The Hyperring: A Low-Congestion Deterministic Data Structure for Distributed Environments

Baruch Awerbuch* Christian Scheideler†

Abstract

In this paper we study the problem of designing searchable concurrent data structures with performance guarantees that can be used in a distributed environment where data elements are stored in a dynamically changing set of nodes. Searchable data structures are data structures that provide three basic operations: INSERT, DELETE, and SEARCH. In addition to searching for an exact match, we demand that for a data structure to be called "searchable", Search also has to be able to search for the closest successor or predecessor of a data item. Such a property has a tremendous advantage over just exact match, because it would allow to implement many data base applications.

We are interested in finding a searchable concurrent data structure that has (1) a low degree, (2) requires a small amount of work for INSERT and DELETE operations, and (3) is able to handle concurrent search requests with low congestion and dilation.

We present the first deterministic concurrent data structure, called *Hyperring*, that can fulfill all of these objectives in a polylogarithmic way. In fact, the Hyperring has a degree of $O(\log n)$, requires $O(\log^3 n)$ work for INSERT and DELETE operations, and can handle concurrent search requests to random destinations, one request per node, with congestion and dilation $O(\log n)$ w.h.p.

Most of the previous solutions for distributed environments are not searchable (in our sense) but only provide exact lookup, and those that are searchable do not have proofs about the congestion caused by concurrent search requests.

1 Introduction

A *searchable data structure* has to provide three basic operations: Insert(d), Delete(name), and Search(name). Insert(d) inserts a data item d with some name into the structure, Delete(name) removes the data item with a given name from the structure, and Search(name) returns the data item representing the closest match (e.g. the closest prefix) to name that has been inserted and has not been deleted (e.g.,[4]).

A *precise-lookup dictionary* is a data structure that looks up exact matches only. It is relatively easy to implement such a data structure via oblivious methods such as hashing. Examples of concurrent data structures with this approach are Chord, CAN, Pastry, and Tapestry [22, 18, 19, 23]. However, the resulting structures do not have the ability to run range queries, fuzzy search, database queries, etc in an efficient way. It takes more work to implement searchable data structures capable of supporting predecessor or successor search. In the sequential domain, these can be implemented via adaptive methods such as 2-3 trees, B-trees, and red-black trees.

Implementing adaptive, searchable data structures in a distributed system poses some challenges. A 2-3 tree, for example, is inappropriate for a distributed environment because of high congestion at the root. What is needed is a distributed, low-congestion analog of 2-3 trees. Many attempts have been made to solve this problem. Work on concurrent variants of search trees has been reported in [10, 13, 16, 21]. However, instead of parallelizing the data structure *itself*, these approaches only parallelize the *way* it is accessed, which is not suitable for a dynamic and error-prone distributed environment.

Recently, randomized solutions for concurrent and searchable data structures suitable for a distributed environment were suggested, essentially independently, by Li and Plaxton [11], Aspnes and Shah [1] and Harvey et al [8]. The underlying pointer structures are called "hyperdelta networks", "skip graphs" and "skip nets". They constitute a simple and elegant extension of a randomized skip list data structure of Pugh [17] to the distributed environment. In [11, 1, 8], only some basic properties of these data structures such as diameter and expansion were shown, leaving the following issues open: How to keep the congestion low for concurrent search, insert, or delete operations, and how to get rid of randomization in the construction of the data structure?

The contribution of this paper is to propose solutions to these problems. More precisely, we present local, deterministic update rules for insert and delete operations and prove upper and lower bounds on their effect on the data structure. The outcome of our investigations is a searchable deterministic data structure, called *Hyperring*, that can be viewed as a low-congestion version of a 2-3 tree or the deterministic skip list pro-

*Dept. of Computer Science, Johns Hopkins University, Baltimore, USA. Email: baruch@cs.jhu.edu. Supported by NSF grant ANIR-0240551 and NSF grant CCR-0311795.

†Dept. of Computer Science, Johns Hopkins University, Baltimore, USA. Email: scheideler@cs.jhu.edu. Supported by NSF grant CCR-0311121 and NSF grant CCR-0311795.

posed by Munro et al [14].

Independently of our work, Harvey and Munro [9] also proposed a deterministic searchable concurrent data structure that is similar to our structure. However, as it turns out from our results, their structure does not even guarantee a polylogarithmic congestion whereas we can show logarithmic congestion bounds.

An obvious advantage of a local deterministic over a randomized solution is the ability to *locally self-correct* the data structure in order to return the data structure to a low-congestion state after an adversarial disruption or unlucky combination of keys. This property is called self-stabilization by Dijkstra in his 1974 paper [6] and is considered an important property in existing peer-to-peer systems [22, 18, 19]. We comment that *by definition*, pseudo-random constructions cannot be self-correcting. However, the work in [22, 18, 19] deals with some systems aspects of self-correction.

1.1 Existing work on concurrent searchable data structures Imagine a collection of data items Data being sorted in a doubly-linked list based on their names. To make searching efficient, we need shortcut pointers. A naive way of doing this is to assign ranks to the items based on some global (e.g. lexicographical) ordering of their names. Each item $d \in \mathsf{Data}$ keeps pointers to all items d' whose ranking is exactly 2^i larger than its own for all i. The pointer graph is essentially a hypercube. Because of the expansion properties of the hypercube, it can be shown that, in a data structure with n data items, n concurrent search requests to random items can be executed with congestion and dilation $O(\log n)$, w.h.p. However, ranking is a very sensitive function; each data item that joins or leaves changes the ranking, making dynamic operations (Insert, Delete) expensive if a perfect ranking is to be maintained: the update work would be $\Omega(n)$.

To solve this problem, consider the approach in [11, 1, 8] of interconnecting nodes in a hierarchy of cycles on top of a cycle C containing all data items (or nodes) in sorted order. Each node pads its name with a random bit string. First, we decompose C into two sorted cycles: C_0, which contains all nodes with first bit 0, and C_1, which contains all nodes with first bit 1. We continue this process recursively on C_0 and C_1, generating even smaller cycles in the next higher level, namely C_{00}, C_{01}, C_{10}, and C_{11}, etc. That is, at level i of the recursion, a cycle $C_{b_1,b_2\ldots b_i}$ is a doubly-linked cycle of all nodes with the padding sequence $b_1, b_2 \ldots b_i$. Since the padding sequences are chosen at random, it is not hard to show that the work for inserting or deleting a data item is $O(\log n)$ w.h.p., and searching for a data item only requires to traverse $O(\log n)$ edges w.h.p.

1.2 The Hyperring data structure Next, we outline informally the main ideas of our construction. The exact model, the problem and complexity measures are defined in Section 2. One of our main results is:

THEOREM 1.1. *The* Hyperring *is a deterministic concurrent data structure with the following features:*

- *the* degree *of each node in the data structure is at most $O(\log n)$*
- *the* work *required is $O(\log n)$ for a* Search *operation and $O(\log^3 n)$ for* Insert *and* Delete
- *the* congestion *for n concurrent* Search *operations forming a permutation, or one operation per node with random destination, is $O(\log n)$ w.h.p. (where the probability only depends on the search algorithm)*
- *the* node expansion *of the data structure is $\Omega(1/\log n)$.*

The degree of a node and the expansion match the bounds known for skip graphs [11, 1, 8]. However, the congestion created by concurrent operations has not been shown to be even polylogarithmic in [11, 1, 8], and our results imply that the only previously known deterministic concurrent data structure suitable for a distributed environment [9] does not even guarantee a polylogarithmic congestion.

Recall the bit padding approach for skip graphs. In the Hyperring, we use a deterministic rule for bit padding (see also [9]): there can be at most two consecutive 0's and 1's in each ring. Pictorially, links corresponding to two consecutive 0's or 1's, e.g. $1, 0, 0, 1$, look like a bridge (see Figure 1).

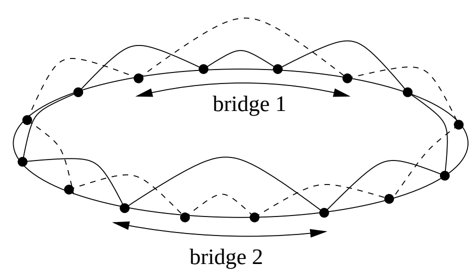

Figure 1: An example of a Hyperring. Padding creates 0 and 1 sub-rings. Notice bridge 1 with sub-sequence $1, 0, 0, 1$ and bridge 2 with $0, 1, 1, 0$. The bridges have a distance of 5 from each other.

In addition to requiring at most two consecutive 0's or 1's, we also require that these sequences, respectively their corresponding bridges, are sufficiently far apart from each other in each ring. We call a Hyperring k-*separated* if the minimal distance between two bridges on a ring is at least k (i.e. there are at least $k-1$ nodes between them). The Hyperring in Figure 1 is, for example, 5-separated. It turns out that separation is crucial for handling congestion.

1.3 Main insights In Section 2 we state the formal specifications of a concurrent data structure and state precisely the measures used in Theorem 1.1 and the theorems below. In Section 3, we prove:

THEOREM 1.2. *A Hyperring with a constant separation has, in the worst case, a poor expansion (i.e.* $\alpha = O(1/n^{\epsilon})$*).*

This also implies a bad congestion for routing requests. Hence, a Hyperring must have a non-constant separation to be useful for concurrent operations. However, here we face a dilemma. If the separation we require is, for example, $\log n$, then updates to the Hyperring done at some previous time might have used a completely different n than the n used by current updates. It may seem like a disaster, since we may be forced from time to time to revisit old insertions and deletions once the system grows beyond a certain size. Fortunately, we can show in Section 4 *that this is not necessary.*

THEOREM 1.3. *Logarithmic separation at the time the request was executed is sufficient to guarantee* $\alpha = \Omega(1/\log n)$.

In other words, it is sufficient that a bridge is a logarithmic number of nodes away from existing bridges *at the time when it is created*, where the logarithm is taken to the current number of nodes, and it is OK if this is not true any more in the future.

2 Model and statement of the problem

The following model and measures need to be defined in order to formally state Theorem 1.1.

2.1 Searchable concurrent data structure and performance metrics We require a (concurrent) searchable data structure $\mathcal{C}$ to offer the following operations

- $\mathsf{Insert}(d)$: adds data item d with name (identifier) $\mathsf{Name}(d)$ to $\mathcal{C}$.
- $\mathsf{Delete}(\mathsf{name})$: removes d with $\mathsf{Name}(d) = \mathsf{name}$ from $\mathcal{C}$.
- $\mathsf{Search}(\mathsf{name})$ retrieves from $\mathcal{C}$ the closest successor d^* of name, i.e.

$$d^* = \mathsf{argmin}\{\mathsf{Name}(d') \mid d' \in \mathsf{Data}, \mathsf{Name}(d) \geq \mathsf{name}\}$$

For simplicity, we assume that every name is only once in the system at any time.

We use the standard performance metrics in communication theory to analyze our data structures. All metrics are meant for the worst case, where by "worst case" we mean the worst case over all possible inputs (i.e. selections of names for the data items).

- *degree*: this measures the maximum degree of a node in the data structure, and thus the *space* necessary to maintain it.
- *work*: this measures the number of times a message has to be sent over a link in a Insert, Delete, or Search operation. We assume that messages can carry at most $O(\log n)$ bits.
- *(node) expansion*: this measures the connectivity or fault-tolerance of the data structure. The expansion α of a graph $G = (V, E)$ is defined as $\alpha = \min_{U \subset V,\, |U| \leq |V|/2} |N(U)|/|U|$ where $N(U) = \{w \in V \setminus U \mid \exists v \in U : (v, w) \in E\}$ is the neighbor set of U.
- *congestion* $\gamma(G)$: this measures the (expected value of the) maximum number of Search operations traversing a node in G for the case that every node has a Search request to a random destination.

To give some examples, the best expansion a constant degree graph can have is a constant (and these graphs are known as expanders), whereas the hypercube with n nodes is known to have an expansion of $O(1/\sqrt{\log n})$. The best congestion a constant degree graph with n nodes can have is $\Theta(\log n)$, and there is an easy search strategy for the hypercube with congestion $\Theta(\log n)$. (However, the hypercube is not optimal because it has logarithmic degree.) In contrast, a complete binary tree has an expansion of $O(1/n)$ and a congestion of $\Omega(n)$.

The expansion and congestion of a graph are closely related to each other.

CLAIM 2.1. *For every graph G of size n with congestion* $\gamma(G)$, $\alpha(G) = \Omega(1/\gamma(G))$.

Proof. Suppose that there is a set $U \subseteq V$ with $|U| \leq |V|/2$ and $|N(U)| = o(|U|/\gamma(G))$. Then consider the experiment of choosing a random search problem with n requests, one per node. It is easy to see that the expected number of requests that have to cross the cut $(U, \bar{U})$ is equal to $|U|$. Thus, there must be a node in $N(U)$ that is passed by $|U|/|N(U)| = \omega(\gamma(G))$ requests on expectation. However, according to the definition of the congestion, every node in G should only be passed by $O(\gamma(G))$ requests on expectation, creating a contradiction. $\square$

Notice that it is not possible in general to conclude from the expansion of a graph about its congestion, because just knowing that a data structure has a good expansion does not necessarily imply that it allows efficient searching. For the congestion to be low, it is important to select the right paths, which is a non-trivial task in dynamic networks.

We note that [12, 15] are the only results in the peer-to-peer literature so far addressing the congestion of concurrent searching (which they only do for DHT-based approaches).

3 Hyperrings of constant separation

In this section we present a family of dynamic graphs called *k-separated Hyperrings*. It offers two primitives:

- Add(v): This adds node v to the Hyperring next to the node that v contacted.
- Remove(v): This removes node v from the Hyperring.

Detailed proofs of this section can be found in [3].

3.1 The basic construction The basic idea behind the Hyperring is similar to [1, 8, 9, 11]: it is organized as a hierarchical structure of rings. Suppose it currently contains n nodes. Then the Hyperring consists of approximately $\log n$ levels of rings, starting with level 0. Each level $i \geq 0$ consists of approximately 2^i directed cycles of approximately $n/2^i$ nodes, which we call *rings*. All rings have the same orientation. For every ring R at level i, two rings of level $i+1$ share its nodes in an intertwined fashion. A ring at level i will also be called an *i-ring* in the following, and a level i edge will simply be called an *i-edge*. Consider some i-ring R and let (u, v, w, x) be four consecutive nodes on R. We say that (u, v, w, x) form an *i-bridge* (or simply a *bridge* if i is clear from the context) if there is an $(i+1)$-edge from u to x and an $(i+1)$-edge from v to w. An $(i+1)$-edge is called *perfect* if it bridges exactly two i-edges.

It is possible to maintain a Hyperring with at most one bridge in every ring. However, in this case we would create too much update work for Insert or Delete operations. Instead, we only demand that i-bridges are sufficiently far apart from each other. A Hyperring is called *k-separated* if in every i-ring R the i-bridges on R are at least k nodes apart from each other, which means that there are at least $k-1$ nodes between the quadruples of nodes forming a bridge. We start with a few properties of Hyperrings which are easy to prove.

LEMMA 3.1. *For every $k \geq 0$, the k-separated Hyperring has a maximum degree of at most $2(1 + 2/(k+1)) \log n$ and a diameter of at most $3 \log n$.*

So concerning the degree and diameter, k-separated Hyperrings look appealing even for $k = 0$. Unfortunately, we can also show the following main result.

THEOREM 3.1. *For every $k \geq 0$, the k-separated Hyperring has, in the worst case, an edge expansion of*

$$O(1/n^{1/(2(3(k+4))^2)}) \ .$$

Proof. To simplify the calculations, we will ignore rounding effects throughout the proof because they turn out to be insignificant.

We will construct a Hyperring that gives a node set of size $n/2$ with a poor expansion. For this we need the following lemma.

LEMMA 3.2. *Consider some ring R and some set S of consecutive nodes in R with $|S| \leq |R|/2$. Then two intertwined rings R' and R'' can be constructed on top of R so that*

$$|V(R') \cap S| = \frac{1}{2}\left(1 - \frac{1}{3(k+4)}\right)|S|$$

and

$$|V(R')| = \frac{1}{2}\left(1 + \frac{1}{3(k+4)}\right)|V(R)|$$

and R' and R'' do not violate the k-separation.

Proof. Consider some set S' of consecutive nodes on R. If we require the Hyperring to be k-separated, then we can minimize the number of nodes in $V(R') \cap S'$ by using sequences of $1 + \lceil k/2 \rceil$ edges where one edge bridges three edges and the $\lceil k/2 \rceil$ others bridge two edges in R. Hence, R' and R'' can be constructed on top of R without violating the k-separation so that

$$\begin{aligned} |V(R') \cap S'| &\leq \frac{|S|}{3 + 2 \cdot \lceil k/2 \rceil} \cdot (1 + \lceil k/2 \rceil) \\ &= \left(\frac{1}{2} - \frac{1}{4(\lceil k/2 \rceil + 1) + 2}\right)|S| \\ &\leq \frac{1}{2}\left(1 - \frac{1}{k+4}\right)|S'| \ . \end{aligned}$$

Now, select a set $S' \subseteq S$ of $|S|/3$ consecutive nodes in S. Then it is possible to construct a ring R' on top of S without violating the k-separation so that

$$\begin{aligned} |V(R') \cap S| &= \frac{1}{2} \cdot |S \setminus S'| + \frac{1}{2}\left(1 - \frac{1}{k+4}\right)|S'| \\ &= \frac{1}{2}\left(1 - \frac{1}{3(k+4)}\right)|S| \end{aligned}$$

We now search for an ϵ so that if R' covers $\frac{1}{2}(1 + \frac{\epsilon}{k+4})|V(R) \setminus S|$ nodes in $|V(R) \setminus S|$ then

$|V(R')| = \frac{1}{2}\left(1+\frac{1}{3(k+4)}\right)|V(R)|$. It holds that $\frac{1}{2}\left(1+\frac{\epsilon}{k+4}\right)\cdot(|V(R)|-|S|)+\frac{1}{2}\left(1-\frac{1}{3(k+4)}\right)|S| = \frac{1}{2}\left(1+\frac{1}{3(k+4)}\right)|V(R)|$

$$\begin{aligned}\Leftrightarrow\quad \frac{3\epsilon-1}{3(k+4)}\cdot|V(R)| &= \frac{3\epsilon+1}{3(k+4)}|S|\\ \Leftrightarrow\quad \epsilon &= \frac{|R|+|S|}{3(|R|-|S|)} \le 1\end{aligned}$$

because we assumed that $|S| \le |V(R)|/2$. Hence, it is possible to obtain the equations stated in the lemma, which completes the proof. □

Consider some set S of consecutive nodes on the 0-ring R_0 of size $n/2$. We construct a k-separated Hyperring with a bad expansion for S inductively.

Consider some i-ring R. If $|V(R)\cap S| \le |V(R)|/2$, then we apply Lemma 3.2 to $S_R = V(R)\cap S$ to obtain two intertwined rings on top of R. Otherwise, we apply Lemma 3.2 to $\bar{S}_R = V(R)\setminus S$ to obtain two intertwined rings on top of R.

Using this construction, we prove the following lemma.

Lemma 3.3. *For every i-ring R with $i \ge (\log n)/(1+1/(2(3(k+4))^2))$, either $S_R = V(R)$ or $S_R = \emptyset$.*

Proof. We can view the Hyperring as a tree $T = (V_T, E_T)$ with its 0-ring representing the root and edges directed towards the leafs. For every node v_R at level i representing an i-ring R, its sons represent the $i+1$-rings on top of R. Consider now an edge $(v_R, v_{R'})$ (i.e. R' is on top of R). If $|S_R| \le |V(R)|/2$, then it follows from the inductive construction that

$$\left.\begin{aligned}|S_{R'}| &= \tfrac{1}{2}\left(1-\tfrac{1}{3(k+4)}\right)|S_R|\\ \text{and}\quad |V(R')| &= \tfrac{1}{2}\left(1+\tfrac{1}{3(k+4)}\right)|V(R)|\end{aligned}\right\}\quad (3.1)$$

and hence,

$$\left.\begin{aligned}|S_{R''}| &= \tfrac{1}{2}\left(1+\tfrac{1}{3(k+4)}\right)|S_R|\\ \text{and}\quad |V(R'')| &= \tfrac{1}{2}\left(1-\tfrac{1}{3(k+4)}\right)|V(R)|\end{aligned}\right\}\quad (3.2)$$

If $|S_R| > |V(R)|/2$, it follows that

$$\left.\begin{aligned}|\bar{S}_{R'}| &= \tfrac{1}{2}\left(1-\tfrac{1}{3(k+4)}\right)|\bar{S}_R|\\ \text{and}\quad |V(R')| &= \tfrac{1}{2}\left(1+\tfrac{1}{3(k+4)}\right)|V(R)|\end{aligned}\right\}\quad (3.3)$$

and hence,

$$\left.\begin{aligned}|\bar{S}_{R''}| &= \tfrac{1}{2}\left(1+\tfrac{1}{3(k+4)}\right)|\bar{S}_R|\\ \text{and}\quad |V(R'')| &= \tfrac{1}{2}\left(1-\tfrac{1}{3(k+4)}\right)|V(R)|\end{aligned}\right\}\quad (3.4)$$

Consider now any path p through T from the root to some node v_R at level $i = (\log n)/(1+1/(2(3(k+4))^2))$. We cut p into subpaths $p_1, p_2, \ldots, p_k$ at those nodes $v_{\tilde{R}}$ with $|S_{\tilde{R}}| = |V(\tilde{R})|/2$. Then it must hold for each p_i that either $|S_{R'}| \le |V(R')|$ for every node $v_{R'}$ in p_i or $|S_{R'}| \ge |V(R')|$ for every node $v_{R'}$ in p_i. Furthermore, to get back to $|S_{R'}| = |V_{R'}|/2$ at the end, half of the edges in p_i must be of type (3.1) resp. (3.3), and the other half of the edges in p_i must be of type (3.2) resp. (3.4). Hence, if m_i is the size of the ring R_i at the beginning of p_i and ℓ_i is the number of edges in p_i, then

$$\begin{aligned}m_{i+1} &= \frac{1}{2^{\ell_i}}\left(1-\frac{1}{3(k+4)}\right)^{\ell_i/2}\left(1+\frac{1}{3(k+4)}\right)^{\ell_i/2} m_i\\ &= \frac{1}{2^{\ell_i}}\left(1-\frac{1}{(3(k+4))^2}\right)^{\ell_i/2} m_i\end{aligned}$$

for all $i < k-1$. Furthermore, if $s_i = |S_{R_i}|$, then

$$s_{i+1} = \frac{1}{2^{\ell_i}}\left(1-\frac{1}{(3(k+4))^2}\right)^{\ell_i/2} s_i\ .$$

For the last subpath p_k we distinguish between two cases. Suppose that at least half of the times in p_k, (3.1) resp. (3.3) applies. Then it holds for the final ring R that for $S' = S_R$ resp. $S' = \bar{S}_R$,

$$\begin{aligned}|S'| &\le \frac{1}{2^{\ell_k}}\left(1-\frac{1}{(3(k+4))^2}\right)^{\ell_k/2}\cdot s_k\\ &= \frac{1}{2^{\ell}}\left(1-\frac{1}{(3(k+4))^2}\right)^{\ell/2}\cdot |S|\\ &\le 2^{-\ell\left(1+\frac{1}{2(3(k+4))^2}\right)}\cdot|S|\end{aligned}$$

where ℓ is the length of p. Since $|S| = n/2$ and $\ell = (\log n)/(1+1/(2(3(k+4))^2))$, it follows that $|S_R| < 1$ resp. $|\bar{S}_R| < 1$ and therefore $S_R = \emptyset$ resp. $S_R = V(R)$.

Otherwise, suppose that at least half of the times in p_k, (3.2) resp. (3.4) applies. Then it holds for the final ring R that

$$\begin{aligned}|V(R)| &\le \frac{1}{2^{\ell_k}}\left(1-\frac{1}{(3(k+4))^2}\right)^{\ell_k/2}\cdot m_k\\ &= \frac{1}{2^{\ell}}\left(1-\frac{1}{(3(k+4))^2}\right)^{\ell/2}\cdot n\\ &\le 2^{-\ell\left(1+\frac{1}{2(3(k+4))^2}\right)}\cdot n\ .\end{aligned}$$

Since $\ell = (\log n)/(1+1/(2(3(k+4))^2))$, it follows that $|V(R)| \le 1$, and therefore R does not exist. Combining the cases proves the lemma. □

Since for every ring R with $S_R \neq \emptyset$ and $S_R \neq V(R)$, S_R can only be connected to two outside nodes via edges in R, it follows from Lemma 3.3 that S is connected to at most

$$2 \cdot \sum_{i=0}^{\frac{\log n}{1+1/(2(3(k+4))^2)}} 2^i \leq 2 \cdot 2^{(\log n)/(1+1/(2(3(k+4))^2))+1} = 4 \cdot n^{1-1/(2(3(k+4))^2)}$$

outside nodes. This proves the theorem. □

Theorem 3.1 and Claim 2.1 together imply the following result.

Corollary 3.1. *For every $k \geq 0$, the k-separated Hyperring has a congestion of $\Omega(n^{1/(2(3(k+4))^2)})$.*

Hence, no k-separated Hyperring with $k = O((\log n)^{1/2-\epsilon})$ for some constant $\epsilon > 0$ can have a polylogarithmic congestion. Hence, to have a good congestion, we need $k = \Omega(\sqrt{\log n})$. However, notice that when k depends on the size of the Hyperring, node insertions and deletions that have been performed in the past might have used a k that significantly differs from the k used by current insertions and deletions. Hence, parts of the Hyperring may be out of date. So the question is whether it is necessary to revisit these parts in order to bring the Hyperring up to date. Fortunately, as one of our main results, we show in the next section that *this is not necessary.*

3.2 Adding and removing nodes

First, we introduce some notation. Let $\mathrm{succ}_i(v)$ be the successor of v in its i-ring and $\mathrm{pred}_i(v)$ be the predecessor of v in its i-ring. For every node v on R, its $> i$-endpoints represent all endpoints of edges in v with level more than i. Notice that each node has two endpoints in each level. By "moving" the i-endpoints from u to v, we mean that we replace the i-edges $(\mathrm{pred}_i(u), u)$ and $(u, \mathrm{succ}_i(u))$ by the i-edges $(\mathrm{pred}_i(u), v)$ and $(v, \mathrm{succ}_i(u))$. By "permuting" the i-endpoints of u and v, we mean that we move the i-endpoints of u to v and the i-endpoints of v to u. Due to space limitations, we omit the proofs of the theorems in this subsection. See [3] for details.

Adding a node The basic approach behind Add is that we integrate a node u into the Hyperring level by level, starting with level 0. In each level i, we integrate the node by either removing an already existing bridge in its $k+2$-neighborhood or by creating a new bridge. A bridge is removed by first dragging it over to u by permuting $> i$-endpoints (see Figure 3). Then case (b) or (c) in Figure 2 is applied. Otherwise, we just apply case (a). Add terminates once we reach a ring of size in $\{4, \ldots, 7\}$ (for larger rings, two new subrings are created).

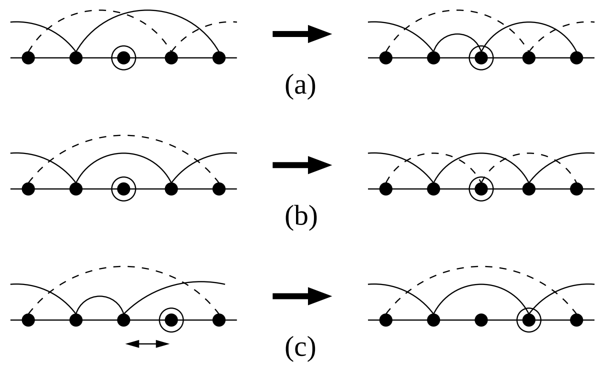

Figure 2: The three cases when adding a node. Case (c) reduces to case (b).

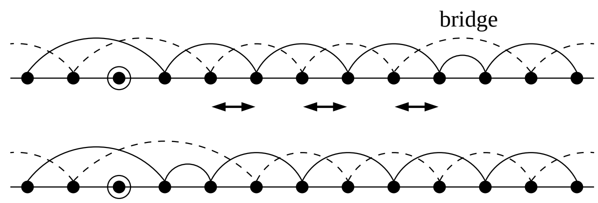

Figure 3: Permuting $> i$-endpoints drags the bridge over to obtain, e.g., case (c) in Figure 2.

Theorem 3.2. Add *preserves the k-separation of the Hyperring and requires $O(k \log^2 n)$ work.*

Removing a node We also remove a node u from the Hyperring level by level, starting with level 0. In each level, we remove the node by either removing an already existing bridge in its $k+2$-neighborhood or by creating a new bridge. A bridge is removed by first dragging it over (Figure 3) and then applying case (b) or (c) in Figure 4. Otherwise, we just apply case (a). Remove terminates once we reach a ring of size in $\{4, \ldots, 7\}$ (rings smaller than 4 are removed).

Theorem 3.3. Remove *preserves the k-separation of the Hyperring and requires $O(k \log^2 n)$ work.*

4 Hyperrings of non-constant separation

For every ring R, $|R|$ denotes the number of edges (or nodes) it consists of, and for every edge e, $|e|$ denotes the number of 0-edges bridged by it. For every node v, $\ell(v)$ denotes the number of levels it participates in.

Consider the case that every node v initiating Add or Remove sets $k_d = 6(d+3)$ as separation, where $d = \ell(v) + 2$. Since $d = O(\log n)$, Theorems 3.2 and 3.3 imply that Add and Remove operations require a work of at most $O(\log^3 n)$. (Although this bound may

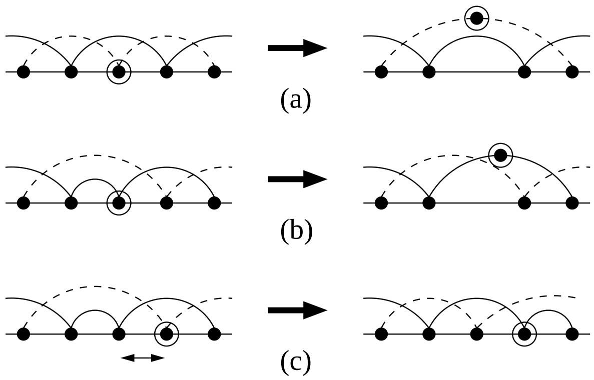

Figure 4: The three cases when removing a node. Case (c) reduces to case (b).

sound large, we note that with the help of some tricks, the work can be parallelized so that Add and Remove need $O(\log n)$ time to complete.) The tricky aspect with using such a separation parameter is that it is not a fixed, global parameter. Different nodes may use a different k_d, and the value of k_d can vary significantly over time. Nevertheless, we show that the following properties can be guaranteed at any time:

PROPOSITION 4.1.

1. *the* ring distortion is low, *i.e. for every i-ring R, $|R| \in [\frac{1}{2} \cdot n/2^i - 1, 2 \cdot n/2^i + 1]$ and*
2. *the* edge distortion is low, *i.e. for every i-edge e, $|e| \leq 4 \cdot 2^i$.*

For this we need some notation. For every set of consecutive nodes S on an i-ring R and every integer $d \geq 0$, b_d^S denotes a class of bridges that have a distance of at least k_d to other bridges in S, where $k_d = 6(d+3)$. The main invariants we want to prove are:

INVARIANT 4.1. *For every $i \geq 0$ and every set of consecutive nodes S on an i-ring R with $\sum_d b_d^S \geq 2$ it holds:*

1. *For every bridge B in b_d^S, no bridge is in the k_d-neighborhood of B and*
2. *$b_d^S \leq \frac{\min\{|S|, 2^{d-i+2}\}}{k_d}$.*

We prove this invariant by complete induction on the number of operations: Suppose that after operation q the invariant is still true. Then we can show the following lemma.

LEMMA 4.1. *Suppose that Invariant 4.1 holds. Then, for every $i \geq 0$ and every i-ring R, $|R| \in [\frac{1}{\sqrt{e}} \cdot \frac{n}{2^i} - 1,\ \sqrt{e} \cdot \frac{n}{2^i} + 1]$. Furthermore, it either holds for all nodes v that $\ell(v) \in [\lfloor \log n \rfloor - 1, \lfloor \log n \rfloor + 1]$ or for all nodes v that $\ell(v) \in [\lfloor \log n \rfloor, \lfloor \log n \rfloor + 2]$.*

Proof. Let R be some i-ring and R' be some $(i+1)$-ring on top of R. Let $b_R = \sum_j b_j^R$ be the total number bridges on R. Suppose that b' of the bridges assign the inner edge to R' and the remaining b'' bridges assign the outer edge to R'. Each time R' uses an inner edge, its offset to traverse R is reduced by one, and each time R' uses an outer edge, its offset is increased by one. Hence, it holds that

$$\begin{aligned} |R'| &= \frac{|R| + b' - b''}{2} = \frac{|R| + b' - (b_R - b')}{2} \\ &= \frac{|R| - b_R}{2} + b' . \end{aligned}$$

Notice that this is always an integer since b_R is even if and only if $|R|$ is even, because otherwise R' would not be able to go around R just once (which is guaranteed by the algorithm). Now, $b_R = \sum_d b_d^R$ and from Invariant 4.1 it follows (for $b_R \geq 2$) that

$$\sum_d k_d \cdot b_d^R \leq |R| \quad \text{and} \quad b_d^R \leq \frac{\min\{|R|, 2^{d-i+2}\}}{k_d} .$$

Obviously, b_R is maximized if using as many b_d^R with small d as possible, i.e.

$$b_R \leq \max\left\{1, \sum_{j \leq \log |R| + i - 2} \frac{\min\{|R|, 2^{j-i+2}\}}{k_j}\right\} .$$

Hence,

$$\begin{aligned} |R'| &\leq \frac{|R| + b_R}{2} \\ &\leq \frac{1}{2}\left(|R| + \max\left\{1, \sum_{j \leq \log |R| + i - 2} \frac{\min\{|R|, 2^{j-i+2}\}}{k_j}\right\}\right) \\ &\leq \frac{1}{2}\left(|R| + \max\left\{1, \sum_{j \leq \log |R| + i - 2} \frac{2^{j-i+1}}{3(j+3)}\right\}\right) \\ &\leq \frac{1}{2}\left(|R| + \max\left\{1, \frac{2^{\log |R| - 1}}{\log |R| + i + 1}\right\}\right) \\ &\leq \frac{1}{2} \cdot \max\left\{|R| + 1, \left(1 + \frac{1}{2(\log |R| + i + 1)}\right)|R|\right\} \end{aligned}$$

Since R decomposes into exactly two $(i+1)$-rings, it also holds that

$$|R'| \geq \frac{1}{2} \cdot \min\left\{|R| - 1, \left(1 - \frac{1}{2(\log |R| + i + 1)}\right)|R|\right\} .$$

This allows us to prove the following claim.

CLAIM 4.1. *For every i-ring R with $i \geq 1$,*

$$|R| \in \left[\left(1-\frac{1}{2\log n}\right)^s \frac{n}{2^i}-1,\ \left(1+\frac{1}{2\log n}\right)^s \frac{n}{2^i}+1\right]$$

for some $s < \log n$ (which represents the switching point from the right term of the max *(resp.* min*) expression to the left.*

Proof. The claim can be shown by complete induction on i. First we show that for all $i \leq s$,

$$(4.1) \quad |R| \in \left[\left(1-\frac{1}{2\log n}\right)^i \frac{n}{2^i},\ \left(1+\frac{1}{2\log n}\right)^i \frac{n}{2^i}\right].$$

This is certainly true for $i = 0$. Given that it is true for some i, it is also true for $i+1$, because for any i-ring R,

$$\begin{aligned} \log|R| + i + 1 &\geq \log\left(\left(1-\frac{1}{2\log n}\right)^i \frac{n}{2^i}\right) + i + 1 \\ &\geq \log n \end{aligned}$$

for all $i \leq \log n$. Consider now some i-ring R with $i \geq s$. Then it holds for any $i+1$-ring R' on top of R that

$$|R'| \in \left[\tfrac{1}{2}(|R|-1),\ \tfrac{1}{2}(|R|+1)\right] .$$

Using this it follows by complete induction that for any $i+s$-ring R' on top of an s-ring R,

$$(4.2) \quad |R'| \in \left[\frac{1}{2^i}|R| - \left(1-\frac{1}{2^i}\right),\ \frac{1}{2^i}|R| + \left(1-\frac{1}{2^i}\right)\right].$$

Combining (4.1) with (4.2) yields the claim. □

It is well-known that for all $s < \log n$,

$$\left(1-\frac{1}{2\log n}\right)^s > e^{-1/2} \quad \text{and} \quad \left(1+\frac{1}{2\log n}\right)^s < e^{1/2}$$

Hence, for every i-ring R with $i \geq 1$,

$$|R| \in \left[\frac{1}{\sqrt{e}} \cdot \frac{n}{2^i} - 1,\ \sqrt{e} \cdot \frac{n}{2^i} + 1\right] .$$

According to our rules for Add and Remove, the second highest ring R must have a size in $[8, 15]$. If there are two second highest rings R_1 and R_2 at different places of the Hyperring where R_2 is more than two levels higher than R_1, then there must be an i with

$$\frac{1}{\sqrt{e}} \cdot \frac{n}{2^i} - 1 \leq 15 \quad \text{and} \quad \sqrt{e} \cdot \frac{n}{2^{i+3}} + 1 \geq 8 .$$

This means that $2^i \geq n/(16\sqrt{e})$ and $2^i \leq \sqrt{e} \cdot n/56$, which is not possible. Hence, the highest rings can be by at most two levels apart. Thus, the number of levels in which the nodes participate must be either all in $[\lfloor \log n \rfloor - 1, \lfloor \log n \rfloor + 1]$ or all in $[\lfloor \log n \rfloor, \lfloor \log n \rfloor + 2]$. □

With the help of Lemma 4.1 we can prove that the invariant also holds after operation $q+1$.

LEMMA 4.2. *If Invariant 4.1 is true after operation q, then Invariant 4.1 remains true after operation $q+1$.*

Proof. Suppose that operation $q+1$ is an insertion of a node u in R. Let the neighbors of u be v and w. According to Lemma 4.1, it holds for every node v that $\ell(v) \geq \log n - 2$. Hence, our topology control scheme will check a neighborhood of k_d nodes around u with $d \geq (\log n - 2) + 2 = \log n$. If a bridge is found, it is removed, which certainly does not endanger Invariant 4.1. If no bridge is found in this neighborhood, then a bridge is created on top of u. If this bridge is in the $k_{d'}$-neighborhood of some existing bridge in R with $d' > d$, then we downgrade this bridge to d. This does not endanger Invariant 4.1(1) since we get back to neighborhoods of bridges that do not contain any other bridges. It therefore remains to check Invariant 4.1(2), i.e. for the d chosen by u it must hold that $b_d^S \leq \min\{|S|, 2^{d-i+2}\}/k_d$.

Since on one hand $|S| \leq |R|$ and $|R| \leq \sqrt{e} \cdot n/2^i + 1$ and on the other hand $d \geq \log n$ and therefore $2^{d-i+2} \geq 4 \cdot n/2^i$, it holds that $\min\{|S|, 2^{d-i+2}\} = |S|$. Hence, we only have to show that $b_d^S \leq |S|/k_d$. This is certainly true, because if no bridge is in the neighborhood of another bridge, then the bridges counting for b_d^S must be by at least k_d nodes apart, which proves the lemma for the case that a node is inserted.

It remains to consider the event that some node u has to be deleted in R. The proof for this follows along the same lines as for an insertion event and is therefore omitted here. □

Finally, using the invariant, we can prove a bound on the distortion of edges.

LEMMA 4.3. *For every i, the number of 0-edges bridged by an i-edge is at most $2\sqrt{e} \cdot 2^i$.*

Proof. Consider some fixed i-edge e. For some fixed $j \in \{0, \ldots, i-1\}$, let S be the sequence of $(j+1)$-level edges bridged by e and S' be the sequence of j-level edges bridged by S. Then it holds that

$$|S'| \leq 2|S| + \sum_d b_d^S .$$

From Invariant 4.1 and the proof of Lemma 4.1 it follows that

$$\sum_d b_d^S \leq \max\left\{1,\ \frac{|S|}{2(\log|S| + j + 1)}\right\} .$$

Hence,

$$\begin{aligned} |S'| &\leq 2|S| + \max\left\{1,\ \frac{|S|}{2(\log|S|+j+1)}\right\} \\ &= \max\left\{2|S|+1,\ 2\left(1+\frac{1}{2(\log|S|+j+1)}\right)|S|\right\} \end{aligned}$$

Thus, using complete induction similar to the proof of Claim 4.1 it follows that for every i-level edge e, the amount of 0-edges that are bridged by e is at most

$$2 \cdot 2^i \left(1+\frac{1}{2i}\right)^i \leq 2\sqrt{e} \cdot 2^i \ .$$

□

4.1 Randomized Hyperrings At the end of this section, we also address the issue of randomizing Hyperring updates. Suppose that we modify Add for the case that there is no bridge in the $k+2$-neighborhood of u in some ring R in the following way: u chooses a random number in $\{0,1\}$. If it is 0, then u includes itself in the ring of its left neighbor, and otherwise u includes itself in the ring of its right neighbor on top of R.

If each node now chooses $k_d = 15(\log d + 2)$, then the same invariant as in the deterministic case can be used to show that Proposition 4.1 also holds, with high probability, in the randomized case. One may ask whether for random updates k_d can be further reduced. However, k_d has to be $\Omega(\log d)$ to avoid a more than constant factor ring and edge distortion.

5 The Hyperring data structure

In this section we show how to use the Hyperring as a data structure. We assume that every node represents a data item and that data items are sorted on the 0-ring according to their names.

5.1 The Search operation Consider the Search algorithm in Figure 5. Since Search prefers edges of higher level and every $i+1$-edge bridges at most 3 i-edges for every i, we obtain the following facts.

Fact 5.1. *Any* Search *request moves along a sequence of edges of non-increasing level and uses at most two edges in each level.*

Combining these facts with Proposition 4.1, we immediately obtain the following lemma.

Lemma 5.1. *Every* Search *request uses a path of length at most $O(\log n)$.*

Furthermore, we get the following result.

Search(name):
A request P is sent to name with the help of the fact that the names of the nodes are in a sorted order on the 0-ring: Suppose that v is currently the node storing P. As long as $\mathsf{name} \notin [\mathsf{Name}(v), \mathsf{Name}(\mathrm{succ}_0(v)))$, forward P to the node w with maximum i so that $w = \mathrm{succ}_i(v)$ and $\mathsf{Name}(w) \leq \mathsf{name}$ (if there is no such w, choose the w among v's successors with maximum name). If this rule stops at a node w with $\mathsf{Name}(w) < \mathsf{name}$, then go to $\mathrm{succ}_0(w)$ and stop.

Figure 5: The Search algorithm.

Lemma 5.2. *The congestion caused by n* Search *requests with random destinations is $O(\log n)$, w.h.p.*

Proof. Fact 5.1 implies that every i-ring R can only receive requests from rings on top of it. Thus, it can only receive requests from its own nodes. Consider now an arbitrary node v in R. It is easy to check that only those requests will be sent to v whose destination is bridged by the i-edge e leaving v in R. From Proposition 4.1 we know that e bridges at most $4 \cdot 2^i$ nodes and that R consists of at most $3 \cdot n/2^i$ nodes. Since every node is the starting point of one request and every request has a random destination, the expected number of requests that want to reach v in R is at most $(4 \cdot 2^i/n) \cdot (3 \cdot n/2^i) = 12$. Combining this with the fact that every request only uses at most 2 edges in R (see Fact 5.1), the expected number of requests that traverse v in R is at most 24. Because every node participates in at most $\log n + O(1)$ levels, the overall expected number of search requests passing through v is $O(\log n)$. Using the fact that every request picks a random destination independently from other requests, it follows from the Chernoff bounds (e.g., [7]) that the congestion caused by Search is also $O(\log n)$ with high probability. □

This result can be extended to routing requests according to arbitrary permutations $\pi : V \to V$ on the set of nodes V. The problem here is that we cannot just use Valiant's trick of first routing the packets to random intermediate destinations before routing them to their correct destinations, because in an order-based network (where the nodes do not necessarily have uniformly distributed names as in DHT-based approaches) it is not obvious how to select a random destination. However, the following trick works instead:

A request P for name is sent to a random ring in each level, starting with level 0. In level 0, the source node s of P decides with probability 1/2 whether to keep P or to forward it to its closest predecessor in level 0 that belongs to the 1-ring s does not belong to. Once

P has reached the node it is supposed to reach in level $i-1$, it enters level i. In level i, the node v currently storing P decides with probability 1/2 whether to keep P or to forward it to its closest predecessor in level i that belongs to the $(i+1)$-ring v does not belong to. Once P reaches the highest possible level, it chooses a node in the ring belonging to this level uniformly at random as its intermediate destination.

Arguments based on the structural properties of the Hyperring suffice to show that this results in a near-uniform, random request distribution and creates a congestion of $O(\log n)$, w.h.p.

5.2 Insert and Delete operations The Insert(Name) and Delete(Name) operations are simply a combination of Search(Name) and Add resp. Remove. Hence, they achieve the following result.

THEOREM 5.1. *The work for an* Insert *and* Delete *operation is* $O(\log^3 n)$.

6 Conclusions

In this paper, we presented the first deterministic data structure suitable for distributed environments that can route Search requests with low congestion. The major drawback of this data structure is that Insert and Delete operations need $O(\log^3 n)$ work. It would be interesting to come up with deterministic data structures that only need $O(\log^2 n)$ or even $O(\log n)$ work for this. Techniques similar to transforming a hypercube into a butterfly may help.

References

[1] J. Aspnes and G. Shah. Skip graphs. In *Proc. of 14th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393, 2003.

[2] Y. Aumann and M. A. Bender. Fault tolerant data structures. In *Proc. of 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 580–589, 1996.

[3] B. Awerbuch and C. Scheideler. The Hyperring: A low-congestion deterministic data structure for distributed environments. Technical report, Johns Hopkins University, 2003. See http://www.cs.jhu.edu/∼scheideler.

[4] P. Beame and F. Fich. Optimal bounds for the predecessor problem. In *In Proc. of the 31st ACM Symposium on Theory of Computing (STOC)*, pages 295–304, 1999.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[6] E. W. Dijkstra. Self stabilization in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[7] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305–308, 1989/90.

[8] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USITS 2003*.

[9] N. J. Harvey and I. Munro. Brief Announcement: Deterministic SkipNet. To appear in *Proc. of the ACM/SIAM Symp. on Principles of Distributed Computing (PODC)*, 2003.

[10] P. Krishna. *Highly Scalable Data Balanced Distributed Search Structures.* Ph.D. thesis, University of Florida, 1995.

[11] X. Li and C. Plaxton. On name resolution in peer-to-peer networks. In *Proc. of the 2nd Workshop on Principles of Mobile Computing*, pages 82–89, 2002.

[12] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[13] X. Messeguer. Skip trees, an alternative data structure to skip lists in a concurrent approach. *Informatique Theorique et Applications* 31(3):251–269, 1997.

[14] I. Munro, T. Papadakis, and R. Sedgewick. Deterministic skip lists. In *Proc. of the ACM/SIAM Symposium on Discrete Mathematics (SODA)*, pages 367–375, 1992.

[15] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *SPAA Proceedings*, 2003.

[16] W. Paul, U. Vishkin, and H. Wagener. Parallel computation on 2-3 trees. *RAIRO – Theoretical Informatics* 17(4):397–404, 1983.

[17] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures*, pages 437–449, 1989.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM 2001*.

[19] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[20] C. Scheideler. *Universal Routing Strategies for Interconnection Networks.* Lecture Notes in Computer Science 1390. Spinger, 1998.

[21] D. Shasha and N. Goodman. Concurrent search structure algorithms. *TODS* 13(1):53–90, 1988.

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

[23] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *UCB Technical Report UCB/CSD-01-1141*, 2001.