Self-stabilizing De Bruijn Networks

Andréa Richa[1]*, Christian Scheideler[2]**, Phillip Stevens[1]
[1] Computer Science and Engineering, SCIDSE, Arizona State University
Tempe, AZ 85287, USA; {aricha,pcsteven}@asu.edu
[2] Department of Computer Science, University of Paderborn, D-33102 Paderborn, Germany; scheideler@upb.de

Abstract. This paper presents a dynamic overlay network based on the De Bruijn graph which we call *Linearized De Bruijn (LDB) network*. The LDB network has the advantage that it has a guaranteed constant node degree and that the routing between any two nodes takes at most $O(\log n)$ hops with high probability. Also, we show that there is a simple local-control algorithm that can recover the LDB network from any network topology that is weakly connected.

1 Introduction

Peer-to-peer networks (P2P) are characterized by their lack of centralized control and scalability. While such qualities make them highly versatile, they also make it difficult to design efficient routing algorithms that do not break down under high network flux. One approach to designing a P2P network is to start with a classical family of graphs which has all the qualities desired for the P2P network. Then, one would try to design a dynamic variant of this family of graphs that is able to accommodate any number of peers and that can handle a high rate of joining and leaving peers. Such an approach is used in the design of the P2P networks presented, e.g., in [18], [21], [19], [16], and [15].

A general approach to transform classical families of graphs into dynamic P2P networks was formalized by Naor and Wieder in [16] and is called the *continuous-discrete approach*. The basic idea of this approach is to first transform a classical family of graphs into an infinite-size network in a continuous space that preserves essential properties of these graphs. The communication algorithms needed can be easily understood and designed under this space. Following this, one must simply find a way to transform the network from the continuous to the dynamic discrete domain that preserves the basic properties of the network as well as these algorithms. However, while this approach yields P2P networks that are easy to maintain and to use as long as the topology stays in the desired form, it is not clear how these networks can recover from a misconfigured topology.

In this work, we present the *Linearized De Bruijn (LDB)* network, which is based on a discretization of a continuous variant of the classical De Bruijn

* Supported in part by NSF awards CCF-0830791 and CCF-0830704
** Supported in part by DFG awards SCHE 1592/1-1 and SFB 901 (On-the-Fly Computing).

network, and which preserves the static network's $O(\log n)$ *routing time and constant node degrees*. Moreover, the LDB network is *self-stabilizing* in the sense that it can recover from any case in which the topology is still weakly connected. Recall that a directed graph G is called *weakly connected* if for any pair of nodes v, w there is a path in G from v to w when considering all edges to be undirected. Other dynamic variants of the De Bruijn network have been proposed in the literature before (e.g., [16]), but none of them is self-stabilizing.

This paper is organized as follows. In Section 2, we present the related work in the literature and Section 3 defines the structure of the LDB network. We then present our routing algorithm in Section 4 and prove its logarithmic bound. Section 5 presents the self-stabilization results for the LDB while Section 6 describes and analyzes join and leave operations. Section 7 concludes the paper and also presents some lines for future work.

2 Related Work

Our work expands on that of Naor and Wieder in [16]. Both our construction and their distance-halving network are based on a continuous extension of the De Bruijn graph to the unit interval. However, our self-stabilizing construction provides additional fault-tolerance while maintaining constant node degree. Naor and Wieder's first construction has constant average degree w.h.p. but does not offer a way to recover from faults. They also suggest a construction which offers stronger results with regards to fault tolerance, but requires an increase in the average degree to $\Theta(\log n)$. The De Bruijn network has also been used as the basis for several other peer-to-peer networks, including [1], [7], [11], and [14]. These constructions also achieve logarithmic routing and constant average degree w.h.p., but none of them is self-stabilizing.

Various self-stabilizing overlay networks have been designed in recent years. Cramer and Fuhrmann [5] present a self-stabilizing ring network and Caron et al. [3] describe a Snap-Stabilizing Prefix Tree for Peer-to-Peer systems. Onus et al. [17] present a linearization technique to transform an arbitrary connected network into a sorted list. Our paper is based on this technique and shows how to extend it to dynamic De Bruijn networks. Clouser et al. [4] propose another variant of the linearization technique to construct a deterministic self-stabilizing skip list. Jacob et al. [10] generalize the linearization technique to two dimensions and present a self-stabilizing construction for Delaunay graphs. In another paper, Jacob et al. [9] present a self-stabilizing variant of the skip graph and show that it can recover its network topology from any weakly connected state in $O(\log^2 n)$ communication rounds with high probability. Gall et al. [8] discuss models that capture the parallel time complexity of locally self-stabilizing networks that avoids bottlenecks and contention. Recently, Kniesburges et al. [13] showed how to obtain a self-stabilizing Chord network. In [2] the authors present a general framework for the self-stabilizing construction of any overlay network. However, the algorithm requires the knowledge of the 2-hop neighborhood for each node and involves the construction of a clique. In that way, failures at the structure of the overlay network can easily be detected and repaired. All of the constructions above that result in networks of low diameter and high expansion (such as the skip graphs) result in a logarithmic degree while our network guarantees a constant degree.

3 The Linearized De Bruijn Network

The d-dimensional De Bruijn graph is an undirected graph $G = (V, E)$ with node set $V = \{0, 1\}^d$ and an edge set E in which every node with label $(x_1, \ldots, x_d) \in \{0, 1\}^d$ is connected to the nodes $(0, x_1, \ldots, x_{d-1})$ and $(1, x_1, \ldots, x_{d-1})$. When letting $d \to \infty$ and interpreting every label $(x_1, \ldots, x_d)$ as $x = \sum_{i=1}^{d} x_i/2^i$, the node set converges to $U = [0, 1)$ and the edge set to the family $F = \{f_0, f_1\}$ of functions

$$f_0(x) = x/2 \quad \text{and} \quad f_1(x) = (1 + x)/2 \ .$$

Thus, (U, F) represents a continuous form of the De Bruijn graph. The question is how to transform this continuous form into a dynamic discrete form for any number of peers. We propose the following form.

Definition 1. *The* Linearized De Bruijn network *(LDB) $G = (V, E)$ is a directed graph where the node set V can be partitioned into the set of* real *nodes V_R and a set of* virtual *nodes V_V. Each real node $v \in V_R$ has a real-valued label*[1] *in the interval $(0, 1)$; in addition, each $v \in V_R$ hosts two virtual nodes in V_V: a* left *virtual node, $l(v)$, with label $\frac{v}{2}$ and a* right *virtual node, $r(v)$, with label $\frac{v+1}{2}$. The collection of all real and virtual nodes $v \in V$ is arranged in sorted order of their labels, and $(v, w) \in E$ if and only if v and w are consecutive in the linear ordering* (linear edges) *or w is a virtual node of v* (virtual edges)*.*

The definition above reflects the ideal, stable state of the LDB, and our goal is to provide a self-stabilizing mechanism to get to that state from any weakly connected graph. Note that the virtual edges between a real node v and its virtual nodes $l(v)$ and $r(v)$ are static throughout the self-stabilization process as all these nodes are hosted by v, so v can maintain them directly. These virtual edges actually constitute the edges of the continuous De Bruijn construction. In addition to that, v may have a collection of non-virtual edges that start at v, $l(v)$ or $r(v)$ and that eventually are to be transformed into the linear edges in the definition. As we will see, the combination of virtual and linear edges will allow De Bruijn-like routing in the LDB.

In the following, we will say that a node v is to the *right* (resp., *left*) of a node w whenever the label of v is greater (resp., smaller) than the label of w. Given a (real or virtual) node v, we define $N(v)$ as the *neighborhood* of v, which consists of all (real and virtual) nodes that can be reached via non-virtual edges from v. In other words, $N(v)$ represents the current knowledge of v of the other nodes in the network. We define $\mathit{pred}(v) = \max\{w \in N(v) \mid w < v\}$

[1] We may indistinctly use v to denote a node or its label, when clear from the context.

and $succ(v) = \min\{w \in N(v) \mid w > v\}$. In the ideal state, $pred(v)$ and $succ(v)$ represent the linear edges of v, while in the non-ideal case, $pred(v)$ and $succ(v)$ may just be candidates for v's linear edges.

Analogous to the consistent hashing approach [12], we assume that the real nodes are assigned to points in $(0, 1)$ in a pseudorandom manner, so we can assume that the real nodes are distributed uniformly at random over the interval $(0, 1)$. Note that in the LDB network every real node has a degree of at most 8 (two linear edges for each of v, $l(v)$ and $r(v)$, all hosted by node v, plus the two virtual edges $(v, l(v))$ and $(v, r(v))$). As the LDB network organizes the nodes in a sorted list (in the ideal state), it may be used similar to [12] to construct a distributed hash table.

4 Routing Algorithm

In this section, we outline the algorithm we will use when routing from a node v to a destination node w in the LDB network. In order to implement our algorithm, we must first get a good estimate for $c \log n/n$ (in this paper, all logarithms are to the base 2) for a constant c. For this we use the following lemma adapted from [20]. Recall that the n real nodes are distributed uniformly and independently at random over $(0, 1)$.

Lemma 1. *Let $I(j) \subseteq (0, 1)$ be any interval of size $(1/2)^j$ starting at a real node and let $N(j)$ be the number of real nodes in $I(j)$. For any constant $c > 1$ there is a constant $\epsilon \in (0, 1)$ (that can be arbitrarily small depending on c) so that w.h.p.*[2] *it holds: if $|I(j)| < (1 - \epsilon)(c \log n)/n$ then $j > N(j)/c - \log N(j)$ and if $|I(j)| > (1 + \epsilon)(c \log n)/n$ then $j < N(j)/c - \log N(j)$.*

Proof. Suppose that $|I(j)| < (1 - \epsilon)c \log n/n$ for some constant $\epsilon > 0$. For $\delta > 0$ with $(1-\delta)^2 = (1-\epsilon)$ it follows from the Chernoff bounds that $N(j) = \alpha c \log n$ for some $\alpha \leq 1-\delta$ w.h.p. given that c is large enough compared to δ (resp. ϵ). Also, $N(j) \geq 1$ as $I(j)$ starts at a real node, so $\alpha \geq 1/(c \log n)$. As $(1 - \delta)^2 c \log n/n < \alpha c \log n/n^\alpha$ for any $1/(c \log n) \leq \alpha \leq 1-\delta$ if n is sufficiently large, it follows that in this case $|I(j)| < N(j)/2^{N(j)/c}$ and therefore $j > N(j)/c - \log N(j)$. Thus, it holds that as long as $|I(j)| < (1 - \epsilon)(c \log n/n)$, $j > N(j)/c - \log N(j)$ w.h.p. The other side is shown in a similar way. □

The lemma allows us to accurately estimate $c \log n/n$ which will be helpful for defining appropriate intervals to identify shortcuts in the routing: starting with a real node v, go to the right until a point x is reached so that for the interval $I(j) = [v, x)$ it holds that $j < N(j)/c - \log N(j)$ for the first time. According to the lemma, it holds for this interval that $|I(j)|$ is within $(1 - \epsilon)(c \log n)/n$ and $(1 + \epsilon)(c \log n)/n$ w.h.p. Similar bounds also hold if $I(j)$ is required to end at a real node as the distance between two consecutive real nodes in $(0, 1)$ can

[2] With high probability, that is, with probability at least $1 - 1/n^c$, for some constant $c > 0$.

be shown to be at most $(\epsilon c \log n)/n$ w.h.p. (given that c is sufficiently large compared to ϵ) so that the deviation from $(c \log n)/n$ increases to a factor of at most $(1 \pm 2\epsilon)$. The lemma also allows us to accurately estimate $\log n$: let $I(j)$ be defined as above. Then j is within $\log n - \log\log n - \log c - \log(1 \pm \epsilon)$ and hence $j + \log j + \log c$ is equal to $\log n$ up to some small additive constant independent of c w.h.p.

We can interpret the nodes $x_0, x_1, \ldots, x_k$ in the definition below as the first k nodes one would have followed when routing on the continuous De Bruijn network from node v_0 to a destination given by a node whose highest order k bits are $b_{k-1}, \ldots, b_0$.

Definition 2. *Let b_0, ..., b_{k-1} be a sequence of bits and $v_0 \in (0,1)$. We define a* sequence of ideal De Bruijn hops *x_0, ..., x_k recursively by $x_0 = v_0$ and $x_{i+1} = x_i/2$ if $b_i = 0$ and $x_{i+1} = (1 + x_i)/2$ if $b_i = 1$.*

The routing algorithm is presented in Algorithm 1 and proceeds in three basic stages. Throughout the algorithm, v represents the node at which the message to be routed is currently located. First (Lines 1 – 12), the algorithm determines a close estimation of $\log n$ (according to Lemma 1) so that it can determine how many bits of the destination it needs to fix while emulating the classic De Bruijn routing. It also defines the intervals T_i: $T_i = [i/2^j, (i+1)/2^j)$ for some integer j with $1/2^j \in [(1-\epsilon)c \log n/n, (1+\epsilon)2c \log n/n]$. As we will show later, these intervals are chosen so that our algorithm visits each of these intervals at most once during the routing. The second stage (Lines 13 – 34) is where the network actually uses the virtual edges to emulate routing in the continuous De Bruijn network. Starting from the least significant bit chosen to be fixed and proceeding towards the most significant bit in the destination address, the network will follow the left virtual edge or the right virtual edge depending on whether the bit is a 0 or 1, respectively. It must then proceed linearly from the current virtual node to find a new real node v from which to perform a De Bruijn hop (Lines 21 – 28). If the routing process detects that a later ideal De Bruijn hop, x_k, (as in Definition 2) is in the same interval T_i as the message to be routed (Line 17; we will show that we can always find such a valid x_k, which matches the respective bits of the destination in at least one more position, in the interval T_i, w.h.p.), it may skip ahead and proceed *from the current node* v as if it has already fixed all the less significant matching bits between the destination and x_k. Finally (Lines 35 – 37), the message moves linearly from x_κ towards the destination address.

Figure 1 illustrates some of the variables and actions taken by our algorithm, where v_k is equal to node v at the start of iteration k (a virtual node unless $k = 0$), y_k (resp., $y_{k'}$) is the value of the real node v found at the end of the while loop in Lines 26 – 28 in iteration k (resp., iteration k' immediately preceding iteration k), x_k is as defined in Line 17 in iteration k, T_{i_k} is equal to the interval T_i in iteration k.

Before we show a hop bound of $O(\log n)$ for the routing, we state and prove some basic facts.

Algorithm 1 Routing in the LDB

```
1: $v = v_0$
2: $N = 1$
3: repeat
4:    $v = succ(v)$
5:    if $v$ is a real node then
6:       $N = N + 1$
7:    end if
8: until $\log(1/|v - v_0|) \leq N/c - \log N$
9: $j = \lceil \log(1/|v - v_0|) \rceil$
10: $\kappa = j + \log j + \log c$
11: In the following let $T_i = [i(1/2)^j, (i+1)(1/2)^j]$ for all $0 \leq i < 2^j$ and let $b_i$ be the
    $(\kappa - i)$th bit of the destination address
12: Fix $x_0$, ..., $x_\kappa$ recursively as in Definition 2 based on $b_0$, ..., $b_{\kappa-1}$ and initial point
    $v_0$
13: $v = v_0$
14: $k = 0$
15: while $k \neq \kappa$ do
16:    Let $i$ be such that $v \in T_i$
17:    $k = max\{k : x_k \in T_i\}$
18:    if $k = \kappa$ then
19:       Break
20:    end if
21:    if $v$ is left of the midpoint of $T_i$ then
22:       Let $next(x) = succ(x)$ for all nodes $x$
23:    else
24:       Let $next(x) = pred(x)$ for all nodes $x$
25:    end if
26:    while $v$ is a virtual node do
27:       $v = next(v)$
28:    end while
29:    if $b_k = 0$ then
30:       $v = l(v)$
31:    else if $b_k = 1$ then
32:       $v = r(v)$
33:    end if
34: end while
35: while $v \neq$ destination node do
36:    $v = pred(v)$ or $v = succ(v)$, whichever is closer to the destination.
37: end while
```

Definition 3. *Let I be some interval in $(0,1)$. We define $\Psi(I) = \{x \in (0,1) : x/2 \in I$ or $(x+1)/2 \in I\}$. If $\mathcal{I}$ is some collection of intervals in $(0,1)$ then we define $\Psi(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \Psi(I)$.*

Lemma 2. *Let I be some interval in $(0,1)$. The total length of the collection of intervals in $I \cup \Psi(I)$ is at most $3|I|$.*

Proof. Suppose $I = (i_1, i_2)$ is an interval with length l. If $I \subseteq (0, 1/2)$ or $I \subseteq (1/2, 1)$, then $\Psi(I)$ is just some interval in $(0,1)$ with length $2l$. If $1/2 \in I$, then $I = I_L \cup I_R$ where $I_L = (i_1, 1/2]$ and $I_R = [1/2, i_2)$, and $\Psi(I) = \Psi(I_L) \cup \Psi(I_R)$ which has length less than or equal to $2l$. Thus in general, if I has length l, the length of $\Psi(I) \leq 2l$ and in particular, the length of $I \cup \Psi(I)$ is at most $3l$. □

Corollary 1. *Let $\mathcal{I}$ be some collection of intervals in $(0,1)$. Then the length of the collection of intervals in $\mathcal{I} \cup \Psi(\mathcal{I})$ is at most three times the length of the collection of intervals in $\mathcal{I}$.*

The following lemma will prove useful for giving an upper bound on the length of a routing path.

Lemma 3. *Let $\mathcal{I}$ be a collection of intervals with total length at most $\frac{p \log n}{n}$ for some constant p. Then w.h.p. the number of nodes in $\mathcal{I}$ is $O(\log n)$.*

Proof. A node is in an interval in $\mathcal{I}$ if there is a real node in an interval in $D = \mathcal{I} \cup \Psi(\mathcal{I})$. So we must show that the number of real nodes in D, which has total length at most $\frac{3p \log n}{n}$, is bounded by $(1+a)\log n$, for some constant a, with high probability. Let v_1, ..., v_n be the real nodes in the network. For each v_i, define

$$X_i = \begin{cases} 1 : v_i \in D \\ 0 : v_i \notin D \end{cases}$$

Note that $E[\sum_{i=1}^{n} X_i] \leq 3p \log n$. So for $3p(1+a') = (1+a)$ we have

$$\begin{aligned} P[D \text{ has more than } (1+a)\log n \text{ real nodes}] &= P[\sum_{i=1}^{n} X_i \geq (1+a')3p \log n] \\ &\leq e^{\frac{-3a'^2 p \log n}{3}} \\ &= n^{-p'a'^2}, \end{aligned}$$

where $p' = p/\ln 2$. The inequality follows from standard Chernoff bounds. □

Now we are ready to prove that the logarithmic length of a routing path holds with high probability.

Theorem 1. *The number of edges on the routing path from a source node v to a destination node w in the LDB network G is $O(\log n)$, w.h.p.*

Proof. First we bound the number of hops performed during Lines 1 – 12. By Lemma 1 the length of the interval traversed to calculate j is $O(\log n/n)$, so from Lemma 3 it follows that the number of hops traversed in Lines 1 – 12 is $O(\log n)$.

Next we bound the number of hops performed in the middle stage of the algorithm, Lines 13 – 34. In each iteration of loop beginning on Line 15 we have a value of k determined at Line 17, an associated x_k, an interval T_{i_k} from Line 16, a node from which we begin, which we call v_k, and a real node whose virtual node we hop to at the end of the iteration (Lines 26 – 28) which we call y_k, and an interval $S_k = [v_k, y_k]$ (or $[y_k, v_k]$). Note that, by definition, $v_k \in T_{i_k}$ and x_k always exist (since we can always take $k = 0$) — we will actually show below that the sequence of indices k computed in the while loop is strictly increasing w.h.p., implying that the sequence of x_k's found by our algorithm strictly increases the number of most significant bits matched to $b_0, \ldots, b_{k-1}$. We will also show that y_k also belongs to T_{i_k} w.h.p., implying that $S_k \subseteq T_{i_k}$ w.h.p. Figure 1 illustrates $y_{k'}, v_k, y_k, x_k$ and T_{i_k}, where k' was the value of k chosen on the iteration prior to k.

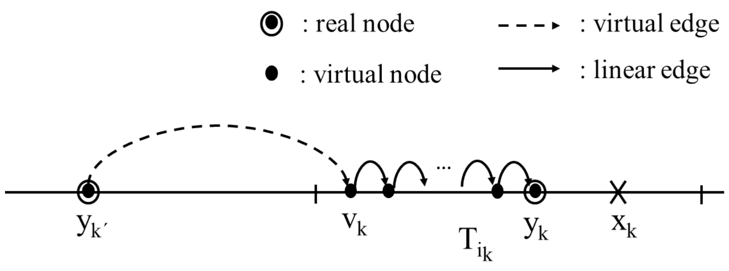

Fig. 1. An example of one iteration of the main routing loop.

Claim. At every iteration k it holds w.h.p. that $k > k'$, where k' was the value of k in the previous iteration of the while loop, and $y_k \in T_{i_k}$.

Proof. Note that on the first iteration of the loop, $x_k = v_k = y_k$, so the claim holds.

Now suppose $k > 0$ and that the claim holds for the iteration prior to the current iteration, when k was equal to k'. Either $v_k = l(y_{k'})$ or $v_k = r(y_{k'})$, which implies $x_{k'+1} = l(x_{k'})$ or $x_{k'+1} = r(x_{k'})$ respectively. In either case, since $y_{k'}$ and $x_{k'}$ are both in $T_{i_{k'}}$ and each T_i has length $(1/2)^j$ and is offset from 0, $x_{k'+1} \in T_{i_k}$, implying that $k \geq k' + 1$. Furthermore, if $y_k \notin T_{i_k}$ then $|S_k| \geq c \log n/2n$ since we move in the direction from v_k to the midpoint of T_{i_k} in Lines 26 – 28. Since by definition there are no real nodes in the interior of S_k,

$$P[|S_k| \geq c \log n/2n] \leq \left(1 - \frac{c \log n}{2n}\right)^n < e^{-(c \log n)/2} = n^{-c'/2},$$

where $c' = \frac{c}{\ln 2}$. Thus w.h.p. $y_k \in T_{i_k}$. $\square$

Since we always take k to be maximum index such that $x_k \in T_{i_k}$ (Line 17), this also implies that the message will never return to an interval after leaving it, else we would have $x_l \in T_{i_k}$ with $l > k$, a contradiction. Hence, since $y_k \in T_{i_k}$ implies that $S_k \subseteq T_{i_k}$, we have that all S_k's are disjoint w.h.p.

Let $S = \bigcup S_k$. We will bound the total length of S w.h.p. Assume $|S| > \frac{2c \log n}{n}$. Since each S_k contains only one real node by definition, there is a set of real nodes V_0 of size at least $n - \kappa$ such that no node in V_0 falls in S. We have

$$\begin{aligned} P[\text{no } v \in V_0 \text{ falls in } S] &\leq (1 - \frac{2c \log n}{n})^{n-\kappa} \\ &\leq exp((\frac{-2c \log n}{n}) \cdot (n - 2\kappa)) \\ &= exp(-2c \log n) \cdot exp(\frac{4c\kappa \log n}{n}) \\ &= O(\frac{1}{n^{2c'}}) \end{aligned}$$

for $c' = \frac{c}{\ln 2}$, since $\kappa = \log n + \Theta(1)$. Thus w.h.p. $|S| \leq \frac{2c \log n}{n}$.

Then according to Lemma 3, w.h.p. the number of virtual nodes in S is $O(\log n)$. Since the S_i's were shown to be disjoint w.h.p., this implies that the number of hops taken before Step 32 of the algorithm is $O(\log n)$ w.h.p.

Finally, we bound the number of hops in Lines $35 - 37$. After taking the final virtual hop we arrive at v_κ, which we know is in the same interval, T_{i_κ} as x_κ. Thus $|v_\kappa - x_\kappa| \leq \frac{c \log n}{n}$. Furthermore, since x_κ shares its more significant κ bits with the destination, w, we have

$$|x_\kappa - w| \leq (1/2)^\kappa = (1/2)^{\log n + c'} = \frac{2^{-c'}}{n}.$$

Thus after all bits have been fixed, with high probability, the packet will be at a distance from its destination which is $O(\frac{c \log n}{n})$. Hence by Lemma 3 the total number hops between the current location and the destination is $O(\log n)$ w.h.p. $\square$

5 Self-stabilization

We use the standard synchronous message passing model already used by Onus et al. [17]: the time steps are synchronized and all messages sent out at time step t arrive at their destinations before the beginning of time step $t + 1$. That is, no messages are ever in transit at the beginning of a new time step so that we do not have to worry about old messages that are still in the system. In addition to that we assume that there are no fake or outdated node identifiers in the system at any time. If so, we would have to worry about failure detectors which we do not want to do here to keep the treatment simple. We also assume that during the self-stabilization process the node set does not change. While joining nodes would be of no danger and would only delay the time needed till the self-stabilization process finishes, leaving nodes would cause outdated node identifiers, which we are not considering here.

Given the assumptions above, we show that the LDB network can self-stabilize from any initial state in which the nodes form a weakly connected graph. In the absense of any outside means that would allow disconnected nodes to reconnect, the assumption that the graph be initially weakly connected is necessary for the self-stabilization mechanism to ensure that at the end all nodes in the system form the desired topology. In the following, whenever we use the word "connected", it means "weakly connected".

Our self-stabilization mechanism builds on and extends the linearization technique of Onus et al. [17] as well as Kniesburges et al. [13]: in each time step, each node v with left neighbors $u_\ell < u_{\ell-1} < ... < u_1$ and right neighbors $w_1 < w_2 < ... < w_r$ replaces every edge (v, u_i) with $i > 1$ with (u_{i-1}, u_i) and every edge (v, w_i) with $i > 1$ with (w_{i-1}, w_i) by contacting the corresponding neighbors (i.e., it *linearizes* its neighborhood which explains the name of the technique as well as the name LDB we gave to our network). Also, it asks u_1 to establish (u_1, v) and w_1 to establish (w_1, v). Due to our message passing model, all notifications sent out by v in order to establish these edges can be received and processed by its neighbors so that their neighborhoods include the new edges at the beginning of the next time step. As shown in [13] (Section 3.1.2), for any weakly connected graph G of size n, at most $O(n)$ time steps are needed by the linearization rule above to transform G into a bi-directed sorted list. However, we have to deal here with the problem that we cannot let the virtual edges participate in the linearization of v, $l(v)$ and $r(v)$ as otherwise we would never reach a stable network (because v would continually introduce $l(v)$ and $r(v)$ to its closest neighbors). Thus, the linearization should only be applied to the non-virtual edges of v, $l(v)$ and $r(v)$ for all nodes v.

Suppose now that we start with some arbitrary directed network $G = (V, E)$ that is weakly connected, where V includes the real as well as the virtual nodes. Let $E' \subseteq E$ be the set of all non-virtual edges and let $G' = (V, E')$. Since the linearization does not include the virtual edges, the possibility remains that G' is not connected even though G is connected. To stabilize from this state to the desired LDB topology, we introduce a light-weight probing algorithm for each node v to determine if there is a path along the non-virtual edges leading from v to v's virtual nodes. We will show that by performing both the probing and the linearization algorithms, a network in any weakly connected state (over both virtual and non-virtual edges) will converge to the LDB in $O(n)$ steps.

5.1 Linear Probing

Let $x \in V$ be a real node. At each time step, x will probe for its left and right virtual node. The probing for the left virtual node is given in Algorithm 2, and the probing for the right virtual node works in an analogous way. The following property immediately follows from Algorithm 2.

Algorithm 2 Probing in the LDB

```
1:  $y = pred(x)$
2:  while $y$ is a virtual node do
3:      if $y = l(x)$ then
4:          Exit
5:      else if $pred(y)$ does not exist then
6:          Establish a non-virtual edge between $x$ and $l(x)$
7:          Exit
8:      else
9:          $y = pred(y)$
10:     end if
11: end while
12: $y = l(y)$
13: while $y < l(x)$ do
14:     if $succ(y)$ does not exist then
15:         Establish a non-virtual edge between $x$ and $l(x)$
16:         Exit
17:     end if
18: end while
19: if $y \neq l(x)$ then
20:     Establish a non-virtual edge between $x$ and $l(x)$
21: end if
```

Lemma 4. *For every real node v and every graph $G' = (V, E')$ formed by the non-virtual edges, the probing of v for $l(v)$ or $r(v)$ terminates in at most $3n$ steps.*

In the stable state, i.e., the ideal LDB network has been established, the probing is also very light-weight as stated in the next lemma, which follows from the fact that the real and virtual nodes are distributed uniformly at random in $(0, 1)$.

Lemma 5. *In the ideal LDB network with labels chosen uniformly at random for the real nodes it holds that for every real node v, the expected length of the path travelled by its probe to $l(v)$ and $r(v)$ is a constant.*

Hence, in the stable state, the linearization rule together with the probing rule only involves an expected constant number of steps to check the correctness of the LDB network. Thus, any faults can be detected quickly.

5.2 Convergence of linearization with linear probing

Finally, we prove that linearization with linear probing quickly converges to the desired LDB topology.

Theorem 2. *Using linearization together with linear probing, any weakly connected network (over virtual and non-virtual edges) will converge to the LDB network within $O(n)$ time steps.*

Proof. Let $G = (V, E)$ be the graph containing all virtual and non-virtual edges and $G' = (V, E')$ the the graph containing only the non-virtual edges. We need a sequence of lemmas to prove the theorem. The first lemma shows that weak connectivity is preserved for any pair of nodes.

Lemma 6. *Consider any connected component C in G'. If C is connected at the beginning of step t, then C is also connected at the beginning of step $t + 1$.*

Proof. As we do not have any fake or outdated node identifiers, the linearization rule will only perform transformations that preserve connectivity for the non-virtual edges (namely, the neighborhood of each node is transformed into a sorted list). As the probing mechanism may only create additional non-virtual edges, the connectivity property of C will be preserved. □

Lemma 7. *If G is connected but G' is not connected, then there must be a real node v that is not connected to $l(v)$ or $r(v)$ in G'.*

Proof. Suppose that G is connected and every real node $v \in V$ is connected to $l(v)$ and $r(v)$ in G' but G' is not connected. Let $C_1, \ldots, C_k$ be the connected components in G', $k \geq 2$. As these are connected in G, there must exist a real node v in some C_i with $l(v)$ or $r(v)$ being in C_j for some $j \neq i$. However, that contradicts our assumption that all real nodes are connected to their left and right virtual nodes in G' which completes the proof. □

Now we are ready to prove an upper bound on the number of steps it takes until G' is weakly connected.

Lemma 8. *If G is connected, then it takes at most $O(n)$ steps until G' is connected.*

Proof. Suppose that G' is initially not connected. Then it follows from Lemma 7 that there must be a real node v that is not connected to $l(v)$ or $r(v)$ in G'. W.l.o.g. we assume that v is not connected to $l(v)$. Then we follow the left probing of v till it reaches a real node or ends before reaching one. In the latter case v establishes a non-virtual edge to $l(v)$, so v is in the same connected component of G' as $l(v)$. Otherwise, let v' be the real node reached by the probing of v. Suppose that v's probe is not able to reach $l(v)$ from $l(v')$. Then v establishes a non-virtual edge to $l(v)$ and also in this case v and $l(v)$ are in the same connected component in G'. Hence, it remains to consider the case that v's probe succeeds in reaching $l(v)$ from $l(v')$. Then it follows from Lemma 4 that after $O(n)$ steps $l(v)$ and $l(v')$ are in the same connected component in G'. As Lemma 6 guarantees that v and v' remain in the same connected component of G', it follows that if v' and $l(v')$ are in the same connected component of G' after $O(n)$ steps, so are v and $l(v)$. Therefore, instead of focussing on the probe initially sent out by v, we focus on the probe initially sent out by v'. Continuing with the same arguments for v' as for v, it either holds that v' is connected to $l(v')$ after $O(n)$ steps or there is a node v'' with the property that v'' is in the same connected component as v' and $l(v'')$ is in the same connected component as $l(v')$ after $O(n)$ steps. In the latter case we switch to v'' and consider its initial probe for $l(v'')$. Continuing with these arguments, we must end up with a real node w that is in the same connected component as v after $O(n)$ steps, and w is connected to $l(w)$ in G' after $O(n)$ steps, which is in the same connected component as $l(v)$ after $O(n)$ steps. Hence, after $O(n)$ steps v is connected to $l(w)$ in G'. Since this argument applies to all nodes v that are initially not connected with $l(v)$ in G', the lemma follows. □

Finally, we need the following lemma, which has already been shown in [13].

Lemma 9. *If G' is connected, then the linearization rule ensures that after $O(n)$ steps G' forms a bi-directed sorted list.*

As it is known that the bi-directed sorted list is the only stable structure of the linearization rule [17] and in that case the linear probing will not add any further edges, the theorem follows. □

6 Join and Leave

We rely on the self-stabilization rules above to ensure that we can have very simple join and leave operations and still maintain the network structure. When a node v joins a network via a node w, it simply establishes a non-virtual edge between w and each of v, $l(v)$, and $r(v)$. Then our self-stabilization mechanism will ensure that the nodes will be placed in their proper location in $O(n)$ rounds. Other join operations are possible, such as using the routing algorithm to place the nodes more quickly within the network, but they are not discussed here. Since the self-stabilization rules can repair the network from any weakly connected state, our simple join mechanism can handle arbitrary concurrent join operations, which can be quite tricky to handle with a dedicated join operation. Also, notice that the routing mechanism will still work correctly while nodes are joining for two reasons: (1) the routing just relies on the virtual edges and the edges specified by *pred* and *succ* to proceed,and (2) joining nodes will only affect the *pred* and *succ* values of the nodes already in the system when they reached their right place.

When a node v leaves the network, v must simply introduce *pred*(v) and *succ*(v) to one another and the same for $l(v)$ and $r(v)$. Following this, the network will immediately be in a correct state with no additional work.

Finally, we show that if a node leaves without properly introducing its neighbors, the network will remain weakly connected with very high probability. So we know that the self-stabilization algorithms will still be able to return the network to a proper state.

Theorem 3. *Let G be an LDB network and let v be a real node in G. If there exists a real node $w \in G$ such that $l(v) < w < r(v)$, then the graph $G - \{v, l(v), r(v)\}$ is weakly connected.*

Proof. Partition $(0,1)$ into four regions $R_1 = (0, l(v))$, $R_2 = (l(v), v)$, $R_3 = (v, r(v))$, $R_4 = (r(v), 1)$. Assume $w < v$. The case where $w > v$ is symmetric. First note that from the linearization edges all nodes in a single region R_i are connected. Also note that any region with at least one virtual node but no real nodes is connected to another region, since each virtual node must have an incoming edge from a real node. Since $w < v$, $\frac{w}{2} < \frac{v}{2}$. Thus $l(w) \in R_1$, and so R_2 is connected to R_1. If there is a real node u in $R_3 \cup R_4$, then consider two cases:

Case 1: $v \leq \frac{1}{2}$. Then $v < \frac{w+1}{2} = r(w) < r(v)$, so R_3 is connected to R_1 and R_2. If $u \in R_3$ then $r(u) > r(v)$ so R_3 is connected to R_4 and so the whole graph is connected. If $u \in R_4$ then $l(u) < r(v)$ so R_4 is connected to $R_1 \cup R_2 \cup R_3$.

Case 2: $v \geq \frac{1}{2}$. Then $\frac{u}{2} < v$. If $u \in R_3$ then this implies R_3 is connected to $R_1 \cup R_2$. Also, since $u > v$, $r(u) > r(v)$ so R_3 is connected to R_4 and thus the entire graph is connected. If $u \in R_4$ then R_4 is connected to $R_1 \cup R_2$ and if there is a real node $y \in R_3$ then $r(y) \in R_4$ so the entire graph is weakly connected. □

Corollary 2. *If a node v leaves in the LDB with no further action, the network remains weakly connected with probability* $1 - \frac{1}{2^{n-1}}$.

Proof. The interval between $l(v)$ and $r(v)$ has length $\frac{1}{2}$. By Proposition 3 the network will be weakly connected if there are no real nodes in $(l(v), r(v))$, which has probability $\frac{1}{2^{n-1}}$. □

7 Conclusion

In this paper we presented the first self-stabilizing dynamic overlay network construction based on a De Bruijn graph, which also retains the main attractive properties of the classical De Bruijn construction, namely, logarithmic time routing and constant node degree. As future work, we believe that our approach, which uses an underlying sorted list of the nodes in order to aid in the network routing and self-stabilization, can also be generalized for other popular topologies, in particular ones that follow the continuous-discrete approach by Naor and Wieder.

References

1. I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. *Proc. of the 17th Intl. Parallel and Distributed Processing Symposium (IPDPS)*, p. 40, 2003.
2. A. Berns, S. Ghosh, and S. V. Pemmaraju. Brief announcement: a framework for building self-stabilizing overlay networks. *Proc. of the 29th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 398–399, 2010.
3. E. Caron, F. Desprez, F. Petit, and C. Tedeschi. Snap-stabilizing prefix tree for peer-to-peer systems. *Parallel Processing Letters* 20(1):15–30, 2010.
4. T. Clouser, M. Nesterenko, and C. Scheideler. Tiara: A self-stabilizing deterministic skip list. *Proc. of the 10th Intl. Symposium on Self-Stabilizing Systems (SSS)*, pages 124–140, 2008.

5. C. Cramer and T. Fuhrmann. *Self-stabilizing ring networks on connected graphs.* Technical Report 2005-5, System Architecture Group, University of Karlsruhe, 2005.
6. N. De Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen 49*, 1946, pp. 758–764.
7. P. Fraigniaud, P. Gauron. D2B: A De Bruijn based content-addressable network. *Theoretical Computer Science* 355(1):65–79, 2006.
8. D. Gall, R. Jacob, A. Richa, C. Scheideler, S. Schmid, and H. Täubig. Time complexity of distributed topological self-stabilization: The case of graph linearization. *Proc. of the 9th Latin American Theoretical Informatics Symposium*, pages 294–305, 2010.
9. R. Jacob, A. Richa, C. Scheideler, S. Schmid, and H. Taeubig. A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. *Proc. of the 28th ACM Symposium on Principles of Distributed Computing (PODC)*, 2009, pp. 131–140.
10. R. Jacob, S. Ritscher, C. Scheideler, and S. Schmid. A self-stabilizing local Delaunay graph construction. *Proc. of the 20th Intl. Symposium on Algorithms and Computation (ISAAC)*, 2009.
11. M. Kaashoek and D. Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.
12. D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. *Proc. of the 29th ACM Symposium on Theory of Computing (STOC)*, 1997.
13. S. Kniesburges, A. Koutsopoulos, and C. Scheideler. Re-Chord: A self-stabilizing Chord overlay network. To appear in *Proc. of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2011.
14. D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience. *Proc. of the 2003 ACM SIGCOMM Conference* ,pp. 395-406, 2003.
15. D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. *Proc. of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.
16. M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approch. *Proc. of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 50–59, 2003.
17. M. Onus, A. Richa, and C. Scheideler. Linearization: Locally Self-Stabilizing Sorting in Graphs. *Proc. of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007.
18. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. *Proc. of the ACM SIGCOMM Data Communication Festival*, 2001.
19. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Proc. of the IFIP/ACM Intl. Conference on Distributed Systems Platforms (Middleware)*, pp. 329–350, 2001.
20. C. Scheideler and S. Schmid. A distributed and oblivious heap. *Proc. of the 36th Intl. Colloquium on Automata, Languages and Programming (ICALP)*, pp. 571–582, 2009.
21. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer look-up protocol for internet applications. *IEEE/ACM Transactions on Networking* 11(1): 17–32, 2003.