Viceroy: Scalable Emulation of Butterfly Networks For Distributed Hash Tables*

Dahlia Malkhi† Moni Naor‡ David Ratajzcak§

November 11, 2003

Abstract

We propose and analyze a randomized variant of the Butterfly network that is amenable to decentralized construction and modification as nodes join and leave a network. The proposed networks have constant out-degree, and maintain a logarithmic diameter and nearly optimal congestion with high probability. In expectation, only a constant number of links are changed during a modification, and a logarithmic number with high probability. These properties make our construction particularly suitable for distributed hash tables (DHTs), which have been heavily promoted as a building block for internet-scale distributed services.

1 Introduction

Distributed hash tables (DHTs) provide a scalable solution to the classic problem of storing and sharing data among a large and dynamic set of participating servers ([7, 18, 24, 27, 29, 32, 33]). DHTs were originally suggested in Chord [32], and employ hashing to partition the key-value pairs among the participating servers so that only a small fraction of the keyspace must be moved to redistribute load when servers join and leave as participants. To further reduce the cost of such dynamism, DHTs do not require that changes in the key-server mapping be propagated to all current participants. Rather, each server maintains connection and routing information with a small number of other participating servers so that a client's lookup request for a key can be forwarded appropriately to the server managing the key. The set of connections between servers forms the *overlay routing network*, which is incrementally maintained as the system evolves.

In this paper, we focus only on the problem of setting an overlay routing network among a dynamic set of servers. Our construction achieves the following performance properties:

Low Congestion: The number of lookup requests routing through the system may be considerable. No server should receive a disproportionate amount of traffic when the requests are for random targets. More precisely, we define the *load* on a server to be the probability that a server is on a route with random source and destination. We wish the load to be balanced among all servers.

*A preliminary version of this paper appeared in PODC 2002.

†School of Computer Science and Engineering, The Hebrew University Jerusalem 91904, Israel; email: `dalia@cs.huji.ac.il`.

‡Incumbent of the Judith Kleeman Professorial Chair, Department of Computer Science and Applied Math, The Weizmann Institute of Science, Rehovot 76100 Israel; email: `naor@wisdom.weizmann.ac.il`. Research supported in part by an IST grant RAND-APX.

§Computer Science Division, University of California at Berkeley, Berkeley, CA 94720, USA; email: `dratajcz@cs.berkeley.edu`.

Incremental maintenance cost: The cost of adding or removing any server should be small, as measured by the amount of state that must be transferred, the number of servers impacted by the change, and the number of messages generated by the maintenance protocol. Specifically, each step (join or leave) should incur changes to the state of only a constant number of peers.

Degree: The number of connections on which a server can forward a lookup request should be small, as these connections must be removed (either immediately or lazily) and the routing information changed whenever either end leaves the network. We aim for a constant degree.

Dilation: The length of a forwarding path directly affects the latency of responding to a client's lookup request. An overlay should minimize the maximum actual path length of any request serviced by the lookup protocols (the dilation), as well as the expected path length for random requests. Our specific goal is to have logarithmic path lengths.

The task of constructing network topologies with low degree, low dilation, and balanced congestion has been the focus of considerable research for static environments [4, 31]. However, DHTs must work in a highly dynamic environment in which the size of the network is not known *a priori*, and where there are no permanent servers for maintaining either the hash function or the overlay network (all servers are assumed to be ephemeral). Hence, there must be a decentralized protocol, executed by joining and leaving servers, that incrementally maintains the structure of the system. Additionally, a joining server should be able to correctly execute this protocol while initially only having knowledge of a single, arbitrary participating server.

In this paper, we present the Viceroy network, a variant of Butterfly Networks yielding $O(1)$ out-degree, $O(\log n)$ dilation, and $O(\log^2 n/n)$ congestion, with high probability. Thus Viceroy achieves the same dilation and congestion as existing solutions [32, 33, 29] with fewer edges. We believe this is a significant achievement for two primary reasons: first, reducing the number of edges in the graph reduces the ambient traffic associated with pings and control information, and maintaining a constant degree network assuages concerns about the cost of too many open connections at servers; second, there are cases when it is desirable for a server to notify its outgoing and incoming connections that it is leaving, and thus the degree of the network directly relates to the total cost of joins and leaves. While it is a justifiable criticism that low degree networks are unsuitable for failure-prone environments, we note that an accepted way of handling fault tolerance (as in Chord [32] and in Pastry [29]) is not in the network topology itself but by "thickening" edges; fault tolerance of f simultaneous faults is achieved by replacing every edge by $f + 1$ redundant edges connecting close neighbors of the endpoints of the original edge. This is also similar to the "supernode" approach suggested in Lynch et al. [17], where nodes are actually replicated groups of servers. Section 6 discusses our application of these ideas.

Contributions of this work: We present a simple and efficient network construction that maintains constant degree networks in a dynamic environment. Routing is achieved in these networks in $O(\log n)$ hops and with nearly optimal congestion with high probability. Furthermore, servers joining and leaving the system induce only $O(\log n)$ messages and require only $O(1)$ servers to change their state. Viceroy is the first manifestation of a decentralized and dynamic network where all of these properties are achieved simultaneously. This addresses the first challenge in the paper by Ratnasamy et al. in [28].

A preliminary version of the Viceroy construction was presented in [18]. In the current version, we simplify the routing algorithm by making it almost purely greedy, and we present a more concise analysis. Several related works have appeared since the original Viceroy publication and are discussed in Section 7.

1.1 Related work

The proliferation of Internet-scale services and the advent of peer-to-peer applications has focused considerable attention on the resource lookup problem presented here. Many popular services, such as DNS, LDAP, and file-sharing services (such as Napster and Gnutella), rely on some partitioning of data across a dynamic network of servers. Recently, several schemes have been proposed that address the scalability and dynamic requirements of a world-wide lookup service.

The Chord lookup service [32] presents a solution that has greatly influenced our approach. Like in Chord, Viceroy employs consistent hashing [13] as the hash function mapping keys to servers, and uses a doubly-connected ring as the underlying routing structure. The algorithm of Plaxton et al. [25] was originally devised to route web queries to nearby caches, and has been employed in the Tapestry naming service [33] and Pastry [29]. The above works differ in the overlay routing network used for locating data efficiently. Compared with our scheme, all of the above methods have logarithmic out-degree overlays (variants of a hypercube graph), whereas Viceroy achieves the same dilation and congestion with only constant out-degree.

The work by Ratnasamy et al. [27] (CAN) dynamically maintains an approximation to a d-dimensional torus, for a chosen constant d. Their expected routing complexity is $O(dn^{(1/d)})$, compared with logarithmic in our construction.

Work on traditional data structures has produced some comparable results. SkipLists [26], for example, lend themselves to distributed construction and similarly employ randomization for efficient routing. SkipLists, however, have $O(n)$ congestion and only expected constant out-degree. Certain types of constant linkage-cost data structures are competitive with our scheme in every way except for congestion [14, 23]. Proposals for distributed data structures typically address the problem of incremental maintenance cost and state partitioning, but either require centralized control, or do not address congestion [10, 16].

There are other works that have looked at the problem of dynamic network construction with different emphases than ours. Pandurangan et al. [24] address the problem of dynamically constructing a low-degree, logarithmic diameter network under a probabilistic model of arrival and departure; however their construction does not provide a routing scheme, and their intended application is to disseminate queries to every server rather than to route to a particular server. Their method is also not fully distributed and employs a central server for newly connected nodes. Several works are concerned with building graphs resilient to various failure scenarios, including adversarial failures [15, 7, 5, 30].

There are additional issues pertaining to overlay networks beyond the scope of our work. One issue that arises is non-uniformity of server identifiers in the key-space due to either poor initial random choices or degradation due to non-independent server arrivals and departures. This phenomenon has been observed by others; in Naor et al. [21], the uniform dispersal of identities is termed "smoothness", and the performance of the *Distance Halving* network is shown with respect to smoothness. In CAN [27], this issue is addressed by requiring nodes to periodically change identifiers after departures. In Chord [32], a background overhaul process constantly re-adjusts random choices. Preliminary strides toward implicitly balancing overlay networks are made in Abraham et al. [1]. Another method of achieving smoothness is the virtual node technique introduced in Chord [32]. Using this technique, allowing each node to emulate up to $\log n$ virtual nodes achieves nearly uniform balancing at the expense of a logarithmic multiplicative factor increase in the out-degree of nodes. Though outside our current scope, it should be mentioned that these techniques can be combined with our construction to enhance load balance.

Another important issue in overlay routing networks is *locality-awareness*. In addition to the total number of hops in a route, it is desirable to keep the accumulated cost proportional to the cost of crossing the network. CAN [27] preserves locality by assigning the nodes identifiers that directly reflect their geographical location. Thus, a route between a starting point and an endpoint directly corresponds to the real distance that should be travelled. Toplus [9] also uses geographically-oriented node identifiers for locality. It uses IP addresses, and employs hierarchical routing, thus achieving roughly logarithmic dilation. Other works [25, 29, 33] randomize node identifiers, and achieve locality by having nodes select links based on proximity. Our work does not address locality, though recently, some of its concepts were applied by Abraham et al. in [2] to achieve a constant-degree locality-aware DHT.

2 System model and notations

The system consists of a dynamic set of nodes (servers). At any point in time, an existing node may choose to leave the system, or a new node may choose to join. In response to such a modification, the system reconfigures to include/exclude the joining/leaving node and establish or remove connections between nodes, leaving an overlay routing network among the new set of participating nodes.

Our construction makes use of the following notation. Each node u has two properties, its unique identifier $u.id$ and a level $u.level$. As an abuse of notation, we sometimes refer to $u.id$ simply as u. Identifiers are simply bit-strings. A prefix of length k of a bit string b is denoted by $b[k]$. The term $b[s-e]$ denotes a slice of b's bits, from the s'th through the e'th. The i'th bit of a string b is denoted by b_i. The length (in bits) of a string b is denoted by $|b|$.

In order to define an ordering among binary identifiers, we use a real metric. That is, let a binary string $\langle b_1, ..., b_k \rangle$ have the real value $\sum_{i=1..k} b_i/2^i$. This induces the real order among nodes and real distance metrics. For a node b, the node b' that is the minimal among greater nodes (if exists) is its unique SUCCESSOR; the unique PREDECESSOR is the inverse of SUCCESSOR.

2.1 Identifiers and levels

Nodes select identifiers as random bits strings. In practice, choosing strings of length 128 is sufficient to avoid collisions; in the unlikely case that two nodes select the same identifier, one is revoked. At any point in time, the *active* identifier $u.id$ of a node u with identifier id is the shortest unique prefix of id. Since node identifiers are selected at random upon arrival, and assuming that node departures are also independent of node identifiers, we have that the following property holds:

Property 2.1 [1]

In a system containing n nodes whose identifiers are selected as described above. Let $min\_prefix = \min_u |u.id|$, $max\_prefix = \max_u |u.id|$. Then w.h.p. $max\_prefix \leq 3\log n$ and $min\_prefix > \log n - \log\log n - O(1)$.

Nodes can thus estimate $\log n$ by examining the length of their own active identifiers. Nodes have, in addition to their identifier, a *level* attribute. Each node u chooses its level $u.level$ at random from among $1..k$, where $k = |u.id|$ is the length (in bits) of the node's identifier. Node u determines the length of its active identifier $u.id$ by examining its SUCCESSOR and its PREDECESSOR and taking the greatest common prefix.

[1] A proof of this simple property is provided for completeness in the Appendix.

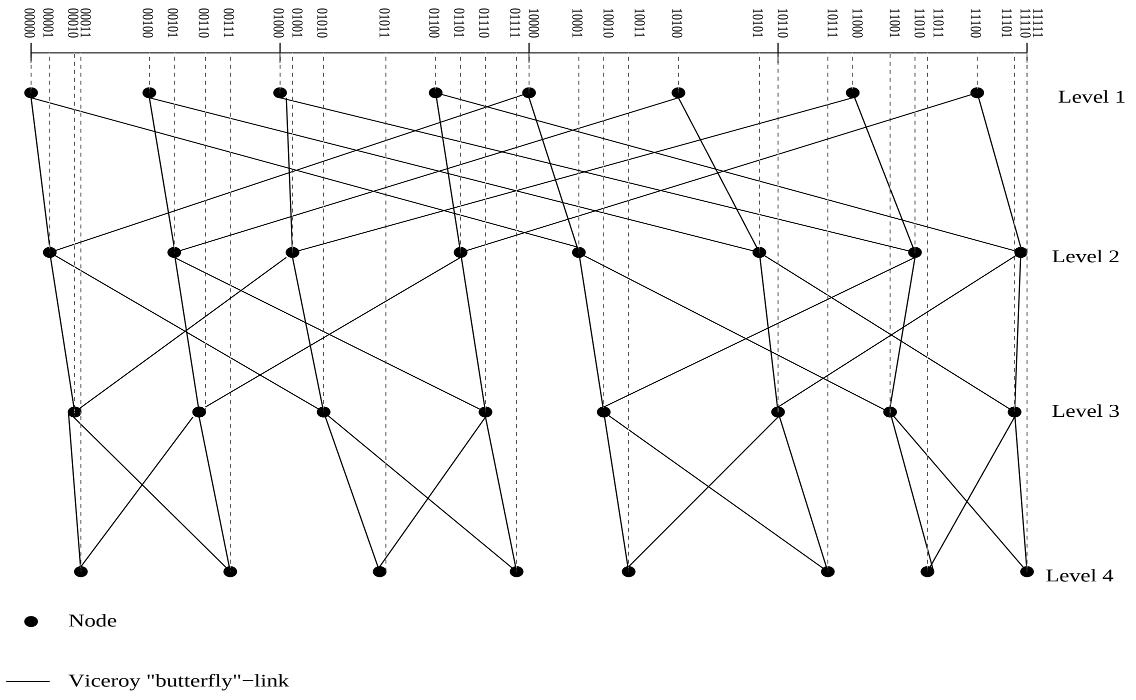

Figure 1: A typical Viceroy network of 32 nodes. Only Long and Medium-left links are shown for simplicity.

Several improvements on the above estimate of $\log n$ have been explored. First, the balancing technique of Abraham et al. [1] guarantees a logarithmic additive gap between max_prefix and min_prefix against adversarial arrivals and departures. Balancing to within a gap of 1 at a somewhat increased cost is discussed by Naor and Wieder [21]. Second, for randomized balancing, other techniques of Abraham et al. [1] and Adler et al. [3] balance the identifiers space so that there is a constant bound on the gap between max_prefix and min_prefix. Third, a node u can achieve a better estimate of $\log n$ than taking $|u.id|$ by the methods of Horovitz and Malkhi in [11] and of Manku in [19].

3 The Viceroy Network

Given $N = n \log n$ nodes whose identifiers and levels have been selected, the Viceroy network is completely determined as described below.

With $n \log n$ nodes in the network, we expect to have approximately $\log n$ network levels, each containing roughly n nodes. A node b of level $1 \leq i \leq \log n$ is responsible for routing to all identifiers $\langle b_1 b_2 ... b_{i-1} * \rangle$. It does so by pointing to certain level-$(i+1)$ nodes with identifiers $\langle b_1 b_2 ... b_{i-1} 0/1 * \rangle$. In addition, b is also linked to a level-$(i-1)$ node (if $i > 1$), and to its own successor and predecessor. In order to route to any target, we first descend down the levels to a node at level one, then move greedily to the target. A Viceroy network with 32 nodes is depicted in Figure 1.

More precisely, in the Viceroy network each node b of level $\ell = b.level$ has six out-going links as follows:

Short: A Successor link connects to SUCCESSOR(b); a Predecessor link connects to PREDECESSOR(b).

Medium: The are two Medium links. A Medium-left link connects to the closest level-$(\ell+1)$ node whose id matches $b.id[\ell]$ and is smaller than $b.id$. A Medium-right link connects to the closest level-$(\ell+1)$ node whose id matches $b.id[\ell]$ and is greater than $b.id$.

Long: A Long link connects to the closest level-$(\ell+1)$ node whose identifier matches $\langle b_1..b_{\ell-1}(1-b_\ell)*\rangle$.

Parent: A Parent link connects to the closest level-$(\ell-1)$ node.

A node also maintains information about all in-bound links.

The lookup primitive starts at a node x looking for the unique node whose id is a prefix of a target value y. The lookup procedure has two logical phases: First search for a node whose level is 1 ("root"), then follow links greedily. Figure 2 below describes precisely the lookup algorithm of target value y starting at node x.

```
LOOKUP($x, y$):

Initialization: Set $cur$ to $x$.

Proceed to root: If $cur.level = 1$ goto greedy-search phase.
    Else set $cur = cur$.Parent and repeat.

Greedy search: If $y$ is between $cur.id$ and $cur$.SUCCESSOR.$id$ in the real metric (or if $cur$.SUCCESSOR is
    empty), then return $cur$. Let $m$ be the link of $cur$ such that the distance from $m$ to $y$ is minimal.
    Then move to $m$ and repeat.
```

Figure 2: LOOKUP algorithm.

4 Analysis

The analysis below focuses on high-probability events. The expectations can be more easily derived, and are therefore stated without proof. Furthermore, the stress in the analysis is on simplicity, at the expense of non-tight constants. Below, we say that a property holds with high probability (w.h.p.) if it holds with probability $1-1/n^{1+\epsilon}$ for some $\epsilon > 0$.

4.1 Dilation

Dilation is a measure of the actual length of a lookup, as determined by the network structure and its lookup algorithm. The expected dilation of Viceroy is less than $3\log n$: At most $\log n$ steps to proceed to a root, another at most $\log n$ to fix $\log n - \log\log n$ prefix target bits, and finally $\log n$ Short steps to fix the remaining $\log\log n$ bits.

For the worst-case analysis of dilation, we focus on a route taken from a random initial node x in search of a random target Y which is found on node y (that is, $y.id$ is a prefix of Y). Let the 'proceed to root' phase of the lookup procedure consist of the sequence $x = u_0, u_1, ..., u_k$ of nodes. Let the 'greedy phase' of the lookup consist of the sequence $u_k = v_1, ..., v_m = y$ of nodes.

Lemma 4.1 *For every node u with level $u.level \le \log n - \log\log n$, the number of nodes between u and u.Medium-left (u.Medium-right), if it exists, is at most $6\log^2 n$ w.h.p.*

Proof: Starting from any node u, its Medium-left link is the first smaller node that has level $u.level + 1$. The link is k nodes away, where k is the first success in a sequence of Bernoulli trials, each succeeding if the node reached is the appropriate Medium link, i.e., has level $u.level + 1$. By Property 2.1, the probability of success is at least $1/3 \log n$. The probability that in a sequence of $6 \log^2 n$ steps there are no successes is $\left(1 - \frac{1}{3 \log n}\right)^{6 \log^2 n} < \frac{1}{n^2}$. (The case fo Medium-right is analogous.) ∎

The following lemma indicates that the greedy phase of the lookup procedure uses Medium/Long links that fix bits one by one, until within $\log^2 n$ nodes away from the target. At that point, the greedy routing may shift to use Short links.

Lemma 4.2 *In the 'greedy search' phase of the lookup of value Y from node x, let the j'th greedy step v_j, for $1 \leq j \leq m$, be such that v_j is more than $O(\log^2 n)$ nodes away from y. Then w.h.p. node v_j is reached over a Medium or Long link, and hence satisfies $v_j.level = j$ and $v_j[j] = Y[j]$.*

Proof: Let $1 \leq j \leq \log n - 2 \log \log n - \log(3 + \epsilon)$. The proof is by induction. Suppose that the lemma holds for the $(k-1)$'st step, v_{k-1}. We make use of the fact from Property 2.1 above, that each node chooses a level at most $3 \log n$. We have that the probability that no node has prefix $Y[k]$ and level k is at most

$$\left(1 - \frac{1}{2^k 3 \log n}\right)^n \leq e^{-\left(\frac{(3+\epsilon)\log^2 n}{3 \log n}\right)} \leq \frac{1}{n^{1+\epsilon/3}} .$$

Depending on whether the k'th bit of Y matches that of v_{k-1} already or not, we get that an appropriate link, either Long or Medium, of v_{k-1} exists and satisfies the lemma. Thus, if the chosen link is Long or Medium, the induction holds.

Unfortunately, the proof is not complete yet, as we need to show that the appropriate link is the closest to target. Otherwise, a greedy route might not select the needed link, if one of the other links of v_{k-1} connects closer to Y. In particular, it could choose a Short link. We first note that in that case, since a link that has k common prefix bits with Y exists, any link closer to Y must have at least k common prefix bits as well. If the chosen link was a Medium or Long link, then it connects to a level k node with an appropriate prefix, and the induction holds. Otherwise, the chosen link is a Short link. In this case, note that by Lemma 4.1, a Medium link is at most $O(\log^2 n)$ nodes away from v_{k-1}. Since the target y must reside between the chosen Short link and the Medium link in that direction, we reach a node v_k that is $O(\log^2 n)$ nodes away from the target, and the lemma holds.

∎

Lemma 4.3 *Let v be a node that is $O(\log^2 n)$ nodes away from the target y. Then w.h.p., within $O(\log n)$ greedy steps the target y is reached from v.*

Proof: The analysis proceeds in two logical parts. The first part, called *hopping*, uses Medium links to hop over several nodes at a time. We will show that there is a logarithmic number of such hops. Intuitively, this is due to the fact that Medium links hop over $O(\log n)$ nodes in expectation. The second part uses Short links to step to the target. We will show that this part is also logarithmic.

For the first part, let the direction with shortest distance from v to y be the right-direction (similarly, left). Starting from any node u, its Medium-right (Medium-left) link is k nodes away, where k is the first success in a sequence of Bernoulli trials, each succeeding when a node is reached that is appropriate as a

Medium link, i.e., has level $u.level + 1$. The second hop is the second such success in the sequence. And so on. Each success in the sequence is counted as one hop.

Starting from a node u, and considering each successive node until a hop, the probability that a node is the next hop is at most $1/(\log n - \log\log n - O(1)) \leq 2/\log n$ by Property 2.1. The probability that in a sequence of $\log^2 n$ steps there are more than $c \log n$ hops is therefore the probability that in the sequence above of $\log^2 n$ Bernoulli trials there are more than $c \log n$ successes. This probability can be bounded, for an appropriate choice of c, by:

$$\binom{\log^2 n}{c\log n}\left(\frac{2}{\log n}\right)^{c\log n} \leq \left(\frac{\log^2 ne2}{c\log n \times \log n}\right)^{c\log n} \leq \frac{1}{n^2} .$$

For the second part, let w be the last hop in the hopping part. I.e., from w to the target y we have to use Successor (Predecessor) links only. This means that for each node z from w to y, there is no suitable Medium-right (Medium-left) link between z and y. Suppose that the number of nodes from z to y is greater than $\log n$. Denote by B_z the event that there is no suitable Medium-right (Medium-left) link for z between z and y. The event B_z occurs if no node between z and y has level $z.level + 1$. The probability of B_z is bounded by $\left(1 - \frac{1}{2\log n}\right)^{\log n} \leq e^{-1/2}$. Therefore, the probability that B_z holds for more than $c' \log n$ nodes is at most $\left(e^{-1/2}\right)^{c' \log n} \leq 1/n^2$, for appropriate choice of $c' > 0$. In total, there are $O(\log n)$ nodes in the second part w.h.p.

Putting the two parts together, we get that by following greedy routing, hopping over $\log^2 n$ nodes takes w.h.p. $O(logn)$ steps. □

Putting all of the above together, we get that the dilation of the network is $O(\log(n))$ w.h.p:

Theorem 4.4 *The total length of a route from x to y is $O(\log n)$ w.h.p.*

Proof: First, note that the algorithm terminates and finds the target: The 'proceed to root' phase finds a level-1 node, and the 'greedy search' phase reduces the distance to the target with every step; by our real metric, a node with longest matching prefix to y is found, and this is indeed the target.

As for dilation, by Property 2.1, the level selected by x is at most $\log n + \log\log n + O(1)$. Hence, The 'proceed to root' phase of the lookup procedure takes at most $2\log n$ steps, reaching a node whose level is one. By Lemma 4.2, within $\log n$ steps a node u is reached that is at most $\log^2 n$ nodes away from y. Finally, using $O(\log n)$ more steps, Lemma 4.3 assures that the target is reached. Thus, in total $O(\log n)$ are needed to reach the target. □

4.2 Load

The length of routes is $O(\log n)$, and hence, given the almost symmetrical nature of our construction, it is not difficult to show that the expected load on every node u is $O(\log n/n)$. This is optimal for logarithmic dilation networks. Furthermore, methods for guaranteeing this load in the worst case are discussed in Section 6.

For the worst case analysis, we once again focus on a route initiated at a random starting point x and a random search target Y found on a node y. Recall that for a node identifier b, the clause $b[s - e]$ denotes the slice of bits from the s'th to the e'th.

Lemma 4.5 *Denote by $\rho = \log n - 2\log\log n - \log(3+\epsilon)$. Then for every node u, its Long, Medium, and DownLevel links are determined up to their ρ's bit w.h.p. Specifically, we have u.DownLevel$[\rho]$ = u.Medium-left$[\rho]$ = u.Medium-right$[\rho]$ = u$[\rho]$, and u.Long$[\rho]$ matches u$[\rho]$ except for the u.level'th bit.*

Proof: The probability that any link of u does not satisfy the conditions of the lemma is given by the probability that no node v exists at the desired level and whose first ρ bits match the specified prefix. This probability is at most $\left(1 - \frac{1}{2^{\rho}\cdot 3\log(n)}\right)^n < \frac{1}{n^{1+\epsilon/3}}$. ∎

Theorem 4.6 *The load on every node is* $\log^2 n/n$ *w.h.p.*

Proof: Let a node u participate in the 'proceed to root' phase. Then by Lemma 4.5, $u[\rho] = x[\rho]$.

Next, let a node u be on the 'greedy phase', such that u is more than $\log^2 n$ nodes away from y. Let $\ell = u.level$. Our proof will show that w.h.p. $u[\ell] = y[\ell]$ and $u[(\ell+1)-\rho] = x[(\ell+1)-\rho]$. Thus, all nodes on route that are $\log^2 n$ away from target are determined up to their ρ's bit.

The claim concerning $u[\ell]$ follows from Lemma 4.2. The claim concerning $u[(\ell+1)-\rho]$ is proved by induction on the sequence of routing steps as follows. Suppose that for some node u' on the route the claim holds. Let u be the next routing step taken after u'. If u is still $\log^2 n$ away from the target, then as shown in Lemma 4.2, u is either a Medium or Long link of u'. Denote $\ell = u'.level$. By hypothesis, $u'[(\ell+1)-\rho] = x[(\ell+1)-\rho]$. Therefore, if a node that maintains the induction exists, it is the closest to u'. By Lemma 4.5, such a node exists w.h.p., and the induction follows.

Summarizing the above, in all routing steps until $\log^2 n$ nodes away from y, the first ρ bits are fixed by choice of x and y. Since all identifiers are of length at most $\log n + \log\log n + O(1)$, there are at most $2^{2\log\log n + O(1)} = O(\log^2 n)$ choices of x and y for which u participates in the route in this way.

In addition, u may participate in the Short steps of $\log^2 n$ vicinity nodes. In total, the load on u is $O(\log^2 n/n)$. ∎

4.3 Degree

The out-degree of the network is precisely six. Hence, the in-degree is expected constant. Furthermore, methods for guaranteeing this in-degree in the worst case are discussed in Section 6.

Theorem 4.7 *Every node u has in-degree* $O(\log n)$ *w.h.p.*

Proof: Lemma 4.5 assures that any node with a Medium, Long and Parent links connected to u is determined up to its ρ'th bit, w.h.p. The probability that more than $2ce\log n$ nodes match a specific prefix of length $\log n - 2\log\log n - \log c$ at a specific level is bounded by

$$\binom{n}{2ce\log n}\left(\frac{c\log^2 n}{n}\frac{1}{(\log n)/2}\right)^{2ce\log n} \le \left(\frac{ne}{2ce\log n}\frac{2c\log n}{2n\log n}\right)^{2ce\log n} \le \frac{1}{n^2}\ .$$

Of these, w.h.p. $O(\log n)$ have a specific level. Hence, w.h.p the in-degree is at most $O(\log n)$. ∎

5 Distributed maintenance

In this section we describe how to maintain the Viceroy network in a distributed manner to accommodate node arrival and departure. This entails specifying a JOIN and a LEAVE algorithms, depicted in Figure 3 and Figure 4, respectively.

JOIN:

Choose identifier: Select at random 128 bits $x = \langle x_1, x_2, ..., x_{128} \rangle$ for identifer.

Set Short links: Invoke LOOKUP(x) at an initial contact point, and let x' be the resulting node. Insert x between x' and x'.Successor, setting Predecessor and Successor links appropriately.

Choose level: Let k be the maximal number of matching prefix bits between x and either Successor or Predecessor. Choose $x.level$ between $1..k$ at random.

Set Medium links: Denote $p = x_1, x_2, ..., x_{x.level+1}$. If Successor has a prefix matching p and level $x.level+1$, set x.Medium-right link to it. Otherwise, if Successor has a prefix matching p, move to Successor and repeat.

Likewise, starting from x, if Predecessor has a prefix matching p and level $x.level+1$, set x.Medium-left link to it. Otherwise, if Predecessor has a prefix matching p, move to Predecessor and repeat.

Set the Long link: Denote $p = x_1, x_2, ..., x_{x.level}, 1-x_{x.level+1}$. Invoke LOOKUP($x$) to find a node matching p at level $x.level+1$; this implies stopping the greedy-routing phase of LOOKUP after the $(level+1)$'st step. Set x.Long to the node reached.

Set the Parent link: If Successor has level $x.level - 1$, set x.Parent to it. Otherwise, move to Successor and repeat.

Set in-bound links: Denote $p = x_1, x_2, ..., x_{x.level}$.

First, set in-bound Medium-left links: So long as Successor y has a prefix p and a level different from $x.level$, if $y.level = x.level - 1$, set y.Medium-left to x; in any case, move to Successor and repeat.

Setting in-bound Medium-right links is done analogously by following Predecessor links.

Now we get to inbound Long links. So long as Successor y has a prefix p and level different from $x.level$, move to Succesor and repeat. If a node y is reached with a prefix p and level $x.level$, then if any of its in-bound Long links closer to x than to y, change them to x. Do the analogous procedure following Predeceesor links.

Finally, we set incoming Parent links. So long as Predecessor y has level other than $x.level$, if $y.level - 1 = x.level$ set y.Parent to x, and in any case, move to Predecessor and repeat.

Figure 3: JOIN algorithm.

LEAVE: Update x.Successor and x.Predecessor so as to remove x from in-between them. Re-assign all in-bound links using procedures similar to the ones in JOIN.

Figure 4: LEAVE algorithm.

Theorem 5.1 *A* JOIN *operation by a new node x incurs an expected $O(\log n)$ number of messages, and $O(\log^2 n)$ messages w.h.p. The expected number of nodes that change their state as a result of x's join is constant, and w.h.p. is $O(\log n)$.*

A LEAVE *operation by a node x incurs $O(\log n)$ number of messages, and $O(\log^3 n)$ messages w.h.p. The expected number of nodes that change their state as a result of x's departure is constant, and w.h.p. is $O(\log n)$.*

Proof: (Sketch) During a join procedure, a constant number of out-links are sought. As the lookup of any target takes $O(\log n)$ messages, the main communication cost to examine is the cost of single-stepping the Short links in order to find the Medium/Long links. Here, analysis similar to that in Lemma 4.1 shows that the expected number of such steps is $O(\log n)$, and w.h.p. it is $O(\log^2 n)$. Likewise, the analysis in Theorem 4.7 shows that the new node will have w.h.p. $O(\log n)$ in-links, and hence, will incur state change in $O(\log n)$ nodes.

A LEAVE operation by a node x incurs change of state in each one of its incoming links. Each of these changes entails a procedure of finding replacement links, similar to a joining procedure, whose complexity is given above. As there is an expected constant number of incoming links, there is an expected constant number of incurred link-search procedures. Likewise, there is a logarithmic number of link-searches w.h.p.
$\square$

6 Bounding Indegrees - The Bucket Solution

So far we have concentrated on fixing the outdegree of nodes and creating a sparse network. This obviously yields small *average* indegree. However, the largest indegree in the network might still be as large as $\log n$ (w.h.p.) and thus a single server leaving would cause $\log n$ linkage changes. To combat this we add another background process to the system which we call "buckets." Unfortunately, the details of the bucket mechanism are more involved than the rest of Viceroy so we only sketch its operation here.

Our approach is to improve the identifier and level selection procedures so that in stretches of length $(\log n)/n$ we will have a constant number of servers (between c_1 and c_2 for some fixed c_1 and c_2; we actually take c_1 to be 1) from each level. We achieve this by maintaining a distributed coordination mechanism between contiguous, non-overlapping sets of servers on the ring, called *buckets*, consisting of $\Theta(\log n)$ servers each. Inside each bucket, we maintain a simple ring (which mostly overlaps the general ring, except that the end points of the bucket are also connected). In more details, the buckets are maintained so that several properties hold:

Size: The size of a bucket is always $\Theta(\log n)$, that is whenever the size of a bucket drops below $\log n$, it is merged with a neighbor bucket, and whenever the size exceeds $c \log n$, for a fixed $c \geq 2$, the bucket is split into two. Such merging and splitting might involve reassignment of levels to *all* members of the bucket. Within the bucket, a precise estimate of $\log n$ is maintained. Furthermore, if when merging with an adjacent bucket the total number of servers in the two merged buckets is greater than $(2c+2)/3$, then instead of a single bucket, two buckets with equal number of servers are created.

Diversity: The levels inside a bucket are *not* assigned at random, rather it is assured that from each level in $[1 \ldots \log n]$ there is *at least* one member and *no more* than c members. This assures that the indegree of any server is bounded by $2c$. To maintain this last property it might be necessary to reassign a new level to some remaining server in the bucket when a server leaves the system (and the bucket).

Overhead: Consider first what happens when a server joins or leaves the system and no merging or splitting of buckets occurs. In case of a join, the server should find a level with fewer than c_2 servers in the bucket and choose it as its level. Such a level must exists if $c_2 \leq c$. When a server of level j leaves the system, in case the number of servers of level j in the bucket drops below c_1, then another server with a different level, of which there are at least $c_1 + 1$ representatives, must be moved to level j; such a server must exist if $c_1 = 1$ and there are at least $\log n$ servers in the bucket and $\log n$ levels. Hence the overhead from such assignments is $O(1)$. Consider now the amount of amortized work that merging and splitting can require. First note that the amount of work merging and splitting may cause is proportional to $c \log n$. The conditions on splitting and merging imply that in a 'new' bucket, i.e. one that is the result of a merge or a split, the number of servers divided by $\log n$ is between the following lower and upper bounds:

Lower Bound: $\min((c+1)/3, c/2)$ where the first term comes the case a merge has resulted in two equal sized buckets and the second from a split

Upper Bound: and $\max((2c-2)/3, c/2)$ (the first when the result is a single bucket and the second from a split).

Therefore, the number of joins or leaves that must happen in this bucket until another split or merge is preformed is at least $\min((c-2)/3, c/2 - 1) \log n = (c-2)/3 \log n$. Therefore the amortized cost of the rearrangements is proportional to

$$\frac{c \log n}{(c-2)/3 \log n} = \frac{3c}{c-2},$$

so the amortized complexity is $O(1)$.

Finally, buckets are also natural units for maintaining replicated data, e.g., on routing information as well as the data itself, in order to achieve fault tolerance.

7 Postscript

Several related papers on DHT constructions have been published since the preliminary publication of this work [18], addressing our constant degree goal. Most of them emulate the De Bruijn graph [6]. These include the *Distance Halving* network [21, 22], the D2B network of [8], the *Koorde* network [12] and the De Bruijn construction of Abraham el al. in [1]. Symphony [20] uses a constant number, k, of links per node to yield $O(\log^2 n/k)$ expected dilation.

In the works of Naor and Wieder [21] and Abraham et al. [1], a process of smoothing (balancing) during joins and leaves guarantees, in addition to constant out-degree, also a worst case constant in-degree, as well as better lookup load balancing.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proceedings of the International Parallel and Distributed Processing Symposium (IDPDS 2003)*, 2003.

[2] I. Abraham, D. Malkhi, and O. Dobzinski. LAND: Stretch $(1+\varepsilon)$ locality aware networks for DHTs. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA04)*, 2004.

[3] M. Adler, E. Halperin, R. Karp, and V. Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. In *Proceedings of 35th ACM Symposium on Theory of Computing (STOC)*, 2003.

[4] T.H̃. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press, 1990.

[5] M. Datar. Butterflies and peer-to-peer networks. In *ESA*, 2002.

[6] N. G. de Bruijn. A combinatorial problem. *Konink. Nederl. Akad. Wetersh. Verh. Afd. Natuurk. Eerste Reelss*, (A49):758–764, 1946.

[7] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.

[8] P. Fraigniaud and P. Gauron. The content-addressable network D2B. Technical Report 1349, LRI, Univ. Paris-Sud, France, 2003.

[9] L. Garces-Erice, K.W. Ross, E.W. Biersack, P. Felber, and G. Urvoy-Keller. Topology-centric look-up service. In *Proceedings of the 5th International Workshop on Networked Group Communications (NGC'03)*, 2003.

[10] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, distributed data structures for internet service construction. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI 2000)*, 2000.

[11] K. Horovitz and D. Malkhi. Estimating network size from local information. *Information Processing Letters*, 2003. To appear.

[12] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[13] D. Karger, E. Lehman, F. T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 1997.

[14] T. Lai and D. Wood. Adaptive heuristics for binary search trees and constant linkage cost. In *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 72–77, 1991.

[15] M.-J. Lin, K. Marzullo, and S. Masini. Gossip versus deterministically constrained flooding on small networks. In *Proceedings of the 14th International Conference on Distributed Computing*, pages 253–267, 2000.

[16] W. Litwin, M.A. Neimat, and D. A. Schneider. LH* - a scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.

[17] N. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in distributed hash tables. In *Proceedings of the International Peer-to-Peer Symposium*, 2002.

[18] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02)*, pages 183–192, 2002.

[19] G.S. Manku. Low degree networks for distributed hashing. In *Proceedings of 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, 2003.

[20] G.S Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[21] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '03)*, 2003.

[22] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[23] T. Ottmann and D. Wood. Updating binary trees with constant linkage cost. *International Journal of Foundations of Computer Science*, 3:479–501, 1992.

[24] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter p2p networks. In *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, 2001.

[25] C. Plaxton, R. Rajaram, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 97)*, pages 311–320, 1997.

[26] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[27] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, 2001.

[28] S. Ratnasamy, I. Stoica, and S. Shenker. Routing algorithms for dhts: Some open questions. In *Proceedings of First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, 2002.

[29] 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)*, pages 329–350, 2001.

[30] J. Saia, A. Fiat, S. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, 2002.

[31] H. J. Siegel. Interconnection networks for simd machines. *Computer*, 12(6):57–65, 1979.

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

[33] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, , and John Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

Proof: [Of Property 2.1 For the first inequality, the probability that there exist two nodes whose identifers matching in $(3+\varepsilon)\log n$ bits is bounded by: $\binom{n}{2}\left(1/2^{(3+\varepsilon)\log n}\right) \leq 1/n^(1+\varepsilon)$. Hence, w.h.p. have identifiers of length $3\log n$ or less.

As for the second inequality, the probability that there exists a node whose identifier matches no other identifier in $\log n \log\log n - \log c$ bits is bounded by: $n\left(1-\frac{1}{2^{\log n - \log\log n - \log c}}\right)^n \leq ne^{-c\log n} \leq 1/n^{c-1}$. Hence, the length of every identifier is at least $\log n - \log\log n - \log c$ w.h.p. □