHTs have the advantage of not using a logical overlay network, thus being much lighter and, therefore, applicable to wireless *ad hoc* networks.

References

[1] F. Araújo, "Positin-based distributed hash tables," Ph.D. dissertation, University of Lisbon, Lisbon, Portugal, 2006.

[2] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* New York, NY, USA: ACM Press, 1997, pp. 654–663.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–350, 2001.

[4] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," UC Berkeley, Tech. Rep. UCB/CSD-01-1141, April 2001. [Online]. Available: citeseer.nj.nec.com/zhao01tapestry.html

[5] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *ACM SIGCOMM*, San Diego, August 2001.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications.* ACM Press, 2001, pp. 161–172. [Online]. Available: citeseer.nj.nec.com/article/ratnasamy01scalable.html

[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 *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications.* New York, NY, USA: ACM Press, 2003, pp. 381–394.

[8] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," University of California at Berkeley, Tech. Rep., December 2003.

[9] N. Lynch, D. Malkhi, and D. Ratajczak, "Atomic data access in content addressable networks: A position paper," in *1st. International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[10] Z. Xu and Z. Zhang, "Building low-maintenance expressways for p2p systems," HP, Tech. Rep. HPL-2002-41, 2002.

[11] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *ACM Symposium on Parallel Algorithms and Architectures*, 1997, pp. 311–320. [Online]. Available: citeseer.nj.nec.com/plaxton97accessing.html

[12] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Twenty-First ACM Symposium on Principles of Distributed Computing (PODC 2002)*, Monterey, California, July 2002.

[13] P. Fraigniaud and P. Gauron, "The content-addressable network D2B," LRI, Univ. Paris-Sud, France, Tech. Rep. 1349, January 2003.

[14] P. Fraigniaud and P. Gauron, "An overview of the content-addressable network D2B," Brief Announcement at 22nd ACM Symp. on Principles of Distributed Computing (PODC), July 2003.

[15] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table." in *IPTPS*, ser. Lecture Notes in Computer Science, M. F. Kaashoek and I. Stoica, Eds., vol. 2735. Springer, 2003, pp. 98–107.

[16] L. Garcés-Erice, K. Ross, E. Biersack, P. Felber, and G. Urvoy-Keller, "Topology-centric look-up service," in *COST264/ACM Fifth International Workshop on Networked Group Communications (NGC)*, Munich, Germany, 2003.

[17] P. Maymounkov and D. Maziéres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002, http://www.cs.rice.edu/Conferences/IPTPS02/.

[18] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in *USENIX 2004 Annual Technical Conference*, June 2004, pp. 127–140, http://www.usenix.org/events/usenix04/.

[19] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, 2002, pp. 233–242.

[20] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek, "Comparing the performance of distributed hash tables under churn," in *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, San Diego, CA, February 2004.

[21] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," in *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000. [Online]. Available: www.cs.cornell.edu/home/kleinber/swn.ps

[22] P. Duchon, N. Hanusse, E. Lebhar, and N. Schabanel, "Could any graph be turned into a small-world?" *Special issue of the international journal Theoretical Computer Science on Complex Networks*, 2005.

[23] L. Barrièrere, P. Fraigniaud, E. Kranakis, and D. Krizanc, "Efficient routing in networks with long range contacts (extended abstract)," in *15th Internation Conference on Distributed Computing*, ser. Lecture Notes in Computer Science, J. Welch, Ed., no. LNCS 2180. Lisbon, Portugal: Springer, October 2001.

[24] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, January 2002.

[25] B. N. L. J. Chu, K. Labonte, "Availability and locality measurements of peer-to-peer file systems," in *Scalability and Traffic Control in IP Networks II*, ser. Proceedings of SPIE, vol. 4868, July 2002.

[26] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment.* New York, NY, USA: ACM Press, 2002, pp. 137–150.

[27] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding availability." in *IPTPS*, ser. Lecture Notes in Computer Science, M. F. Kaashoek and I. Stoica, Eds., vol. 2735. Springer, 2003, pp. 256–267.

[28] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles.* New York, NY, USA: ACM Press, 2003, pp. 314–329.

[29] I. Gupta, K. P. Birman, P. Linga, A. J. Demers, and R. van Renesse, "Kelips: Building an efficient and stable p2p dht through increased memory and background overhead." in *IPTPS*, ser. Lecture Notes in Computer Science, M. F. Kaashoek and I. Stoica, Eds., vol. 2735. Springer, 2003, pp. 160–169.

[30] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A geographic hash table for data-centric storage in sensornets," in *First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, Georgia, September 2002.

[31] J. Eriksson, M. Faloutsos, and S. Krishnamurthy, "Scalable ad hoc routing: The case for dynamic addressing," in *IEEE Infocom 2004*, February 2004.

[32] H. Pucha, S. M. Das, and Y. C. Hu, "How to implement DHT in mobile ad hoc networks?" Student poster, the 10th ACM International Conference on Mobile Computing and Network (MobiCom 2004), September-October 2004.

[33] I. Stojmenovic, "Position-based routing in ad hoc networks," *IEEE Communications Magazine*, July 2002.

[34] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 173–185, 2001.

[35] P. Bose and P. Morin, "Online routing in triangulations," in *10th Annual Internation Symposium on Algorithms and Computation (ISAAC)*, 1999.

[36] E. Kranakis, H. Singh, and J. Urrutia, "Compass routing on geometric networks," in *11th Canadian Conference on Computation Geometry (CCCG 99)*, 1999.

[37] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in *ad hoc* wireless networks," in *International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, 1999, pp. 48–55.

[38] B. Karp and H. T. Kung, "GPRS: Greedy perimeter stateless routing for wireless networks," in *ACM/IEEE International Conference on Mobile Computing and Networking*, 2000.

[39] I. Gupta, R. van Renesse, and K. P. Birman, "Scalable fault-tolerant aggregation in large process groups," in *DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS).* IEEE Computer Society, 2001, pp. 433–442.

[40] M. Li, W.-C. Lee, and A. Sivasubramaniam, "Efficient peer-to-peer information sharing over mobile ad hoc networks," in *Second Workshop on Emerging Applications for Wireless and Mobile Access (MobEA II), in conjunction with the World Wide Web Conference (WWW)*, May 2004.

[41] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *Proceedings of ACM MOBICOM*, 2002.

[42] F. Araújo, L. Rodrigues, J. Kaiser, C. Liu, and C. Mitidieri, “Chr: a distributed hash table for wireless ad hoc networks,” in *The 25th IEEE International Conference on Distributed Computing Systems Workshops (DEBS ’02)*, Columbus, Ohio, USA, June 2005.

[43] M. F. Kaashoek and I. Stoica, Eds., *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2735. Springer, 2003.