

# On de Bruijn routing in distributed hash tables: There and back again\*

## EPFL Technical Report: IC/2004/41

Anwitaman Datta, Sarunas Girdzijauskas, Karl Aberer  
School of Computer and Communication Sciences  
Swiss Federal Institute of Technology Lausanne (EPFL)  
CH-1015 Lausanne, Switzerland

Email: {anwitaman.datta, sarunas.girdzijauskas, karl.aberer}@epfl.ch

### Abstract

We show in this paper that de Bruijn networks, despite providing efficient search while using constant routing table size, as well as simplicity of the understanding and implementation of such networks, are unsuitable where key distribution will be uneven, a realistic scenario for most practical applications. In presence of arbitrarily skewed data distribution, it has only recently been shown that some traditional P2P overlay networks with non-constant (typically logarithmic) instead of constant routing table size can meet conflicting objectives of storage load balancing as well as search efficiency. So this paper, while showing that de Bruijn networks fail to meet these dual objectives, opens up a more general problem for the research community as to whether P2P systems with constant routing table can at all achieve the conflicting objectives of retaining search efficiency as well as storage load balancing, while preserving key ordering (which leads to uneven key distribution).

**Keywords:** Distributed Hash Tables, Routing, de Bruijn networks, Storage Load Balancing

## 1 Introduction

The growing popularity of peer-to-peer networks makes them a very likely candidate for being a substrate for future internet scale information systems. After the initial popularity of centralized Napster, and flooding based networks like Gnutella, several distributed hash tables (DHT),

also known as structured peer-to-peer networks or overlay networks, have emerged [1, 19, 21, 23], providing greater scalability, accuracy and efficiency.

DHTs are typically based on the PRR scheme [16] that had been proposed for efficiently accessing cached copies of distributed objects. The initial research related to DHTs resulted in networks, where if the search space is partitioned in  $N$  disjoint partitions, then each peer needed  $O(\log(N))$  references to route (forward) a query for any particular key to the appropriate partition using on an average  $O(\log(N))$  network messages (query forwarding). This includes Chord [23], CAN [19], Pastry [21], P-Grid [1]. Note that even though CAN has a constant routing table size, it achieves logarithmic search only with routing tables of logarithmic size. For the rest of the paper, we'll refer to these as the traditional DHTs. The next phase of research resulted in P2P networks, where constant degree networks with logarithmic search properties were proposed [12, 9]. These networks were typically emulation of the Butterfly network or the de Bruijn network. Because of the simplicity of realizing de Bruijn networks on top of many existing DHTs as substrate, demonstrated originally by Koorde [9] built on top of Chord, de Bruijn routing has gained a tremendous popularity in the DHT research community, and resulted in proposals for de Bruijn routing for several other traditional DHTs, for instance CAN-d2B [7] on top of CAN, or on P-Grid (as will be shown in this paper in Section 3).

While storage of routing information may not be critical, the constant size of the routing table ( $K$ ) is expected to marginalize the cost of route maintenance while retaining the efficiency of logarithmic (base  $K$ ) search cost.

In this paper, we take a more pragmatic look at the possibilities of using de Bruijn routing in DHTs. There are several aspects, including resilience [3] of P2P networks, that the initial researchers [9] have already pointed out to be the pos-

\*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935.

sible Achilles heel<sup>1</sup> for de Bruijn networks. We thus focus on another aspect of these networks, that has so far largely been ignored for even the traditional DHTs. Storage load-balancing at individual peers when key distribution is uneven has until recently been left unaddressed.

Typically, storage load balancing is taken care of by using a cryptographic hash function to generate the keys that are uniformly distributed over the key space, and hence trivially solving the issue of load-balancing. The solution is elegant and simple, and hence immensely popular. While use of cryptographic hashing makes sense for networks like Freenet [4] where censorship resistance and anonymity are primary objectives, for other applications it is not only unnecessary, but also limits the utility of the P2P network, since a crucial information of ordering of keys is lost while using cryptographic hashing, such that the resulting DHTs then can not support simple extensions of keyword search, for example range queries. Thus, as P2P networks become ever more omnipresent, and more applications use such P2P overlays as the substrate to develop next generation internet scale applications and information systems, we have to rethink the design of DHTs to meet the conflicting goals of preserving the key ordering, storage load-balancing and search efficiency.

It has recently been shown that traditional distributed hash tables which have non constant routing tables (on an average logarithmic of number of partitions of the key space) can have storage load-balancing and logarithmic search cost even if key distribution and hence partitioning of search space is uneven. This has particularly been proven for P-Grid [2]. On the other hand, in this paper we show that constant out-degree networks using de Bruijn routing are unsuitable for achieving the dual goals of load-balancing and efficient search. This demonstrates the limitations despite the apparent advantages and recent interest around de Bruijn network based P2P systems. The arguments we put forward in this paper revolves around this central theme of load-balancing and search efficiency for arbitrary key distributions.

In Section 2 we first give a background of distributed hash tables, and introduce P-Grid, as well as a brief introduction to de Bruijn networks, and describe in Section 3 how they can be realized for efficient routing on top of the traditional DHT [1, 23, 19] abstractions. Next we provide a more elaborate description of the conflicts arising from storage load balancing in Section 4. In Section 5 we investigate the consequences of uneven partitioning of search spaces on de Bruijn routing, and arrive at the conjecture that while tra-

ditional DHTs can meet the conflicting goals, de Bruijn routing is not suitable for such requirements. We conclude in Section 6, highlighting that while conventional DHTs have reached a degree of maturity wherefrom they are well suited as the substrate for internet-scale information systems, constant routing table sized based systems have various open challenges which will comprise our and indeed the whole research communities future work. From the arguments and evidence provided in this paper, we show that de Bruijn networks in particular are unsuitable for many important and practical applications that are expected to use the P2P network as their underlying infrastructure, thus clearly demarcating the assumptions under which and why de Bruijn networks will (not) make sense.

## 2 Background

### 2.1 Distributed Hash Tables

Structured peer-to-peer networks based on distributed hash tables(DHT) provide scalable distributed data structures(SDDS) that can be used to efficiently locate resources in a decentralized manner. There are two important aspects in defining DHTs.

- **Association of resources (keys) to peers:** When any resource is searched in a P2P network, it essentially translates to search for the peer that holds the resource or an index for the resource. Consequently, a resource (corresponding key) needs to be associated with a peer, where the key is generated by using some hashing function on that attribute of the resource (typically the name), by which the resource is later searched.

A simple way to assign a key to a peer is to use the same range for specifying peer identifiers as the key range, and then associating keys based on closeness to peer identifier. This is for instance the case for Chord. While simple, the drawback with this approach is its implications on load-balancing if data distribution is skewed. Even with uniform key distribution, Manku [13] identifies the problem of *partition balance*, which is only aggravated in presence of skewed key distribution.

Ways to mitigate the effect may include generating peer identifiers that conform to the key distribution. While this will potentially solve the problem statically (assuming global knowledge), if key distribution is temporal in nature, then such a mapping mechanism will run into difficulties.

Another approach is to divide the key space dynamically in explicit partitions based on the key distribution,

---

<sup>1</sup>While a small value of  $K$  like 2 will make the network vulnerable to faults, a larger (but sub-logarithmic) constant  $K$  can provide a good trade-off between the maintenance cost as well as fault-tolerance. This issue is beyond the scope of the present paper.

and assign peers to be responsible for these partitions. By disentangling the peer identifiers from keys, and instead using a mechanism to discover the peers responsible for the zones instead of the peers themselves, more flexibility for storage load-balancing is achieved. This is the approach used in P-Grid [2] (also for CAN [19] in principle), where we solve the partition balance problem [13] for any arbitrary key distribution.

- **Efficient mechanisms to route search requests to responsible peers:** This determines the choice of routing table and mechanisms to forward and search (or insert) requests using a proper choice of routing entry from the routing table.

Often when the data structure for DHTs are defined, and constructed, one tends to couple both the routing tables and the association of data (key) to peers together. However we would like to emphasize that these are essentially orthogonal issues. For instance, the same partitioning of the search space can still use different routing tables and hence different routing mechanism.

We illustrate this later in the paper in Figure 1 where we show an example on how the same partitioning of the key space can be explored using P-Grid, CAN and de Bruijn routing.

There are other important aspects for DHTs, including maintenance of the routing tables and their resilience in presence of dynamics in the network [3], also known as churn [20, 11], and proximity and network latency related issues, however these are beyond the scope of this paper.

Next we briefly introduce P-Grid, which has been shown to achieve storage load-balancing while preserving search efficiency even if the key distribution is skewed, that is, even if the key-space is partitioned unevenly [2]. This is important because it shows the existence of traditional P2P systems which can meet the conflicting goals of storage load balancing and search efficiency, and hence its performance becomes a natural benchmark to compare the performance of any other DHT, including the new genre constant degree P2P networks.

In P-Grid, we make the design choice of partitioning the key space and assigning the zones to peers independent of their identifiers, thereby retaining greater flexibility for storage load-balancing.

## 2.2 The P-Grid data structure [1, 2]

Since we compare de Bruijn based networks with performance of P-Grid (as a representative traditional DHT with non-constant routing tables), here we briefly introduce P-Grid. P-Grid is a distributed data structure based on the principles

of distributed hash tables (DHT) [16]. As any DHT approach P-Grid is based on the idea of associating peers with data keys from a key space  $\mathcal{K}$ . Without constraining general applicability we will only consider binary keys in the following. In contrast to other DHT approaches we do not impose a fixed or maximal length on the keys, i.e., we assume  $\mathcal{K} = \{0, 1\}^*$ .

In the P-Grid structure each peer  $p \in Peers$  is associated with a binary key from  $\mathcal{K}$ . We denote this key by  $path(p)$  and will call it the path of the peer. This key determines which data keys the peer has to manage, i.e., the keys in  $\mathcal{K}$  that have  $path(p)$  as prefix. In particular the peer has to store them. In order to ensure that the complete search space is covered by peers we require that the set of peers' keys is *complete*. The set of peers' keys is complete, if for every prefix  $s_{pre}$  of the path of a peer  $p$  there exists a peer  $p'$ , such that  $path(p') = s_{pre}$ , or there exist peers  $p_0$  and  $p_1$ , such that  $s_{pre}0$  is a prefix of  $path(p_0)$  and  $s_{pre}1$  is a prefix of  $path(p_1)$ . Naturally, one of the two peers  $p_0$  and  $p_1$  will be  $p$  itself in that case. Completeness is guaranteed by P-Grid's construction algorithm. We do not exclude the situation where the path of one peer is a prefix of the path of another peer. This situation will occur during the construction and reorganization of a P-Grid. Ideally, this situation is avoided, since otherwise peers with shorter paths (prefixes) will have high storage loads and thus load balancing is compromised. Thus, any algorithm for maintaining a P-Grid should eventually converge to a state where the P-Grid is *prefix-free*, i.e., for peers  $p_0$  and  $p_1$  we have  $path(p_0) \not\subseteq path(p_1) \wedge path(p_1) \not\subseteq path(p_0)$ , where  $s \subseteq s'$  denotes the prefix relationship among strings  $s$  and  $s'$ .

We also allow multiple peers to share the same paths, in that case we call the peers replicas. The number of peers that share the same path is called the *replication factor* of the path. Replication is important to support redundancy and thus robustness of a P-Grid in case of failures and to distribute workload when searching in a P-Grid.

To be able to search in P-Grid, peers maintain *routing tables*. The routing tables are defined as (partial) functions  $ref : Peers \times N \rightarrow \{Peers\}$  with the properties

1.  $ref(p, l)$  is defined for all  $p \in Peers$  and  $l \in N$  with  $1 \leq l \leq |path(p)|$
2.  $ref(p, l) \subseteq Peers_{s_1 s_2 \dots s_{l-1} (1-s_l)}$  with  $path(p) = s_1 s_2 \dots s_{l-1} s_l \dots s_k, k \geq l$

where  $Peers_t = \{p \in Peers | t \subseteq path(p)\}$  for  $t \in \mathcal{K}$ .

For the same association of peers with paths, different P-Grids can be obtained depending on the choice of  $ref(p, l)$ . Algorithms for construction and maintenance of a P-Grid have been introduced in [2]. The construction algorithm

takes care of dynamically partitioning the search space, such that approximately equal number of keys belong to each partition. The maintenance phase dynamically splits or joins the partitions to preserve the storage load-balancing, and works in harmony with the route maintenance mechanisms [3].

Having multiple references at each level  $l$  again is necessary to guarantee robustness of the data structure. In the following,  $r$  denotes the maximum number of references maintained at each level. The search algorithm for locating data keys indexed by a P-Grid is defined as follows: Each peer  $p \in Peers$  is associated with a location  $loc(p)$  (IP address in the network). Searches can start at any peer. Peer  $p$  knows the locations of the peers referenced by  $ref(p, l)$ , but not of other peers. Thus the function  $ref(p, l)$  provides the necessary routing information to forward search requests to other peers in case the searched key does not match the peer's path. Let  $t \in \mathcal{K}$  be the searched data key and let the search start at  $p \in P$ . Algorithm 1 shows P-Grid's basic recursive search algorithm.

---

**Algorithm 1** Search in P-Grid:  $search(t, loc(p))$

---

```

1: if  $path(p) \subseteq t$  then
2:   return( $loc(p)$ );
3: else
4:   determine maximal  $l$  such that  $t_1 \dots t_{l-1}(1 - t_l) \subseteq path(p)$ ;
5:    $r$  = randomly selected element from  $ref(p, l)$ ;
6:   search( $t, loc(r)$ );
7: end if

```

---

Algorithm 1 always terminates successfully, if the P-Grid is complete and all peers are reachable. Due to the definition of  $ref$ ,  $search(t, loc(p))$  will always find the location of a peer at which the search can continue (use of completeness). With each invocation of  $search(t, loc(p))$  the length of the common prefix of  $path(p)$  and  $t$  increases at least by one. Therefore the algorithm always terminates.

In case of an unreliable network, it may occur that a search cannot continue since the peer  $r$  selected from the routing table is not available. Then alternative peers can be selected from the routing table to continue the search.

## 2.3 de Bruijn Networks and routing mechanisms [8, 17]

We use the notation used by Ganesan and Pradhan [8] for the description of de Bruijn networks. The order- $n$  binary de Bruijn graph  $D(n)$  consists of the set of nodes  $Z_2^n = \{0, 1\}^n$ . For  $\alpha, \beta \in Z_2$  and  $x \in Z_2^{n-2}$ , each node  $\alpha x \beta$  is connected to:

- Node  $x\beta\alpha$  via a shuffle arc.

- Node  $x\beta\bar{\alpha}$  via a shuffle-exchange arc.
- Node  $\beta\alpha x$  via a inverse-shuffle arc.
- Node  $\bar{\beta}\alpha x$  via a inverse-shuffle-exchange arc.

Sometimes some of these arcs form loops, and the de Bruijn network corresponding to the de Bruijn graph excludes these redundant arcs.

*Definition:* The directed de Bruijn network is the directed graph of the nodes with the shuffle-arc and shuffle-exchange-arc as the outgoing edges, and the inverse-arcs as the incoming edges (excluding loops). The undirected de Bruijn network thus comprises all the edges of the directed de Bruijn network being bidirectional.

The definitions can be extended for a order- $n$   $k$ -ary de Bruijn network as well, but for the rest of the paper we'll work with the binary network for simplicity, unless explicitly mentioned otherwise. Moreover, by default we'll refer to the directional de Bruijn graph and network, and will specify explicitly whenever we refer to the undirected one, as has been the practice in related literature.

### 2.3.1 Optimal routing in directed de Bruijn network [18]

The greedy routing scheme is optimal for directed de Bruijn network, where optimality implies that routing is done along a path of length less than or equal to the diameter of the network, which is  $n$ . Thus, at every step of routing, the edge corresponding to a left-shift of the current binary string, appended with the next bit of the destination node as the last bit is chosen. Thus, routing from  $V = v_1 v_2 \dots v_n$  to  $W = w_1 w_2 \dots w_n$  will be along the path  $v_1 v_2 \dots v_n \rightarrow v_2 v_3 \dots v_n w_1 \rightarrow v_3 \dots v_n w_1 w_2 \rightarrow \dots \rightarrow w_1 w_2 \dots w_n$  [18].

In the undirected de Bruijn network, such left-shifts are called *L-operation*, and similarly *R-operation* too can be defined, which forms the basis for shortest path routing in undirected de Bruijn network, as described below.

### 2.3.2 Shortest path routing in undirected de Bruijn network [15]

For the undirected de Bruijn network, Mao and Yang provided the shortest path routing algorithm [15] by modifying the original routing algorithm for undirected de Bruijn networks proposed by Pradhan and Reddy [17]. We provide a brief overview of the algorithm for de Bruijn shortest path routing in an undirected network that is based on local computation at the source. For further details and proof of the correctness of the algorithm, please refer to [15], from which we borrow the notations for this subsection.

Let  $X$  be a common substring in source  $V$  and destination  $W$ . Then,  $V$  may be represented as  $V_L X V_R$ , and  $W$  as  $W_L X W_R$ , where each of  $X$ ,  $V_L$ ,  $V_R$ ,  $W_L$  and  $W_R$  may be empty. Let a left shift be defined as a *L-operation*. e.g.,  $11100 \rightarrow 1100*$  Then routing from  $V$  to  $W$  may be done by  $|V_R|$  R-operations, then perform  $|W_R|$  L-operations to correct the bits on the right side of  $X$ , then perform  $|W_L|$  L-operations, and then  $|W_L|$  R-operations to complete the routing. This route is the *RLR path* from  $V$  to  $W$  for the common substring  $X$ , and length of this path can be locally computed at source  $V$ . Similarly a *LRL path* too can be defined, and its length be determined. Then the shortest path from  $V$  to  $W$  will comprise of the shortest of the shorter path computed for all possible common substrings  $X$  of  $V$  and  $W$ .

### 3 de Bruijn Networks for DHTs

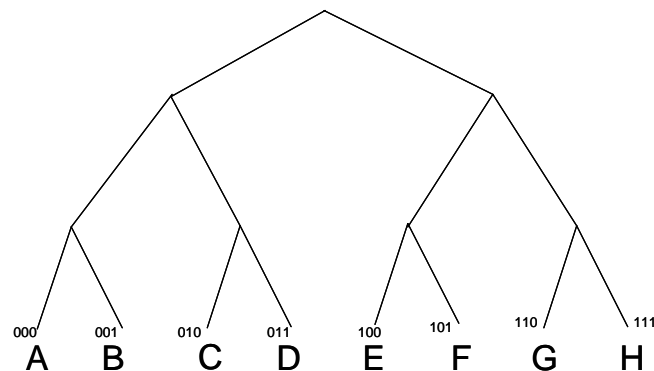
While de Bruijn networks have been in use for parallel computing, interconnection networks and multi-processor chip designing, all these have been static settings. Koorde [9] pioneered in using the de Bruijn network in the context of P2P DHTs. It used Chord as a substrate to build Koorde. de Bruijn network may also be built from scratch, as has been elaborated in CAN-d2B [7]. The simplicity of such degree optimal network means implementation is easy, either on top of an existing DHT substrate, or from scratch, while search is efficient. Figure 1 shows how the same partitioning of key-space may be realized for some traditional DHTs (P-Grid and CAN), and how de Bruijn routing may be used instead.

Subfigure 1(a) shows an instance of P-Grid. P-Grid uses greedy prefix based routing, and the routing process emulates a virtual tree, though there is no hierarchy in P-Grid. For instance peers with prefix 000 are stored in the partition named  $A$ . A peer responsible for partition  $A$  thus keeps in its routing table random reference for the other half of the subtree at each level. So for the first bit, prefix 1, it needs to store any peer that is responsible for any of the partitions  $E, F, G$  or  $H$ . For the prefix 01, it similarly stores reference of any peer that is responsible for the partitions  $C$  or  $D$ , while for prefix 001, it stores reference for a peer responsible for partition  $B$ , while for prefix 000, it itself is responsible to store the keys. Note that multiple peers can be responsible for the same partition, and are called replicas, which thus provide redundancy and robustness, and probabilistic consistency of these replicas is provided using a hybrid push and pull gossiping mechanism [5]. Similarly, multiple random routing references for each prefix may be stored for greater resilience.

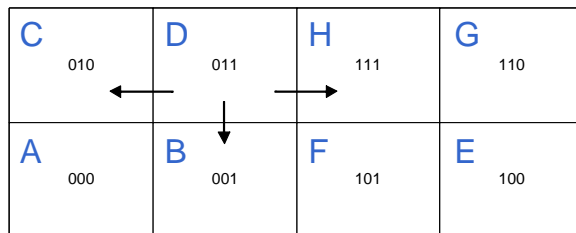
Subfigure 1(b) shows a 2-dimensional CAN network for

the same partitioning of the key space. Unlike prefix resolution in P-Grid, CAN uses a greedy algorithm to resolve any one of the possible bits, thus a peer in partition  $D(011)$  maintains routes to  $C(010)$ ,  $B(001)$  and  $H(111)$

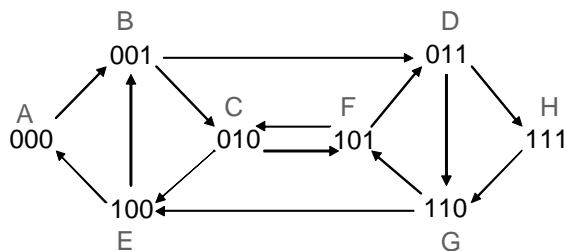
Subfigure 1(c) shows de Bruijn routes for the same partitioning. For instance,  $C(010)$  has routes for the partitions corresponding to left-shift 10 appended with 0/1, that is  $E(100)$  and  $F(101)$ . Koorde [9] and CAN-d2B [7] use similar principles for the routing network.



(a) P-Grid



(b) CAN



(c) de Bruijn

Figure 1: Some DHT routing possibilities

P-Grid is a traditional DHT with non-constant routing table (typically logarithmic), but with logarithmic search cost with high probability [2]. A  $d$ -dimensional CAN has a constant  $2d$  routing entries, but the search cost in CAN is logarithmic only if it has logarithmic dimension. On the other hand, de Bruijn network has constant (2) routing entries, and

still logarithmic search cost. These capture a whole spectrum of DHT designing. Another aspect of de Bruijn network is, like P-Grid and CAN, its simplicity, in contrast with the complexity and need of approximate global-knowledge in Viceroy [12], which emulates a butterfly network and pioneered the family of constant sized routing table DHTs.

So far, the prospects of de Bruijn networks look all rosy. Indeed, the degree optimality implies that cost of route maintenance will also be marginal. These advantages make it the ultimate choice for routing in distributed hash tables. Or does it? We explore a critical limitation of de Bruijn routing as compared to traditional DHTs in the remaining of the paper.

## 4 Storage load balancing in DHTs

As mentioned previously, for a wide range of applications using range queries, it is desirable to preserve the natural ordering of resources in the hashed key space. This will lead to skewed key distribution, such that in order to balance storage load among peers, an uneven partitioning of the key space will be desirable.<sup>2</sup> Figure 2 shows an example to illustrate the point. If a four-bit key space has key distribution with relative frequencies as shown in the bar chart, then the desirable partitioning of the key space is as shown in the upper part. In the figure, we also show the corresponding P-Grid search structure for the partition so formed.

### 4.1 Logarithmic searches in P-Grid with non-logarithmic depth

A random P-Grid that balances storage load per peer also provides efficient (logarithmic) searches. Note that the P-Grid shape is determined by the key distribution, and by random, we mean that the choice of routing entries at each peer for each level is randomly chosen from all the possible options. Self-organizing algorithms for construction of a P-Grid conforming to key distribution using only local knowledge, and randomization of P-Grid routing tables is elaborated in [2]. Here we provide a summary of the results.

Searches in an arbitrarily shaped P-Grid (constructed [2] to ensure storage load balancing in presence of arbitrary key distribution) will be successful using logarithmic messages with high probability. That is to say, searches will be efficient no matter how the key space is partitioned. This is a

<sup>2</sup>It is possible to formulate a more ambitious goal of balancing query load by considering query distribution over the key space, however, it is taken care of by adapting the replication factor accordingly. Balancing replication factor is elaborated in [2] and for brevity and simplicity we exclude from this paper the issue of replication load-balancing in DHTs.

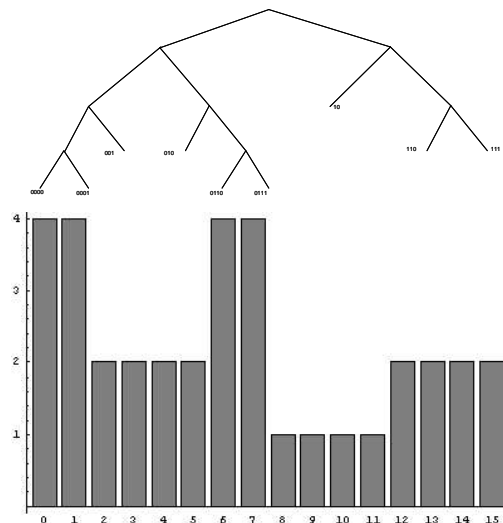


Figure 2: Storage load-balanced partitioning of search space

consequence of the random choice of routing references at each peer ensured by the P-Grid construction and maintenance algorithm [2]. As a consequence of this, and the flexibility obtained by disentangling the peer identifiers from the associated keys (Section 2.1), and thus the flexibility of arbitrary key space partitioning and assignment of peers to these partitions, we can achieve the conflicting goals of storage load balancing as well as efficient searches.

Thus, such a property of simultaneously balancing storage load, while preserving key ordering (which leads to arbitrarily skewed distributions) and search efficiency becomes a benchmark for comparing the properties and usefulness of any other P2P system. In the rest of this paper, we show that de Bruijn routing based P2P networks do not meet these objectives. This is essentially a consequence of the properties of de Bruijn graph, including its deterministic nature and assumption of homogeneity of the node-space of the graph, which makes it unsuitable to be applied in the context of P2P systems.

## 5 de Bruijn routing revisited

In the previous section we identified the benchmark for comparing DHTs performance vis-a-vis load-balancing and search efficiency in presence of arbitrary key distribution. Next we seek to know whether such properties can be expected from de Bruijn routing based networks. We provide examples to demonstrate limitations of de Bruijn routing for an unevenly partitioned key-space.

In Figure 3, we look back at how the de Bruijn routing ta-

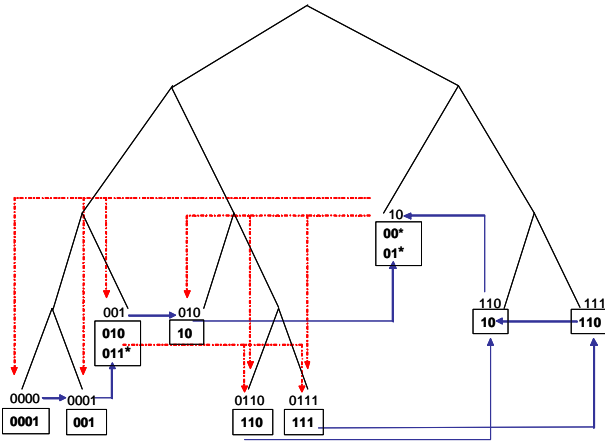


Figure 3: de Bruijn routing when key space partitioning is uneven

bles will be like, provided the key space is partitioned as in the previous example of Figure 2. The logical routing table entries for each peer is shown to be enclosed in the rectangles. For instance peers responsible for the partition 0000 have route to partition 0001. Similarly, peers of partition 0001 should ideally have routes to 0010 and 0011. However, since the granularity of partition is different, there is only one zone 001 to be routed to. Since de Bruijn routing is essentially like a shift-register, the effect of moving from a zone of finer granularity (longer key) to a coarser one (shorter key) is that some information is lost, and if in future, a routing from a coarser partition to a finer one is required, then there will be a difficulty, as elaborated next.

In this example, peers at partition 10 will have the de bruijn routes 00 and 01, however, since each of 00 and 01 are further partitioned, hence a peer responsible for zone 10 will essentially have routes of the form 00\* and 01\*. The possible routing edges are shown in the figure using the perforated directed edges. In this case, there are two ways to populate the routing tables.

(1) Choose any one of the partitions with prefix 00 (i.e. partitions 001, 0001 and 0000. This may lead to the problem that not all partitions will have incoming edges (in directed de Bruijn network), such that these partitions will not be accessible to the network at all. For instance, if peer in partition 001 chooses 0110, peer in partition 10 chooses 0111 and 0001 as their de Bruijn routes, partition 0000 will have no incoming edge. One possible way to mitigate this effect will be to provide a back-edge to every incoming edge. Note that while this will be like the undirected de Bruijn network, it will not exactly be an undirected de Bruijn network. Particularly, the back-edges can not be computed locally, as

were inverse-shuffle(-exchange) arcs, but indeed will have to provide back-edges to the incoming shuffle(-exchange) arcs from other partitions. More importantly, the routing algorithms discussed in Section 2.3 will no more be efficient, since the diameter bounds of de Bruijn graphs [6, 17] will not hold good any more in the event of uneven partitioning of the search space.

(2) The other choice of routing entries will be to have routes to all possible sub-partitions, in this case, 001, 0001 and 0000. This however will imply that the peers will no more have constant outdegree. Additionally, such a system will depend on global knowledge about partitioning granularity for the rest of the key space.

## 5.1 A worst case scenario

Using de Bruijn routing in the worst case (in terms of key distribution) may turn out to be like sequential search, as shown in Figure 4, unlike traditional DHTs like P-Grid, which will still have logarithmic searches with high probability. In P-Grid, the conflicting goals were achieved because of the randomization in the routing process. Since de Bruijn routes are by definition deterministic, it is not surprising that in the event of skewed distribution of keys and uneven partitioning of the key space, de Bruijn routing fails to meet the conflicting goals simultaneously, because it has to make some randomized decisions if it has to retain constant outdegree (choice 1, as elaborated above).

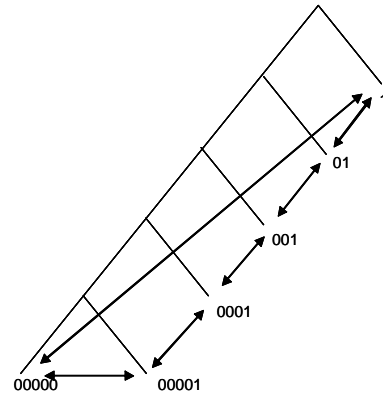


Figure 4: A worst case scenario

These examples demonstrated that de Bruijn routing based DHTs will lose the desirable properties if the key-space is not evenly partitioned. Even if the key-space is partitioned evenly, there may not always be enough partitions  $N$  to satisfy  $N = K^i$  for some  $i$  for a  $K$ -ary de Bruijn network. In such cases also more than  $K$  outdegree is required at some peers. The later problem was exposed in CAN-d2B [7], but

is not so critical as is the effect of uneven key-space partitioning, as elaborated in this paper.

## 5.2 Summary

Peer-to-peer systems increasingly need to accommodate uneven distribution of keys, particularly if ordering of natural names is to be preserved in the key space. Such arbitrary key distribution leads to either of uneven (storage) load distribution, or else uneven partitioning of key spaces. Since load balancing is an important and desirable property for any distributed system, the system should be able to accommodate arbitrary key distributions by dynamically partitioning the key space among the participating peers. In the event of such dynamic partitioning of key space, it has been shown that there exists DHT based P2P systems with non-constant routing table size (P-Grid [2]), which nonetheless retain logarithmic search efficiency. However because of the deterministic nature of the de Bruijn graph, it lacks the flexibility to preserve search efficiency in presence of uneven partitioning of the key space, thus severely restricting their practical utility. This is despite the otherwise desirable properties that de Bruijn networks are degree optimal and logically simple (when key space is evenly partitioned), and hence also easy to implement.

## 6 Conclusions

The interest in de Bruijn networks in the community of distributed systems and parallel computing is quite old, particularly in the VLSI designing of multiprocessor systems [22]. Such systems are static, and de Bruijn network has been used for static interconnection networks. Unlike P2P systems, they do not have to deal with storage load balancing for a dynamic and arbitrary load distribution.

Koorde proposed use of de Bruijn networks for a simple degree optimal solution, followed by CAN-d2B [7]. These initial papers make simplifying assumptions, particularly that of uniform key distribution. In this paper, we showed why such systems will fail to simultaneously meet all the goals under more realistic conditions. There are other constant routing table based P2P systems. Viceroy pioneered constant degree P2P networks, but is considered to be too complicated for implementation. Moreover, the proof of efficiency of Viceroy also depends on uniform key distribution.

It is increasingly obvious though that the assumption of an uniform key distribution severely restricts the utility of the P2P systems, and thus arbitrary key distributions should be efficiently handled. Even traditional DHTs have only recently started to address this issue. Thus, we end this paper

with an outstanding question, as to *whether it is possible to construct a constant routing table sized DHT which meets the conflicting goals of storage load balancing and search efficiency for an arbitrary and changing key distribution?* As has been seen in many other domains, that randomization is often the best way to handle randomization. Hence, it appears that the possible approach would be to use some sort of randomization in the choice of a constant number of routing entries. Symphony [14] based on Kleinberg's proposal of small world networks [10] is such a candidate system (it typically has poly-logarithmic rather than logarithmic search cost). How distributed hash tables using small world routing will perform in presence of skewed data distribution and hence uneven partitioning of key-space is an interesting facet that needs further study, and defines part of the future work. Until then, the traditional DHTs with non-constant routing table size (typically logarithmic) seem to be the safest bet.

## References

- [1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001.
- [2] K. Aberer, A. Datta, and M. Hauswirth. The Quest for Balancing Peer Load in Structured Peer-to-Peer Systems. Technical Report IC/2003/32, EPFL, 2003.
- [3] K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in Peer-to-Peer systems. *To be published in IEEE Transactions on Knowledge and Data Engineering*, 2004.
- [4] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1), 2002.
- [5] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.
- [6] A-H. Esfahanian and S.L. Hakimi. Fault-Tolerant Routing in De Bruijn Communication Networks. *IEEE Transactions on Computers*, 34(2), 1985.
- [7] P. Fraigniaud and P. Gauron. The content-addressable network d2b. Technical Report Technical Report LRI 1349, Univ. Paris-Sud, 2003.
- [8] E. Ganesan and D. K. Pradhan. Wormhole Routing in De Bruijn Networks and Hyper-DeBruijn Networks. In *ISCAS*, 2003.

- [9] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *In 2nd International Peer To Peer Systems Workshop (IPTPS)*, 2003.
- [10] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.
- [11] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.
- [12] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.
- [13] G. S. Manku. Routing networks for distributed hash tables. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing (PODC)*, pages 133–142. ACM Press, 2003.
- [14] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *4th USENIX Symposium on Internet Technologies and Systems, USITS*, 2003.
- [15] J. Mao and C. Yang. Shortest Path Routing and Fault-Tolerant Routing on de Bruijn Networks. *Networks*, 35(3), 2000.
- [16] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.
- [17] D. K. Pradhan and S. M. Reddy. A fault-tolerant communication architecture for distributed systems. *IEEE Transactions on Computers*, 31, 1982.
- [18] Darcy L. Quesnel. De Bruijn Networks, 1995.
- [19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM*, 2001.
- [20] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. Technical Report Technical Report UCB//CSD-03-1299. The University of California, Berkeley, Univ. Paris-Sud, 2003.
- [21] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 2001.
- [22] M. Samatham and D. Pradhan. The de bruijn multi-processor network: a versatile parallel processing and sorting network for VLSI. *EEE Trans. on Computers*, 38(4), 1989.
- [23] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM*, 2001.