The Content-Addressable Network D2B

Pierre Fraigniaud[*‡] Philippe Gauron[†‡]

Abstract

A content-addressable network (CAN) is a distributed lookup table that can be used to implement peer-to-peer (P2P) systems. A CAN allows the discovery and location of data and/or resources, identified by keys, in a distributed network (e.g., Internet), in absence of centralized server or any hierarchical organization. Several networks have been recently described in the literature, and some of them have led to the development of experimental systems. We present a new CAN, called D2B. Its main characteristics are: simplicity, provability, and scalability. D2B allows the number of nodes n to vary between 1 and $|\mathcal{K}|$ where $\mathcal{K}$ is the set of keys managed by the network. In term of performances, any join or leave of a user implies a constant expected number of link modifications, and, with high probability (w.h.p.), at most $O(\log n)$ link modifications. The latency of a lookup routing is $O(\log n)$, w.h.p., in the sense that a key is at most $O(\log n)$ hops away from a consumer. A join involves key redistribution among two nodes (which is the minimum possible), including the node joining the system. Similarly, a leave involves key redistribution among at most three nodes. The set of keys is fairly distributed among nodes, in the sense that every node is responsible for an expected number of $|\mathcal{K}|/n$ keys, and, w.h.p., $O(|\mathcal{K}| \log n/n)$ keys. The traffic load incurred by lookup routing is also fairly distributed, and the expected congestion of a node is $O((\log n)/n)$. Finally, a parameter d allows a trade-off between the degree of the network, which increases linearly with d, and the diameter of the network, which decreases logarithmically with d. Hence, a large d allows faster lookup routing, at the price of a slight increase of the latency for joining and leaving the network. We believe that these properties make D2B currently the most promising network for a practical and efficient use of CANs.

Keywords: peer-to-peer system, content-addressable network, dynamic network, distributed hash table, de Bruijn graph.

Technical Report TR-LRI-1349
January 2003
Laboratoire de Recherche en Informatique
Université Paris-Sud
91405 Orsay cedex
France

[*] CNRS, Laboratoire de Recherche en Informatique (LRI), Université Paris-Sud, France. *http://www.lri.fr/~pierre*. E-mail: *pierre@lri.fr*.

[†] Laboratoire de Recherche en Informatique (LRI), Université Paris-Sud, France. *http://www.lri.fr/~gauron*. E-mail: *gauron@lri.fr*.

[‡] Both authors are supported by the Action Spécifique *Dynamo* ("Analyse Structurelle et Algorithmique des Réseaux Dynamiques") of CNRS, and by the project "Grand Large" of INRIA.

1 Introduction

A *content-addressable network* (CAN) is a distributed logical network allowing the discovery and location of data and/or resources identified by keys[1], in a physical network (e.g., Internet). A CAN is described by a pair $(\mathcal{K}, G)$ where $\mathcal{K}$ is a set of keys, and $G = (V, E)$ is a graph. To every node $v \in V$ of the graph G is assigned a subset of keys K_v such that $\cup_{v \in V} K_v = \mathcal{K}$. More precisely, node v stores a *lookup table* which contains information related to all keys $\kappa \in K_v$. For instance, the lookup table of node v contains the IP-address of a node holding the resource of key κ, for every $\kappa \in K_v$ (see Table 1(a)). There is a directed edge $(u, v) \in E$ from node u to node v in G if u is "aware of" the physical address (e.g., the IP-address) of node v. In other words, node u stores a *routing table* (see Table 1(b)) which contains the IP-address of nodes to which it can set up a direct connection (e.g., a TCP connection), and transmit messages directly. Finally, there is a distributed routing protocol R in G which is in charge of transmitting requests (e.g., for resources) in the logical network. As opposed to usual networks, routing in a CAN is not performed according to a destination label, but according to a resource key. More precisely, R is a function that maps $V \times \mathcal{K}$ to V, under the restriction that, for any key κ, and any node u, $R(u, \kappa)$ is either u or an out-neighbor v of u in G. If $v \neq u$, $R(u, \kappa) = v$ is interpreted as: at the current node u, a lookup for key κ is to be forwarded through edge (u, v). If $R(u, \kappa) = u$ then $\kappa \in K_u$. Therefore, the key κ should provide enough information to allow routing, from any source, to some node holding information about the resource of key κ, i.e., a node such that κ is an entry of its lookup table.

A CAN is an underlying mechanism that can be used to implement *peer-to-peer* (P2P) systems. A user of such a system is connected to a node u, and has access to every resource stored in the system, generally via a connection to a remote node $v \neq u$ storing the resource. P2P systems must perform in absence of any centralized or hierarchical structure, and CANs are good ways to implement distributed lookup protocols for finding out IP-addresses of nodes storing requested resources. Here is a possible scenario. Assume that a user connected to node u is looking for some resource item whose key is $\kappa \in \mathcal{K}$. A lookup message is then sent from u. The format of such a message could be $\langle \textit{lookup}, @_u, \kappa \rangle$ where $@_u$ is the IP-address of node u. Based on κ, the distributed routing protocol R routes the message in the logical network G, from u to some node w satisfying $\kappa \in K_w$. Depending on the implementation, w may or may not store the requested item. In case w does not store the requested item, it stores a lookup table which, for any key κ in K_w, returns the IP-address of a node v storing the resource item of key κ. Hence, in our scenario, w sends to u the IP-address $@_v$ of node v. Once u receives the IP-address $@_v$, it contacts v to retrieve the requested item. Alternatively, w can contact v directly, get the item there, and forward it to u. The advantage of this latter solution is that u does not know who stores the requested item, and v does not know who requested that item, preserving anonymity.

For instance, let $\mathcal{K} = [0, M-1]$, and G be the ring $\{u_0, \ldots, u_{n-1}\}$, where $u_i \in [0, M-1]$, and $u_0 < u_1 < \ldots < u_{n-1}$. There is an outgoing edge from u_i to $u_{i+1 \bmod n}$, $i = 0, \ldots, n-1$. For $i > 0$, node u_i is responsible for all keys κ such that $u_{i-1} < \kappa \leq u_i$, and u_0 is responsible for keys κ such that $0 \leq \kappa \leq u_0$ or $u_{n-1} < \kappa \leq M-1$. Routing in the ring is easy: a request for a key κ is sent from its source u_i to $u_{i+1 \bmod n}$, then from $u_{i+1 \bmod n}$ to $u_{i+2 \bmod n}$, and so on until the request reaches node u_j with $u_{j-1} < \kappa \leq u_j$, or node u_0 if $0 \leq \kappa \leq u_0$ or $u_{n-1} < \kappa \leq M-1$. If the u_i's are chosen uniformly at random, then every node is responsible for roughly the same amount of keys. However, this solution suffers from a major drawback: the average length (number of hops)

[1] This paper does not consider the question of assigning keys to resources, and we refer to [22] for a discussion about this important aspect of the problem.

Lookup table

Resource-Key	IP-address
001100...1010	172.174.15.2
010100...0011	115.215.42.1
⋮	⋮
⋮	⋮
⋮	⋮
011001...1101	123.13.2.1

(a)

Routing table

Node-Label	IP-address
1100	129.175.15.1
110100	132.205.45.1
1101010	147.83.2.4
1101011	134.117.5.8
11011	129.199.96.32

(b)

Table 1: Lookup and routing tables. In this example, resource-keys are m-bit strings, and node-labels are binary strings of arbitrary length.

of a route followed by a lookup message is $\Theta(n)$, where n is the number of nodes currently in the system, resulting in a latency much too high for a practical use (a P2P system should support thousands of users).

Another solution consists to connect every u_i to u_j for all $j \neq i$, i.e., G is the complete graph. Routing is then trivial, but every node u_i must store a routing table with $n - 1$ entries, one for every IP-address of the $n - 1$ other nodes. Such a table would be much too large for any practical use. Also, a new node joining the system would have to distribute its IP-address to all nodes already in the network, resulting in $\Omega(n)$ messages exchanged in the system, and $\Omega(n)$ link modifications, just for one join.

A third solution consists to organize the nodes as a complete binary tree (where the last level may be incomplete). In case of a join or a leave, a small number of messages are sufficient to reorganize the network. Lookup routing is fast, i.e., at most $O(\log n)$ steps. However, the congestion at the root of the tree is huge: roughly half of the lookups are routed through the root!

None of these three solutions scale, either because the diameter of the network is too large, or because the degrees of the nodes are too large, or because the connectivity of the network is too small.

1.1 Statement of the problem

The design of a CAN consists to define (1) a set of keys, (2) a distributed assignment of keys to nodes, (3) a set of dynamic connections between nodes, and (4) a distributed routing mechanism using these connections. The design is subject to the following constraints:

1. At any time, all nodes currently in the system are mutually reachable;
2. Any node can leave the system at any time, and any node can join the system at any time;
3. At any time, keys are evenly distributed among nodes;
4. Lookups are performed on a key-basis, i.e., the route from the consumer of a resource to a supplier of information concerning that resource is set up according to the knowledge provided by the key of the resource only;

5. The *lookup latency* is small, that is the time to reach a node responsible for any given key from any given consumer is small, i.e., the lookup path length must be short;

6. The *traffic load* incurred by lookups routing through the system should be evenly distributed among nodes;

7. The *update time* is small, that is the update of the routing tables and of the connection links due to a leave or a join must be fast;

8. The redistribution of keys due to a leave or a join must be fast.

Constraint 1 is constitutive of a network insuring exhaustive search. Constraint 2 expresses the dynamism of the network, typical of a peer-to-peer system. The next two constraints are related to the primal role of CANs, which is to provide the storage of, and the access to a huge amount of resources. The large lookup table corresponding to these resources must be fairly distributed among the nodes (Constraint 3), and the access to any resource must be driven by its key only (Constraint 4). The last four constraints are related to the performances of the network. More precisely, the user-perception of the quality of a P2P system depends on Constraint 5 because a consumer wants to be served as quickly as possible. To satisfy this requirement, the length of the routing paths must be as short as possible, and the local routing tables must be small. Constraint 6 specifies that no server should be a bottleneck on the performances of the service. The *congestion* of a node is intended to measure the probability that it is involved in a search for a random key, from a random source-node. It is defined as the ratio *load* over total number of pairs (consumer,key), that is the number of lookups that can pass through the node, divided by $n|\mathcal{K}|$. Constraints 7 and 8 control the "degree of dynamism" supported by the system: complex local updates, or transfer of large amounts of data between remote nodes in the logical network, limit the reactivity of the system. To achieve Constraint 7, the number of control messages exchanged between nodes due to a join or a leave must be small. To achieve Constraint 8, nodes storing "close" sets of keys must be close in the CAN.

1.2 Our Results

We describe a new content-addressable network, called D2B. The underlying topology of D2B is the *de Bruijn* graph [7]. It is known that the static version of the de Bruijn graph allows to construct large networks of fixed degree and small diameter [4]. We show that one can use the de Bruijn graph to design *dynamic* networks as well. The expected performances of D2B are summarized in Table 2, in comparison with other CANs previously proposed in the literature.

A first remarkable characteristic of D2B is its simplicity (as simple as, e.g., Chord [22]). All nodes of D2B play roughly the same role in the network, and routing is simple and naturally well balanced. This is in contrast with, e.g., Viceroy [13], in which the 3-phase routing protocol induces an underlying hierarchy among the different levels of nodes. D2B is also perfectly scalable, and the number of nodes can take any value between 1 and 2^m where m is any integer whose value is fixed *a priori* (say $m = 128$ or 256 for a practical use).

The set of keys managed by an n-node D2B is basically the same as for Chord [22], Viceroy [13], and DMBN [6]: $\mathcal{K} = \{0, \ldots, 2^m - 1\}$, and $1 \leq n \leq |\mathcal{K}|$. As for each of these three CANs, the expected number of keys managed by a node of D2B is $|\mathcal{K}|/n$, and is, with high probability[2], at

[2] In this paper, an event $\mathcal{E}$ occurs with high probability (w.h.p.) if $\text{Prob}(\mathcal{E}) \geq 1 - O(1/n)$.

	Update	Lookup	Congestion
CAN [19]	$O(d)$	$O(dn^{1/d})$	$O(dn^{1/d-1})$
Tapestry [24]	$O(d \log n / \log d)$	$O(\log n / \log d)$	$O((\log n)/n)$
Chord [22]	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
DMBN [6]	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
Small World [10]	$O(1)$	$O(\log^2 n)$	$O((\log^2 n)/n)$
Viceroy [13]	$O(1)$	$O(\log n)$	$O((\log n)/n)$
D2B [this paper]	$O(1)$	$O(\log n)$	$O((\log n)/n)$
d-dimensional D2B	$O(d)$	$O(\log n / \log d)$	$O((\log n)/(n \log d))$

Table 2: Comparison of *expected* performance measures of CANs

most $O(|\mathcal{K}| \log n/n)$.

W.h.p., a lookup initiated from any node reaches the node responsible of the requested key after at most $O(\log n)$ hops. Chord and Tapestry [24] satisfy the same property. The simplified version of Viceroy has, w.h.p., a lookup latency $O(\log^2 n)$. An improved version of Viceroy, including a more sophisticated lookup strategy has, w.h.p., lookup latency $O(\log n)$.

The expected degree of D2B is $O(1)$, and there is a constant expected number of control messages that are exchanged during a join or a leave (all control messages are exchanged between neighboring nodes in D2B). Hence the expected time of any update is constant. W.h.p., an update takes at most $O(\log n)$ time. Viceroy performs better with this respect since the degree of Viceroy can be kept constant. However, this requires the use of an involved "bucket mechanism" that complicates the Viceroy network significantly.

The expected congestion of a node in D2B is $O((\log n)/n)$, and the congestion experienced by any node is, w.h.p., $O((\log^2 n)/n)$. These performances are the same as those of D2B's predecessors, including Chord and Viceroy.

Finally, we can define a d-dimensional version of D2B, for $d \geq 2$. (The basic version of D2B has dimension 2.) The d-dimensional version of D2B is build upon the de Bruijn graph of dimension d, and uses the key space $\mathcal{K} = \{0, \ldots, d^m - 1\}$. Its expected degree is $O(d)$, for an expected diameter $O(\log n / \log d)$. This gives a trade-off between the latency for joining or leaving the network, and the latency of a lookup. Also, a large d increases the connectivity of the network, and thus its robustness against processor crashes. This facility is not offered by Viceroy.

To summarize, we claim that D2B is today the most promising candidate for a practical and efficient use of CANs.

1.3 Related Works

Much attention has been given to the construction of large networks of given maximum degree and given diameter (see, e.g., [14] and the references therein). This problem is known as the (Δ, D)-graph problem. Although solutions for this problem can be practically used for the design of static networks, they do not fit well with the dynamic setting of P2P systems for optimal solutions with $n + 1$ nodes may differ significantly from the solutions with n nodes.

Graphs augmented with "long range contacts", as defined in [23], can be used for the design of CANs. In particular, d-dimensional toruses augmented with long range contacts chosen at random according to the harmonic distribution yield networks in which routing can be performed on a key basis in $O(\log^2 n)$ expected number of steps [10]. However, a lower bound of $\Omega(\log^2 n)$ expected number of steps was shown in [2], and there is no evidence that using another probabilistic distribution for the choice of the long range contacts would bring any improvement. Hence, such networks would yield large latencies for the lookups.

The *ad hoc* network community has recently focused on the construction and management of dynamic networks (see [3] and the references therein). However, the proposed solutions are often very specific of the underlying network technology (e.g., Bluetooth), and routing is not performed on a key basis.

DNS provides a host name to IP-address mapping [15], but relies on a set of special root servers, and DNS names are structured to reflect administrative boundaries. File-sharing services like Napster or Gnutella either use central servers (as Napster), or perform searches by flooding (as Gnutella). Flooding overloads the network[3], and cannot be efficiently used in practice at the Internet scale. Freenet [5] does not assign responsibility for resources to specific servers, and looking for a resource takes the form of searches for cached copies of that resource. It provides anonymity, but prevents from guaranteeing success of a request, and from giving bound on the time it takes for a successful request.

In [16] is described a dynamic network with constant maximum degree, and $O(\log n)$ diameter, w.h.p., under a specific probabilistic model of join and leave. However, as already observed by [13], the construction of [16] does not provide a routing scheme, and the intended application is to disseminate queries rather than to route them. In [8], a CAN with logarithmic diameter and fault-tolerant to an adversary deleting up to a constant fraction of the nodes is described. However, the solution is designed for any fixed value of n, and does not provide for the system to adapt dynamically to a large number of joins or leaves. Moreover, searching generates $O(\log^2 n)$ messages, and the degree is $O(\log n)$. A dynamic version has been presented in [6].

This paper is strongly related to the works in [6, 13, 19, 22, 24]. The CAN described in [19] is based on the d-dimensional torus topology, and uses the key space $\mathcal{K} = [0, 1]^d$. The expected diameter is $O(dn^{1/d})$, and the expected degree is $O(d)$. Tapestry [24] (see also [9, 21]) implements the protocol proposed in [17]. The degree of the induced topology is $O(d \log_d n)$, and its expected diameter is $O(\log_d n)$, where d is the base in which the node IDs are encoded. Chord [22] is based on the hypercube topology, and uses the key set $\mathcal{K} = \{0, \ldots, 2^m - 1\}$. Its expected diameter is $O(\log n)$, and the expected degree is $O(\log n)$. Viceroy [13] uses the same key set as Chord, but is based on the butterfly graph. The simplified version of Viceroy has expected degree $O(1)$, expected diameter $O(\log n)$ (w.h.p., $O(\log^2 n)$), and expected congestion $O((\log n)/n)$ (w.h.p., $O((\log^2 n)/n)$). An improved version of Viceroy, including a more sophisticated lookup strategy, has a diameter $O(\log n)$ with high probability. In addition, a "bucket mechanism" allows to fix the maximum degree of the nodes, which is only bounded by $O(\log n)$, w.h.p., in the simplified version. Viceroy is therefore the first known constant-degree CAN with logarithmic diameter. However, its construction and management are relatively complex, and require sophisticated procedures which might be difficult to implement in a practical setting (more difficult than, e.g., Chord). Finally, DMBN [6] uses the same set of keys as Chord and Viceroy, and is based on the multi-butterfly graph. It has same diameter and degree as Chord, but is also fault-tolerant to an adversary deleting up to a constant fraction of the nodes.

[3] It was mentioned in [20] that roughly 50% of the traffic generated by Gnutella is due to the control traffic, even though searches have been limited to within a certain distance from the consumer

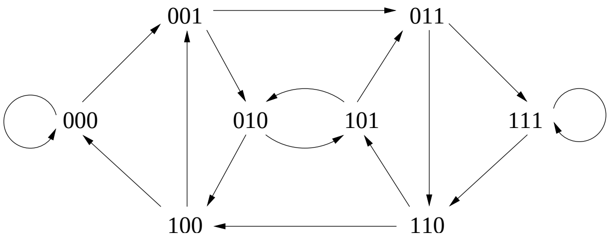

Figure 1: The de Bruijn graph $B(2,3)$

1.4 Organization of the Paper

In the next section, we describe the content-addressable network D2B. Its main properties are derived in Section 3, including a proof of correctness. Section 4 contains a discussion about variants of D2B, including its d-dimensional version, and the ability to choose the label of a node more carefully, in order either to match with the characteristics of the underlying physical network (i.e., Internet), or to decrease the maximum load of the nodes. Finally, Section 5 contains some concluding remarks, including a brief discussion about the implementation of D2B in an experimental global computing platform currently developed at LRI.

2 The Content-Addressable Network D2B

In this section, we describe D2B, including the join and leave procedures. D2B is parametrized by a parameter $d \geq 2$. For the sake of clarity, we describe here the version of D2B for $d = 2$. The description of the d-dimensional D2B network for arbitrary d is given in Section 4.1. We begin the description of D2B by a brief description of the static version of the topology.

2.1 The de Bruijn Graph

The underlying static topology of the d-dimensional D2B, $d \geq 2$, is the de Bruijn graph $B(d,k)$, for $k \geq 1$. $B(d,k)$ is defined from [7]. It is the directed graph whose nodes are all strings of length k on the alphabet $\{0,\ldots,d-1\}$, and there is an edge from any node $x_1x_2\ldots x_k$ to the d nodes $x_2\ldots x_k\alpha$, for $\alpha = 0,\ldots,d-1$. Figure 1 displays $B(2,3)$. Note that there are loops around all nodes $\alpha\ldots\alpha$, $\alpha \in \{0,\ldots,d-1\}$, and that $B(d,k)$ is not vertex-transitive. Nevertheless, we will see that this has no impact on the performances of D2B which are uniformly balanced among nodes. $B(d,1)$ is the complete graph of d nodes, with loops.

$B(d,k)$ has d^k nodes, in-degree and out-degree d, and diameter k. Routing from $x_1\ldots x_k$ to $y_1\ldots y_k$ is achieved by following the route $x_1\ldots x_k \to x_2\ldots x_ky_1 \to x_3\ldots x_ky_1y_2 \to \ldots \to x_ky_1\ldots y_{k-1} \to y_1\ldots y_k$. A shorter route is obtained by looking for the longest sequence that is suffix of $x_1\ldots x_k$, and prefix of $y_1\ldots y_k$. If there is such a sequence $x_i\ldots x_k = y_1\ldots y_{k-i+1}$, then the shortest path from $x_1\ldots x_k$ to $y_1\ldots y_k$ is $x_1\ldots x_k \to x_2\ldots x_ky_{k-i+2} \to x_3\ldots x_ky_{k-i+2}y_{k-i+3} \to \ldots \to x_{i-1}\ldots x_ky_{k-i+2}\ldots y_{k-1} \to y_1\ldots y_k$. De Bruijn graphs are iterated line-graphs. Let us recall that the line-graph $\mathcal{L}(G)$ of a graph G is the graph whose nodes are the edges of G, and there is an edge from node e to node e' in $\mathcal{L}(G)$ if e' is incident to e in G. The iterated line-graph $\mathcal{L}^k(G)$, $k \geq 1$, is defined as $\mathcal{L}^1(G) = \mathcal{L}(G)$, and $\mathcal{L}^{k+1}(G) = \mathcal{L}(\mathcal{L}^k(G))$. One can easily check that $B(d,k) = \mathcal{L}(B(d,k-1))$ by labeling $x_1\ldots x_k$ the node of $B(d,k)$ corresponding to the edge from

$x_1 \dots x_{k-1}$ to $x_2 \dots x_k$ in $B(d, k-1)$. Hence $B(d,k) = \mathcal{L}^{k-1}(K_d)$.

2.2 Overall Description of D2B

The 2-dimensional D2B uses the set $\mathcal{K} = \{0, \dots, 2^m - 1\}$ as key space, also viewed as the set of binary strings of length m. Nodes of D2B are given *labels* that are also binary strings, but of length *at most* m. Thus there are at most 2^m nodes in D2B. Note that this is not a limitation for $m = 128$ (or even 256) in practice, to insure a number of keys much larger than the number of IPv4 addresses (see [24]). For instance, a system with a million nodes uses node-labels on roughly 20 bits. The *value* of a node u labeled $x_1 \dots x_k$, $x_i \in \{0,1\}$, is $val(u) = 2^{m-k} \cdot \sum_{i=1}^{k} x_i 2^{k-i}$.

Definition 2.1 *A* universal prefix set *is a set S of binary words such that, for any infinite word $\omega \in \{0,1\}^*$, there is a unique word in S which is a prefix of ω. The empty set is also a universal prefix set.*

For instance, $\{0, 100, 1010, 1011, 11\}$ is a universal prefix set. D2B insures that, at any time, the set of labels of all nodes currently in the network is a universal prefix set.

Key distribution. Node u labeled $x_1 \dots x_k$ is responsible for all keys between $val(u)$ and $2^{m-k}(1 + \sum_{i=1}^{k} x_i 2^{k-i}) - 1$. More explicitly, the key whose binary representation is $\kappa_1 \dots \kappa_m$ is managed by node $x_1 \dots x_k$ currently in the system if and only if $x_1 \dots x_k$ is a prefix of $\kappa_1 \dots \kappa_m$. Hence, a node labeled $x_1 \dots x_k$ is responsible for 2^{m-k} keys. Conversely, a node responsible for 2^q keys has a label on $m - q$ bits. All keys are assigned since, by construction, the node-labels form a universal prefix set.

Routing connections. At any given time, node labeled $x_1 \dots x_k$ has either a unique out-neighbor of the form $x_2 \dots x_j$, $j \leq k$, or (exclusive) several out-neighbors, of the form $x_2 \dots x_k y_1 \dots y_\ell$ where $1 \leq \ell \leq m - k + 1$. In the latter case, the set of sequences $y_1 \dots y_\ell$ forms a universal prefix set. In particular, if $x_2 \dots x_k y_1 \dots y_\ell$ is an out-neighbor of $x_1 \dots x_k$, then none of the labels $x_2 \dots x_k y_1 \dots y_i$, $i < \ell$, is currently used in the network. In the remaining part of the paper, an out-neighbor of a node u is simply called a *child* of u. The children of a node labeled $x_1 \dots x_k$ are displayed on Figure 2(a). In this example, node $x_1 \dots x_k$ has five children labeled $x_2 \dots x_k 0$, $x_2 \dots x_k 100$, $x_2 \dots x_k 1010$, $x_2 \dots x_k 1011$, and $x_2 \dots x_k 11$. In the network, there is no node labeled $x_2 \dots x_k 1$, $x_2 \dots x_k 10$, or $x_2 \dots x_k 101$.

Symmetrically, at any given time, node labeled $x_1 \dots x_k$ has in-neighbors, simply called *parents*, of the form $\alpha x_1 \dots x_j$, $\alpha \in \{0,1\}$ and $j \leq k$, or of the form $\beta x_1 \dots x_k y_1 \dots y_\ell$, where $\beta \in \{0,1\}$ and $1 \leq \ell \leq m - k - 1$. In the latter case, the set of sequences $y_1 \dots y_\ell$ forms a universal prefix set. Note that the two forms may coexist simultaneously, but then $\alpha \neq \beta$.

Remark. Because of the loops around nodes $0 \dots 0$, and $1 \dots 1$, the child and parent-connections are slightly different for these two nodes. A node u labeled $\alpha\alpha \dots \alpha$, $\alpha \in \{0,1\}$, has children the nodes labeled $\alpha \dots \alpha y_1 \dots y_\ell$, $\ell \geq 1$, where $y_1 = \overline{\alpha}$. The set of labels $y_2 \dots y_\ell$ is a universal prefix set. The parents of node u are labeled $\overline{\alpha}\alpha\alpha \dots \alpha$ with $j \leq k$ symbols α, or (exclusive) $\overline{\alpha}\alpha\alpha \dots \alpha y_1 \dots y_\ell$, with k symbols α, and $\ell \geq 1$. In the latter case, the set of sequences $y_1 \dots y_\ell$ forms a universal prefix set.

Sibling connections. In addition to the child and parent-connections, children of any node u are linked together by *sibling connections* as follows (see Figure 2(a)). If v is a child of u in D2B, then there is an up-sibling connection from v to w where w is the child of u with the smallest

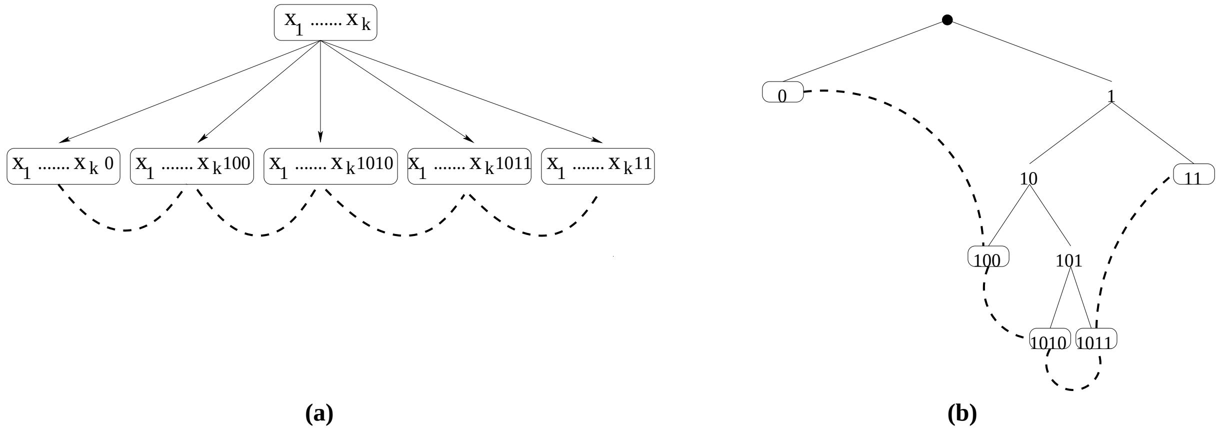

Figure 2: Children (plain arrows) and sibling (doted lines) connections

value $val(w)$ larger that $val(v)$. (If v has the largest value among all children of u, then v has no up-sibling.) Similarly, there is a down-sibling connection from v to w where w is the child of u with the largest value smaller that $val(v)$. (If v has the smallest value among all children of u, then v has no down-sibling.) The sibling connections are used for the purpose of key redistribution, and not for routing.

Routing protocol. Routing in D2B performs roughly the same as in de Bruijn graph. More precisely, let $x_1 \dots x_k$ be the label of a node u in D2B, and let κ be any key. Let $\kappa_1 \dots \kappa_m$ be the binary representation of κ. Let S be the longest binary string that is a suffix of $x_1 \dots x_k$ and a prefix of $\kappa_1 \dots \kappa_m$, possibly $S = \emptyset$. If $S = x_1 \dots x_k$, then u holds κ. Otherwise, if u has a unique child v labeled $x_2 \dots x_j$, then the lookup for key κ is forwarded to this child. If u has several children, then the lookup for key κ is forwarded to the child v labeled $x_2 \dots x_k y_1 \dots y_\ell$ such that $S y_1 \dots y_\ell$ is a prefix of $\kappa_1 \dots \kappa_m$. By the universal prefix set property, such a children exists, and is uniquely defined.

Publication of the keys. A node u of the system aiming to publish a resource computes[4] the corresponding key $\kappa \in \mathcal{K}$, and sends a "request-to-publish message" through the network. The format of such a message is $\langle publish, @_u, \kappa \rangle$, where $@_u$ is the IP-address of u. It is routed like a lookup message, based on the binary representation of κ. When the node responsible for κ receives the message, it places $@_u$ in the entry κ of its lookup table.

2.3 The Join Procedure

As for most of the CANs (see, e.g., [19]), we assume that the IP-addresses of some nodes currently in the network are (at least partially) public. Hence, we assume that a node aiming to join the network knows some contact nodes already in the network, called *entry points*. The joining procedure has mainly three stages:

1. Getting a D2B label;

2. Redistribution of the keys;

[4] Again, we focus on the construction of the CAN only, not on the way keys are assigned to resources.

3. Updating the connections (including routing, and sibling connections).

Let u be a node joining the system, and let v be its entry point in D2B, i.e., u knows the IP-address $@_v$ of v.

2.3.1 Getting a D2B label

To join, u contacts v, and gets a temporary label which is an m-bit string $s_1 \dots s_m$ chosen uniformly at random. A "request-for-join message" is sent from v through the network. The format of such a message is $\langle join, @_u, s_1 \dots s_m \rangle$, where $@_u$ is the IP-address of u. This message is routed as a lookup message, where $s_1 \dots s_m$ plays the role of the key. Hence, the join-message eventually reaches a node w, with label $x_1 \dots x_k$, and responsible for the key $s_1 \dots s_m$. I.e., $x_1 \dots x_k$ is a prefix of $s_1 \dots s_m$. If $k = m$, i.e., $x_1 \dots x_k = s_1 \dots s_m$, then the join fails, and u must choose another temporary label. Such a failure occurs with probability at most $n/2^m$, which is virtually null even for one billion nodes, for $m = 128$ or 256. Hence, assume $k < m$ (in practice k is much smaller than m). Node u gets the label $x_1 \dots x_k 1$, and w extends its label to $x_1 \dots x_k 0$. This operation is called *label extension*.

At this point, only w knows about u, and w continues to act as $x_1 \dots x_k$ until the end of the join procedure, to preserve consistency.

2.3.2 Key redistribution

The part of the lookup table stored by w that corresponds to keys which have $x_1 \dots x_k 1$ as prefix is transfered from w to u. Actually, only the keys corresponding to IP-addresses of nodes holding published resources are transfered to u. Hence, the volume of the transfer is much smaller than 2^{m-k-1} which is the range of keys managed by u. Again, to preserve consistency, node w keeps a copy of the lookup table corresponding to the transfered keys, until the end of the join procedure.

2.3.3 Updating the connections

a) Child-connections. Node u gets from w the IP-addresses of all children of w. We consider two cases, depending on whether w has a loop around it, i.e., whether or not w is labeled $00 \dots 0$ or $11 \dots 1$.

1. General case: there is a pair of indexes i, j such that $x_i \neq x_j$. We consider the two exclusive cases:
   (a) If w has a unique child labeled $x_2 \dots x_j$, $j \leq k$, then this child becomes the unique child of u, and remains child of w (see Figure 3(a)).
   (b) If w has several children, with labels of the form $x_2 \dots x_k y_1 \dots y_\ell$, $\ell \geq 1$ (see Figure 3(b)), then those satisfying $y_1 = 1$ become the children of u. They are informed by w that w is no more their parent, and must be replaced by u. Children of w with $y_1 = 0$ remain children of w.
2. Specific case: w is labeled $\alpha\alpha \dots \alpha$, $\alpha \in \{0, 1\}$. Then it has children of the form $\alpha \dots \alpha y_1 \dots y_\ell$, $\ell \geq 1$, with $y_1 = \overline{\alpha}$. By label-extension, either w or u takes label $\alpha\alpha \dots \alpha\alpha$, while the other

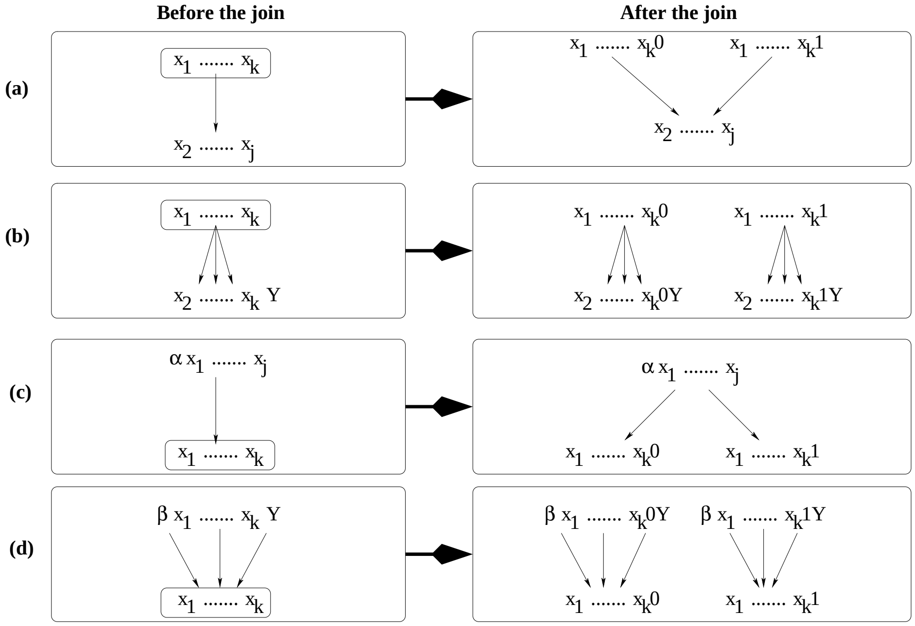

Figure 3: Updating the connections after a join

takes label $\alpha\alpha\ldots\alpha\overline{\alpha}$. Node labeled $\alpha\alpha\ldots\alpha\alpha$ takes $\alpha\alpha\ldots\alpha\overline{\alpha}$ as unique child, while node $\alpha\alpha\ldots\alpha\overline{\alpha}$ takes all nodes $\alpha\ldots\alpha y_1\ldots y_\ell=\alpha\ldots\alpha\overline{\alpha}y_2\ldots y_\ell$ as children.

b) Parent-connections. Every parent w' of w is informed by w of the existence of a new node u labeled $x_1\ldots x_k1$, of the IP-address $@_u$ of u, and of the new label $x_1\ldots x_k0$ of w.

1. General case: there is a pair of indexes i,j such that $x_i\neq x_j$. We consider the two following sub-cases:

   (a) If w' is labeled $\alpha x_1\ldots x_j$ with $j\leq k$ (see Figure 3(c)), then w' takes u as one of its child, and modifies the label of w in its routing table. Hence w' has one more child, and its degree increases by one.

   (b) If w' is labeled $\beta x_1\ldots x_k y_1\ldots y_\ell$ with $\ell\geq 1$ (see Figure 3(d)), then w' keeps w as child if $y_1=0$, or replaces w by u if $y_1=1$.

2. Specific case: w is labeled $\alpha\ldots\alpha$, $\alpha\in\{0,1\}$. Again, by label-extension, either w or u takes label $\alpha\ldots\alpha\alpha$, while the other takes label $\alpha\ldots\alpha\overline{\alpha}$. There are two exclusive sub-cases:

   (a) w has a parent of the form $\overline{\alpha}\alpha\ldots\alpha$ with $j\leq k$ α's. Then node labeled $\alpha\ldots\alpha\alpha$ takes this node as its parents, while node labeled $\alpha\ldots\alpha\overline{\alpha}$ takes both $\alpha\ldots\alpha\alpha$ and $\overline{\alpha}\alpha\ldots\alpha$ as parents.

   (b) w has parents of the form $\overline{\alpha}\alpha\ldots\alpha y_1\ldots y_\ell$, with k α's and $\ell\geq 1$. Then node labeled $\alpha\ldots\alpha\alpha$ takes those with $y_1=\alpha$ as parents, while $\alpha\ldots\alpha\overline{\alpha}$ takes those with $y_1=\overline{\alpha}$, together with node $\alpha\ldots\alpha\alpha$, as parents.

c) Sibling connections. Node u gets from w the IP-addresses of its up-sibling, which is just the former up-sibling of w. This node is informed that its down sibling is no more w but u. The new up-sibling of w is simply u, and the down-sibling of u is w.

2.4 The Leave Procedure

The leave procedure performs in three stages:

1. Finding a substitute for the leaving node;
2. Redistribution of the keys;
3. Updating the connections.

Obviously, if a node crashes, the leaving procedure may not be entirely executed, possibly it would not be executed at all. The case of a crash is in fact very different from the case of a leave, and thus will be considerer later in the text (cf. Section 4.2.1). In the current setting, we consider a node u labeled $x_1 \dots x_k$ leaving properly the system.

2.4.1 Node substitution

If a node v labeled $x_1 \dots x_{k-1}\overline{x_k}$ is in the network, then the lookup tables managed by u and v are merged and stored entirely by v, which is relabeled in $x_1 \dots x_{k-1}$. If $x_1 \dots x_{k-1}\overline{x_k}$ is not a valid label in the network, then the node-substitution procedure is slightly more complex. For instance, $x_1 \dots x_{k-1}\overline{x_k}$ may have been extended in $x_1 \dots x_{k-1}\overline{x_k}0$ and $x_1 \dots x_{k-1}\overline{x_k}1$. Possibly, one of these two latter labels (possibly both) has then been extended, and so on. Such label-extensions create a virtual binary tree rooted at $x_1 \dots x_{k-1}\overline{x_k}$, whose leaves are nodes currently in the system (see Figure 2(b)). In this tree, the children of an internal vertex $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p$ are vertices $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p0$ and $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p1$. Since the depth of this virtual binary tree is finite (it is at most $m - k$), there is at least one pair of leaves whose labels differ only at the rightmost bit-position. Let us call *critical pair* such a pair of leaves. In Figure 2(b), there is a critical pair $\{x_1 \dots x_{k-1}\overline{x_k}1010, x_1 \dots x_{k-1}\overline{x_k}1011\}$.

The sibling connections allow to find a critical pair for every node u labeled $x_1 \dots x_k$ leaving the system, as follows. If $x_k = 0$ a critical-pair message is sent to the up-sibling u' of u. This message has format $\langle \mathit{leave}, @_u \rangle$. If u' has label $x_1 \dots x_{k-1}1$, then $\{u, u'\}$ is a critical pair. Otherwise, u' forwards the message to its up-sibling u''. If the labels of u' and u'' differ only at the rightmost bit-position, then $\{u', u''\}$ is a critical pair. And so on. Since the sibling chain is bounded, a critical pair will eventually be found. In case $x_k = 1$, one proceeds the same using down-sibling connections instead of up-sibling connections.

Informally, one node of the critical pair will be the substitute for u, and the other will be the substitute for the two nodes of the critical pair. This is detailed in the next section.

2.4.2 Updating the network

There are two cases depending whether the leaving node u belongs to the identified critical pair $\{v, v'\}$.

If $u \in \{v, v'\}$, then node u' labeled $x_1 \dots x_{k-1}\overline{x_k}$ belongs to $\{v, v'\}$ as well, and becomes the substitute for u and u'. Hence, u' receives from u all information about the keys managed by u. It also receives from u all the information about the sibling, parents, and children connections of u. Node u' is relabeled in $x_1 \dots x_{k-1}$. (This operation is called *label-contraction.*) Then u' informs its

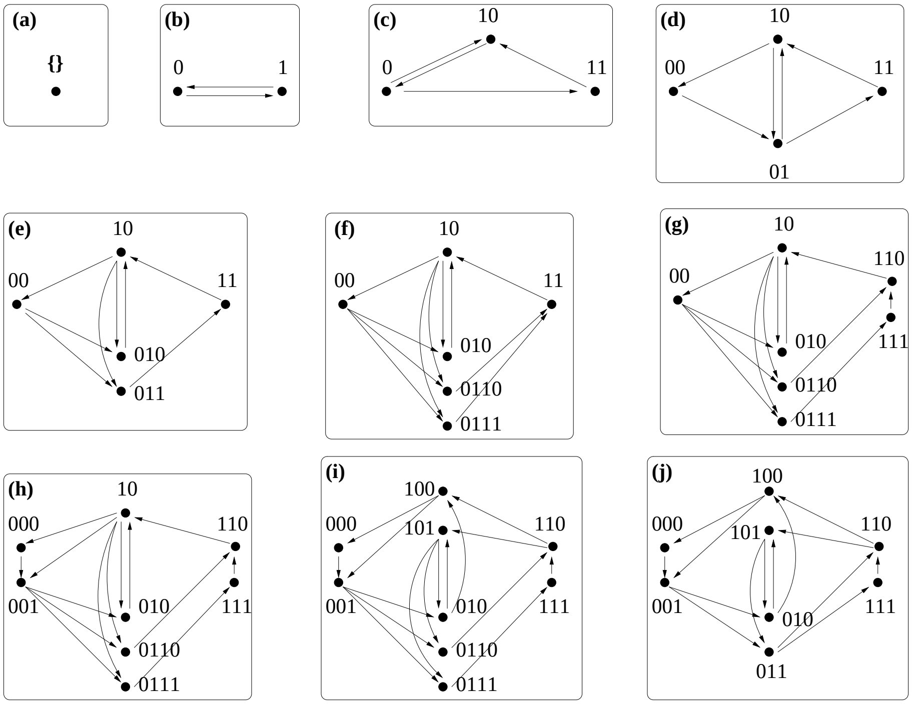

Figure 4: An example of the behavior of D2B

parents that its label was contracted, and u informs its parents that it leaves the network. Node u can then leave the network. Node u' stores in its routing tables the IP-addresses and labels of the former children of u, which now become children of u'. Finally, u' informs the former parents of u that it is now their child, with label $x_1 \dots x_{k-1}$.

If $u \notin \{v, v'\}$, then assume, w.l.o.g., that v is labeled $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p0$ and v' is labeled $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p1$. Node v' is the substitute for v and v', while node v is the substitute for u. Hence v' perform the same procedure for v and v' as u' performed for u and u' in the previous case. In particular, the label of v' is contracted into $x_1 \dots x_{k-1}\overline{x_k}y_1 \dots y_p$. Node v takes the label of u, and retrieves from u its lookup and routing tables. As soon as v has retrieved all information from u, node u leaves the system.

2.5 Example

An example of the behavior of D2B is presented on Figure 4. The first node entering the network (see (a)) takes the empty string $\varnothing$ as label. When a second node joins (see (b)), this label is extended to 0 while the new node takes label 1. Then a new node joins (see (c)). Assuming that it chooses a temporary label $1\cdots$, node labeled 1 extends its label to 10 while the new node takes label 11. A fourth node joins (see (d)). Assuming that it chooses a temporary label $0\cdots$, node 0 extends its label to 00 while the new node takes label 01. The resulting network is the graph $B(2,2)$. In (e), a new node joins with temporary label $01\cdots$. In (f), a new node joins with temporary label $011\cdots$. In (g), a new node joins with temporary label $11\cdots$. And in (h), a new node joins with temporary label $00\cdots$. Note the out-degree 5 of node 10 in (h). In (i), a new node joins with temporary label $10\cdots$. Node 100 and 101 have roughly half the degree of node 10 in (h). Finally, in (j), the node with label 0110 leaves the network, and thus node 0111 contracts its label to 011.

The resulting network is the graph $B(2,3)$, already depicted on Figure 1. The fact that steps (d) and (j) result in de Bruijn graphs is a coincidence, and in general the D2B topology is different from the topology of de Bruijn graphs. However, the "expected topology" of D2B is close to a de Bruijn graph for values of n that are close to powers of 2.

3 Main properties of D2B

This section is entirely dedicated to the proof of the following result. We prove correctness under the assumption that joins and leaves do not overlap. Nevertheless, on can relax this restrictive assumption by using the techniques presented in [11, 12].

Theorem 3.1 *The* D2B *network of key-set $\mathcal{K} = \{m\text{-bit strings}\}$ is provably correct, and satisfies the following:*

- *The expected number of keys managed by a node of an n-node* D2B *network is $|\mathcal{K}|/n$, and is, w.h.p., at most $O(|\mathcal{K}| \log n/n)$.*
- *A lookup for a key κ initiated from any node labeled $x_1 \dots x_k$ is routed correctly, and, w.h.p., reaches the node responsible for the key κ in at most $O(\log n)$ hops. With probability $1 - o(1)$, the longest route followed by a lookup message is at most $O(\log n)$ hops.*
- *At each intermediate node, the routing decision takes $O(\log \log n)$ comparisons of words on $O(\log n)$ bits. The expected congestion of any server is $O((\log n)/n)$, which is optimal among all networks of constant degree. W.h.p., the congestion of a server is at most $O((\log^2 n)/n)$.*
- *During a join or a leave, the key redistribution involves only two nodes for a join, and at most three nodes for a leave. The expected number of link modifications due to a join or a leave is $O(1)$, and is, w.h.p., at most $O(\log n)$.*

The fact that, during a join or a leave, the key-redistribution involves only two nodes for a join, and at most three nodes for a leave, is straightforward by construction. All the other properties are consequences of the following lemmas.

Lemma 3.1 *At any given time, we have :*

1. *For any $\kappa \in \{0,1\}^m$, there is a unique node in the* D2B *network whose label is a prefix of κ.*
2. *Let u be a node of* D2B *labeled $x_1 \dots x_k$, with at least two children. If there are i, j such that $x_i \neq x_j$, then the children of u are of the form $x_2 \dots x_k y_1 \dots y_\ell$, $\ell \geq 1$, and the set of sequences $y_1 \dots y_\ell$ of all the children of u is a universal prefix set. If $x_1 = \dots = x_k = \alpha$, then the children of u are of the form $x_2 \dots x_k y_1 \dots y_\ell$, $\ell \geq 2$, $y_1 = \overline{\alpha}$, and the set of sequences $y_2 \dots y_\ell$ of all the children of u is a universal prefix set.*

Proof. Initially, there is a unique node in the network, labeled by the empty string $\varnothing$. This label is the prefix of any string in $\{0,1\}^*$. Thus Property 1 holds initially. Node with label $\varnothing$ has no parent, nor child. So Property 2 holds as well. We show that these two properties are preserved after a join or a leave.

– ***The case of a join.***

Assume that the network currently satisfies properties 1, and 2, and that a new node u joins the network. Let $s_1 \dots s_m$ be the temporary label of u, and let $x_1 \dots x_k$ be the label of node v currently responsible for key $s_1 \dots s_m$. The joining node u is given label $x_1 \dots x_k 1$, while v extends its label to $x_1 \dots x_k 0$. The key-redistribution protocol described in Section 2.3.2 clearly insures that property 1 is satisfied after the join since all keys with prefix $x_1 \dots x_k 1$ are moved from v to u. For Property 2, we consider separately the children- and parent-connections.

If node v had at least two children before the join, then, by property 2, these children have labels of the form $x_2 \dots x_k y_1 \dots y_\ell$. By construction (cf. the update of the connections in Section 2.3.3), if there are two indexes $i \neq j$ such that $x_i \neq x_j$, then children with labels $x_2 \dots x_k 1 y_2 \dots y_\ell$ become children of u, while children with labels of the form $x_2 \dots x_k 0 y_2 \dots y_\ell$ remain children of v. Since the initial set of sequences $y_1 \dots y_\ell$ is a universal prefix set, the same holds for the two sets of sequences $y_2 \dots y_\ell$ corresponding to u and v. (Note that these sequences may be empty, but an empty string is a universal prefix set, and anyway the lemma considers only nodes with at least two children.) Therefore, property 2 remains satisfied for both u and v. If $x_1 = \dots = x_k = \alpha$, then node $x_1 \dots x_k \alpha$ has a unique child, and node $x_1 \dots x_k \overline{\alpha}$ has children all the initial children of v. By property 2, these children were labeled $\alpha \dots \alpha y_1 \dots y_\ell$ with $y_1 = \overline{\alpha}$, $\ell \geq 2$, and the set of sequences $y_2 \dots y_\ell$ is a universal prefix set. Therefore, Property 2 remains satisfied for u and v.

If node v had a parent with label of the form $\alpha x_1 \dots x_j$ before the join, then, after the join, this parent has replaced its child $x_1 \dots x_k$ by two children labeled $x_1 \dots x_k 0$ and $x_1 \dots x_k 1$, and therefore property 2 holds. If node v had parents with label of the form $\beta x_1 \dots x_k y_1 \dots y_\ell$ before the join, then, after the join, parents of the form $\beta x_1 \dots x_k 0 y_2 \dots y_\ell$ have $x_1 \dots x_k 0$ as unique child, and those of the form $\beta x_1 \dots x_k 1 y_2 \dots y_\ell$ have $x_1 \dots x_k 1$ as unique child. Therefore property 2 holds after the join.

– ***The case of a leave.***

Assume now that the network satisfies properties 1, and 2, and that node u labeled $x_1 \dots x_k$ leaves the network. From the description of the procedure in Section 2.4.2, we assume first, for the sake of simplicity, that u belongs to the critical pair, i.e., there is a node v labeled $x_1 \dots x_{k-1} \overline{x_k}$ currently in the network. By construction, node v relabels itself in $x_1 \dots x_{k-1}$, and takes care of all keys previously managed by u. Hence, property 1 remains satisfied after the leave.

If $x_1 \dots x_k$ had a unique child $x_2 \dots x_j$, $j < k$, before the leave, then $x_1 \dots x_{k-1} \overline{x_k}$ had also $x_2 \dots x_j$ as unique child. After the leave, node $x_2 \dots x_j$ becomes the unique child of $x_1 \dots x_{k-1}$. Hence, property 2 remains satisfied after the leave. If $x_1 \dots x_k$ had a unique child $x_2 \dots x_k$, before the leave, and $x_1 \dots x_{k-1} \overline{x_k}$ had children with labels of the form $x_2 \dots x_{k-1} \overline{x_k} y_1 \dots y_\ell$ before the leave, where the sequences $y_1 \dots y_\ell$ form a universal prefix set, then, after the leave, $x_1 \dots x_{k-1}$ has children $x_2 \dots x_k$ and all the $x_2 \dots x_{k-1} \overline{x_k} y_1 \dots y_\ell$. Hence, property 2 remains satisfied after the leave since $\{x_k\} \cup \{\overline{x_k} y_1 \dots y_\ell\}$ is a universal prefix set. Finally, if node $x_1 \dots x_k$ had children with labels $x_2 \dots x_k y_1 \dots y_p$ before the leave, while $x_1 \dots x_{k-1} \overline{x_k}$ had children with labels $x_2 \dots x_{k-1} \overline{x_k} z_1 \dots z_q$, then, after the leave node labeled $x_1 \dots x_{k-1}$ has children nodes labeled $x_2 \dots x_k y_1 \dots y_p$ and $x_2 \dots x_{k-1} \overline{x_k} z_1 \dots z_q$. Property 2 remains satisfied after the leave since both the $y_1 \dots y_p$'s and the $z_1 \dots z_q$'s are universal prefix sets.

Parents of $x_1 \dots x_k$ and $x_1 \dots x_{k-1} \overline{x_k}$ of the form $\alpha x_1 \dots x_k y_1 \dots y_p$ and $\beta x_1 \dots x_{k-1} \overline{x_k} z_1 \dots z_q$, respectively, have $x_1 \dots x_{k-1}$ as unique child after the leave. Hence property 2 is satisfied. A parent of $x_1 \dots x_k$ and $x_1 \dots x_{k-1} \overline{x_k}$ with label of the form $\alpha x_1 \dots x_j$, $j < k$, has child $x_1 \dots x_{k-1}$ after the leaves. Therefore, if $\alpha x_1 \dots x_j$ satisfied property 2 before the leaves, it satisfies it after the leaves as well. ■

Lemma 3.2 *A lookup initiated by any node labeled $x_1 \dots x_k$ reaches its destination in at most k hops.*

Proof. Let us consider a node u labeled $x_1 \dots x_k$ looking for key $\kappa = \kappa_1 \dots \kappa_m$. The lookup is performed by routing $\kappa_1 \dots \kappa_m$ toward the node v responsible for that key.

Claim. The label of any node $w \neq v$ along the route from u to v is of the form $x_i \dots x_j S$ where $j \geq i$, S is a binary string, possibly empty, and the length of the longest binary string that is a suffix of $x_i \dots x_j S$ and a prefix of κ is of length at least $|S|$.

At u, $i = 1$, $j = k$, and $S = \varnothing$, so the claim holds for the first node of the route. Let $x_i \dots x_j s_1 \dots s_\sigma$ be the label of the current node $w \neq v$, and assume that the length of the longest binary string T that is a suffix of $x_i \dots x_j s_1 \dots s_\sigma$ and a prefix of κ is of length at least σ. Assume first that w is not labeled $\alpha\alpha \dots \alpha$. If w has more than one child, then from Lemma 3.1, there is a child w' labeled $L = x_{i+1} \dots x_j s_1 \dots s_\sigma y_1 \dots y_\ell$ such that $Ty_1 \dots y_\ell$ is a binary string that is a suffix of L and a prefix of κ. From the choice of $y_1 \dots y_\ell$ in the routing protocol, the next node on the route from u to v is the node w' labeled L. This label is of the form $x_{i'} \dots x_j S'$ and satisfies the property of the claim. If w has a unique child, then it is either of the form $x_{i+1} \dots x_{j'}$ where $i + 1 \leq j' \leq j$, or of the form $x_i \dots x_j s_1 \dots s_{\sigma'}$ where $1 \leq \sigma' \leq \sigma$. In both cases, the label of the child satisfies the hypothesis of the claim. The case where w is labeled $\alpha\alpha \dots \alpha$ (which can actually occur only for $w = u$) is treated similarly, again by application of Lemma 3.1. This completes the proof of the claim.

From the claim, if $x_i \dots x_j S$ is the label of the current node along the route from u to v, then the label of next node is of the form $x_{i+1} \dots x_{j'} S'$, where S and S' satisfy the hypotheses of the claim. Therefore, either, after $i - 1$ hops from node labeled $x_1 \dots x_k$, one reaches a node labeled $x_i \dots x_j S$ where $x_i \dots x_j S$ is a prefix of κ, or, after $i - 1$ hops, one reaches a node labeled $x_i S$ where S is a prefix of κ. In the former case, we are done. In the latter, the next node of the route is the destination. Thus, in both cases, one reaches the node v responsible for the key κ, and the number of hops along the route from u to v is at most $(i - 1) + 1 \leq k$. ■

Lemma 3.3 *Assume that nodes joins and leaves at random. Then, w.h.p., the label $x_1 \dots x_k$ of any node of an n-node* D2B *network satisfies $\log n - \log\log n - O(1) \leq k \leq O(\log n)$. Also, with probability $1 - o(1)$, the longest label $x_1 \dots x_k$ satisfies $k = O(\log n)$.*

Proof. Let us consider a node u with label $x_1 \dots x_k$ in D2B. Since nodes independently join and leave at random, the set of labels in an n-node D2B network are those that would be obtained by choosing n integers independently and uniformly at random in $[0, 2^m)$. Let I be an interval of $[0, 2^m)$ starting at $val(x)$, and containing $c2^m \log n/n$ integers, for any constant $c > 3$. The probability that an integer is chosen in I is $c \log n/n$. Let X be the random variable counting the number of integers chosen in I. From Chernoff bound[5], $\text{Prob}(|X - c\log n| > \sqrt{3c}\log n) < 2/n$. Therefore, w.h.p., at least one integer is chosen in I, and thus u is responsible for less than $c2^m \log n/n$ keys. Hence, since a node responsible for at most 2^q keys has a label on at least $m - q$ bits, we have $k \geq \log n - \log\log n - \log c$. Also, w.h.p., no more than $O(\log n)$ integers are chosen in I, and thus u is responsible for at least $\frac{|I|}{2^{O(\log n)}} = \frac{c2^m \log n}{n 2^{O(\log n)}}$ keys. Hence, $k \leq O(\log n)$.

[5] Recall that the so-called Chernoff bound says that, given N pairwise independent Bernouilli variables $X_1, \dots, X_N$ of same parameter $p > 0$, $\text{Prob}(|\sum_i X_i - Np| > k) < 2e^{-k^2/3Np}$, for any positive $k \leq Np$.

Let us split $[0,2^m)$ into $\Theta(n/\log n)$ intervals of size $2^m \log n/n$, and consider n integers independently and uniformly chosen at random in $[0,2^m)$. We apply the following result by Raab and Steger [18], on the "balls into bins" game. Assume that we throw n balls independently and uniformly at random into b bins, where $n = c\, b \log b$ for some constant c. Let X be the random variable counting the maximum number of balls in any bin. Then $\text{Prob}(X > d \log n) = o(1)$ where d is a constant depending on c. Applying directly this result to our setting yields that the probability that the maximum number of integers chosen in any interval exceeds $O(\log n)$ is $o(1)$. Therefore, with probability $1 - o(1)$, the minimum number of keys managed by any node of a D2B network is at least $2^m \log n/n2^{O(\log n)}$, and thus the maximum length of all labels is at most $O(\log n)$. ■

The following is a direct consequence of Lemma 3.3.

Corollary 3.1 *The number of keys managed by a node of an n-node* D2B *network is, w.h.p., at most $O(2^m \log n/n)$.*

The following is a direct consequence of Lemmas 3.2 and 3.3.

Corollary 3.2 *The number of hops experienced by a lookup to reach its destination in an n-node* D2B *network is, w.h.p., at most $O(\log n)$.*

Lemma 3.4 *The expected number of link-modifications due to a join or a leave is constant, and is, w.h.p., at most $O(\log n)$.*

Proof. Let $x_1 \ldots x_k$ be the label of a node u. If u has more than a single child, then these children are labeled with string of the form $x_2 \ldots x_k y_1 \ldots y_\ell$ where $\ell \geq 1$. The range of keys covered by the children of u goes from $x_2 \ldots x_k 0 \ldots 0$ with $m-k+1$ zeros, to $x_2 \ldots x_k 1 \ldots 1$ with $m-k+1$ ones. From Lemma 3.3, w.h.p., $k \geq \log n - \log\log n - O(1)$. Therefore the number of keys managed by all children of u together is at most $2^{m-\log n+\log\log n+O(1)} = O(2^m \log n/n)$. From Chernoff bound, this range of keys is, w.h.p., covered by at most $O(\log n)$ nodes. Therefore, the out-degree of u is, w.h.p., $O(\log n)$. The same argument applies for the in-degree of node u by considering separately parents of the form $0x_1 \ldots x_k y_1 \ldots y_\ell$, and those of the form $1x_1 \ldots x_k y_1 \ldots y_\ell$. Hence the degree of u is, w.h.p., $O(\log n)$. ■

Lemma 3.5 *The expected congestion of a server is $O((\log n)/n)$, and is, w.h.p., $O((\log^2 n)/n)$.*

Proof. Let u be any node currently in D2B, and let $x_1 \ldots x_k$ be its label. We compute an upper bound on the load of u, i.e., on the number of lookups that pass through u, or ends at u. The expected size of the lookup table stored by u is $O(2^m/n)$. Therefore, since there are $n-1$ possible sources, the expected load induced by lookups for keys stored at u is $O(2^m)$. The lookups which traverse u have a specific format. For a source labeled $y_1 \ldots y_\ell x_1 \ldots x_i$, $i \geq 1$, the requested keys must be of the form $x_j \ldots x_k \kappa_1 \ldots \kappa_{m-k+j-1}$ where $j \leq i+1$. The expected number of nodes with a label terminated by the sequence $x_1 \ldots x_i$ is $n/2^i$. The number of keys of the form $x_j \ldots x_k \kappa_1 \ldots \kappa_{m-k+j-1}$ with $j \leq i+1$ is at most 2^{m-k+i}, and thus in average at most $2^{m+i}/n$. Therefore, for a given i, the expected contribution to the load is at most 2^m. Since there are $O(\log n)$ possible values for i, the expected total load is $O(2^m \log n)$, and thus the expected congestion is at most $O(\log n/n)$.

Now, from Lemma 3.3, w.h.p., $k \geq \log n - \log\log n - O(1)$. Therefore, the size of the lookup table stored by u is, w.h.p., at most $O(2^m \log n/n)$. Let $i_0 = \log n - \log\log n$, and let $i \leq i_0$. Applying Chernoff bound, the number of nodes whose labels are terminated by the sequence $x_1 \dots x_i$ is at most $O(n/2^i)$ with probability at least $1 - O(\frac{1}{n \log n})$. Therefore, the contribution of such nodes to the load of u is at most $O(2^{m-k+i} n/2^i) \leq O(2^m \log n)$ with probability at least $1 - O(\frac{1}{n \log n})$. Therefore, nodes with label containing a sequence $x_1 \dots x_i$ as suffix for some $i \leq i_0$ contribute of $O(2^m \log^2 n)$ to the load of u, with high probability.

Let $i > i_0$, and let us compute the contribution to the load of nodes with labels containing a sequence $x_1 \dots x_i$ as suffix. By Chernoff bound, there are, w.h.p., at most $O(\log n)$ nodes with labels terminated by the sequence $x_{i-i_0} \dots x_i$, and therefore at most $O(\log n)$ nodes with labels terminated by the sequence $x_1 \dots x_i$. The contribution of these nodes to the load is at most $O((\log n) 2^{m-k+i})$. Summing up these contributions for all i's, $i_0 < i \leq k$, the resulting contribution to the load is $O(2^m \log n)$.

Therefore, the total load of u is, w.h.p., $O(2^m \log^2 n)$. Thus its congestion is, w.h.p., $O((\log^2 n)/n)$. ■

Remark. An expected congestion of $O((\log n)/n)$ is optimal for an n-node network of constant degree with $|\mathcal{K}|/n$ keys per node. Indeed, let us consider a directed graph with maximum in- and out-degree Δ. The number of nodes at distance $\leq d$ from any node u is at most $\sum_{i=0}^{d} \Delta^i$. Therefore, there are at most $O(\sqrt{n}/\Delta)$ nodes at distance $\leq \frac{1}{2} \log_\Delta n$, and thus there are $\Theta(n)$ nodes at distance $\Omega(\log n)$. Therefore, each node contributes of $\Omega(|\mathcal{K}| \log n)$ to the load, resulting in a global load of $\Omega(n|\mathcal{K}| \log n)$. To have $n - o(n)$ nodes with load $O(|\mathcal{K}| \log n)$, the global load must be balanced among nodes. Thus $n - o(n)$ nodes have load $\Omega(|\mathcal{K}| \log n)$, and thus a congestion $\Omega((\log n)/n)$.

4 Variants of the Construction

In this section, we present several variants of D2B, including the d-dimensional version of the network, an attempt to match the logical network to the physical one, a discussion about the robustness of D2B, and a simple strategy to decrease the degree of the nodes.

4.1 The d-dimensional D2B network

The d-dimensional D2B, $d \geq 2$, uses the set of keys $\mathcal{K} = \{0, \dots, d^m\}$, i.e., the set of words of length m on an alphabet of d letters $0, 1, \dots, d-1$. The underlying topology of D2B is $B(d, k)$. More precisely, a node of the d-dimensional D2B is labeled by a pair $\langle x_1 \dots x_k, [a, b] \rangle$ where $x_i \in \{0, \dots, d-1\}$, and $0 \leq a \leq b \leq d-1$.

Node labeled $\langle x_1 \dots x_k, [a, b] \rangle$ is responsible for the key $\kappa \in \{0, \dots, d-1\}^m$ if and only if $x_1 \dots x_k \alpha$ is a prefix of κ for some $\alpha \in [a, b]$. A universal prefix property, defined similarly to the case $d = 2$, insures that all keys are assigned. During a join, if the temporary label of a node u is managed by node w of label $\langle x_1 \dots x_k, [a, b] \rangle$, then v extends its label in the following way. If $a < b$, then v changes its label to $\langle x_1 \dots x_k, [a, a + \lfloor \frac{b-a}{2} \rfloor] \rangle$ while u takes label $\langle x_1 \dots x_k, [a + \lfloor \frac{b-a}{2} \rfloor + 1, b] \rangle$. If $a = b$, then v changes its label to $\langle x_1 \dots x_k a, [0, \lfloor \frac{d-1}{2} \rfloor] \rangle$ while u takes label $\langle x_1 \dots x_k a, [\lfloor \frac{d-1}{2} \rfloor + 1, d-1] \rangle$.

The children of node $\langle x_1 \dots x_k, [a, b] \rangle$ are either of the form $\langle x_2 \dots x_j, [\alpha, \beta] \rangle$, $j \leq k$, or of the form $\langle x_2 \dots x_k y_1 \dots y_\ell, [\alpha, \beta] \rangle$, $\ell \geq 1$. Routing performs as in the 2-dimensional case, by looking for the longest prefix of the requested key among the suffixes of the labels of the children. The sibling connections are defined in a way similar to the 2-dimensional case, and the leave procedure also performs the same by looking for a critical pair among the sibling nodes.

One can easily check that the expected length k of a label $\langle x_1 \dots x_k, [a,b] \rangle$ is $O(\log_d n)$, yielding an expected degree of $O(d)$ and a diameter $O(\log_d n)$. Hence, the d-dimensional D2B allows a trade-off between the lookup latency, and the time required to update the connections after a join or a leave. It also provides a network more robust against processor crash, as discussed in the next section.

4.2 Improving the performances of D2B

4.2.1 Robustness

A peer-to-peer system must be able to support a certain number of processor crashes, and the brute disconnection of users not respecting the leave procedure. As in most of the CANs proposed in the literature, nodes of D2B must control each other by periodical exchanges of pings between neighbors. When the failure of a node is detected, its neighbors act as for a leave of this node. The local lookup table of the faulty node is however lost. Nodes republish their keys periodically so that lost lookup tables can be reconstructed (see [9] for more detail). If a node loses all its neighbors, i.e., if all neighbors of a node quit brutally the network (without executing the leave procedure), then this node must recontact an entry point (one of those nodes in the network whose addresses are public), and simulate the leave procedure executed by its neighbors. To avoid an excessive use of the entry points, it is desirable that the disconnection of nodes be unlikely. Hence, it is desirable that the degree of a node be sufficiently large. The d-dimensional D2B has expected degree $\Theta(d)$. Hence, one can take d large enough so that a node has little chance to lose all its neighbors simultaneously. If one prefers to use the 2-dimensional D2B (say, for a sake of simplicity), an appropriate solution consists to systematically connect every node $x_1 \dots x_k$ to at least $\log n$ descendants of the form $x_i \dots x_k y_1 \dots y_\ell$, for $i \geq 1$. As a side effect, this solution provides shorter routes to the lookup messages.

4.2.2 Optimized choice of node label

The maximum degree of D2B is determined by a “balls into bins” game. Given an interval I of $[0, 2^m)$ of length $2^m \log n/n$, we have seen that the Chernoff bound insures that at most $O(\log n)$ nodes have values in I, w.h.p., and hence the degree of any given node is $O(\log n)$, w.h.p. Using the result in [18], we have seen that the maximum, taken over all intervals I, of the number of nodes having values in I, is $O(\log n)$, with probability $1 - o(1)$. This follows from the fact that throwing n balls at random into b bins, with $n \simeq b \log b$, results in a maximum number of balls in any bin of $O(\log n)$ with probability $1 - o(1)$.

Now, in their seminal paper, Azar *et al.* [1] considered the following process: balls are thrown one by one; d bins are selected at random for each ball; the ball chooses the bin containing currently the least number of balls among the d selected bins. It is shown in [1] that, as n goes to infinity, the number of balls in the fullest box is $\Theta(n/b + \ln \ln n / \ln d)$, with probability $1 - o(1)$. Hence the deviation to the mean is exponentially less than if no choice is given to the balls, even for $d = 2$. This suggests to give a choice among $d \geq 2$ different labels for each node that joins the network. Each joining node u chooses d temporary labels. For each temporary label L, u computes how many keys would be assigned to it if choosing L as label. Node u chooses the label that maximizes the number of keys that will be under its responsibility. In this way, one expects the keys to be better balanced among nodes.

4.2.3 Matching with the physical network

A CAN is a logical network. In particular, connections between neighbors may not respect the locality constraints of the physical network (geographical, technological, etc.). Tapestry and some other CANs offer several alternative routes between any two nodes, and routing aims to select the best one (whose quality is often estimated as the round-trip time of a ping). However, the setting of the network itself is not optimized, and the routes are computed *a posteriori*. If a joining node selects several temporary labels, it may select the "best" labels among them. The selection could be performed according to the IP-address of the neighbors, giving a preference to the label with neighbors that are close physically in Internet. An alternative choice could be based, as for Tapestry, on pings addressed to the neighbors.

5 Concluding Remarks and Future Works

We have presented D2B, a new content-addressable network for peer-to-peer systems. Beside its simplicity (comparable to the one of, say, Chord), and its scalability, the main characteristics of D2B are: constant expected update time ($O(\log n)$, w.h.p.), any lookup reaches its destination in $O(\log n)$ hops, w.h.p., and the expected congestion of a node is $O((\log n)/n)$ $(O((\log^2 n)/n)$, w.h.p.). These nice features suggest that D2B is a very promising candidate for the construction of distributed lookup tables. A project recently starts at LRI, aiming to merge the concepts of global computing and peer-to-peer systems (see [25]). The integration of D2B as a basic support for the global and P2P computing platform developed within this project is currently under discussion.

References

[1] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[2] L. Barrière, P. Fraigniaud, E. Kranakis, and D. Krizanc. Efficient routing in networks with long range contacts. In *15th Int. Symp. on Distributed Computing (DISC)*, volume 2180 of *LNCS*, pages 270–284, 2001.

[3] L. Barrière, P. Fraigniaud, L. Narayanan, and J. Opatrny. Dynamic construction of bluetooth scatternets of fixed degree and low diameter. In *14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2003.

[4] J.-C. Bermond and P. Fraigniaud. Broadcasting and gossiping in de Bruijn networks. *SIAM Journal on Computing*, 23(1):212–225, 1994.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, pages 46–66. Springer, 2001. See also http://freenetproject.org.

[6] M. Datar. Butterflies and peer-to-peer networks. In *European Symp. on Algorithms*, number 2461 in LNCS, page 310. Springer, 2002.

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

[8] A. Fiat and J. Saia. Censorship resistant peer-to-peer content-addressable networks. In *ACM/SIAM Symp. on Discrete Algorithms (SODA)*, pages 94–103, 2002.

[9] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao. Distributed object location in a dynamic network. In *14th ACM Symp. on Parallel Algorithms and Architecture (SPAA)*, 2002.

[10] J. Kleinberg. The small-world phenomenon: an algorithmic perspective. In *32nd ACM Symp. on Theory of Computing (STOC)*, 2000.

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

[12] N. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in distributed hash tables. In *Peer-to-Peer Systems*, volume 2429 of *LNCS Hot series*, page 295. Springer-Verlag, 2002.

[13] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic lookup network. In *21st ACM Symp. on Principles of Distributed Computing (PODC)*, 2002.

[14] B. McKay, M. Miller, and J. Siran. A note on large graphs of diameter two and given maximum degree. *J. Combin. Theory B*, 74:110–118, 1998.

[15] P. Mockapetris and K. Dunlap. Development of the domain name system. In *ACM SIGCOMM*, pages 123–133, 1988.

[16] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *42th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2001.

[17] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *9th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.

[18] M. Raab and A. Steger. Balls into bins – a simple and tight analysis. In *2nd Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, volume 1518 of *LNCS*. Springer, 1998.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, pages 161–172, 2001.

[20] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Int. Conf. on Peer-to-Peer Computing (P2P)*, 2001.

[21] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *18th IFIP/ACM Int. Conf. on Dystributed Systems Platforms (Middleware)*, 2001.

[22] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[23] D. Watts and S. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998.

[24] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

[25] http://www.lri.fr/archi/parall/introduction.en.html.