A Randomized ID Selection Algorithm for Peer-to-Peer Networks

Gurmeet Singh Manku

Abstract: *We present a low-cost, decentralized algorithm for the problem of selecting host IDs in peer-to-peer systems. The problem arises in the context of distributed hash tables (DHTs). Our algorithm is the first one with the following properties: (a) both arrivals and departures of hosts are handled, (b) partition balance ratio is guaranteed to be at most 4, (c) at most one existing ID is re-assigned in response to departures, and (d) the expected cost is $\Theta(R + \log n)$ messages, where n denotes the current number of participants, and R denotes the cost of routing one message in the DHT. Further, our algorithm is independent of the overlay routing topology, making it applicable to all DHTs. Our algorithm improves upon existing algorithms which have one or more of the following drawbacks: they require $\Theta(R \log n)$ messages, do not handle host departures, or re-assign the IDs of $O(\log n)$ existing hosts in response to a departure.*

Our ID selection algorithm enables emulation of a variety of deterministic and randomized families of interconnection topologies, in a straightforward fashion. Finally, an extension to our algorithm allows improvement of the partition balance ratio to $(1 + \epsilon)$, where $\epsilon > 0$, albeit at the cost of re-assigning the IDs of $O(\frac{1}{\epsilon})$ hosts upon arrival/departure. We believe that ours is the first algorithm that allows such fine-tuning.

1 Introduction

Distributed hash tables (DHTs) in peer-to-peer systems are managed by a large number of hosts as follows. Imagine $[0, 1)$ as a circle with unit perimeter. Each host that joins the system is assigned an ID in $[0, 1)$. At any time, the set of IDs *partitions* $[0, 1)$ into disjoint sub-intervals, managed by one host each. Each host is connected with its successor and its predecessor with TCP links, thereby forming a "ring". Each host also makes TCP links with a few other hosts, as a function of its own ID. Taken together, these TCP links form the *routing network*. In this paper, we treat the routing network as a black-box with the following property: Using the network, it is possible to identify the "manager" of a randomly-chosen point in $[0, 1)$ by paying a cost of R messages, with high probability[1]. $R = \Theta(\log n)$ for Chord [SMK+01], Pastry [RD01], Tapestry [ZHS+04] and Viceroy [MNR02], w.h.p. However, $R = \Theta(\log n / \log \log n)$ w.h.p. for high-degree de Bruijn networks, as has been observed by several groups [KK03, FG03, AAA+03, NW03, LKRG03], Kleinberg-style randomized butterflies [Man03], Symphony [MBR03] and several other randomized networks [MNW04].

ID Selection Problem: Upon arrival, a new host has to select an ID for itself, in order to join the ring (and subsequently, to establish TCP connections with other hosts in accordance with the rules for maintaining the routing network). We assume that the new host knows one existing member of the ring at the outset. At any instant, any member of the ring can ascertain the IDs of k adjacent hosts in the ring by paying a cost of $2k$ messages, or it can identify the manager of a random number in $[0, 1)$ by paying a cost of R messages.

Design Goals: The ID selection algorithm should be simple, decentralized and low-cost in terms of number of messages. The variation in partition-sizes should be minimal. The number of ID re-assignments of existing hosts in response to arrivals and departures should be minimal.

Previous Work: Early designs for DHTs allowed each host to independently choose a number in $[0, 1)$ uniformly at random [SMK+01, ZHS+04, RD01, RFHK01, FG03, KK03, MBR03, MNR02, HJS+03]. This results in $\sigma = \Theta(\log^2 n)$ [NW03], where σ denotes the *partition balance ratio*, defined as the ratio between the largest and the smallest partition sizes. If each host chooses a random number in $[0, 1)$ and *splits* the partition the number falls into, σ diminishes to $\Theta(\log n)$ [AHKV03, NW03]. Further improvement is possible. If each host creates $\Theta(\log n)$ *virtual IDs* [DKK+01], σ reduces to $\Theta(1)$. However, the number of overlay connections per host gets amplified by a factor of $\Theta(\log n)$ – this is costly because higher degree overlay networks require more resources for state-maintenance.

Two different approaches for ID selection have recently been proposed, each of which guarantees $\sigma = \Theta(1)$ with only one ID per host.

[1] By "with high probability" (w.h.p.), we mean "with probability at least $1 - O(n^{-\alpha})$ for some constant $\alpha > 0$, for a system with n hosts".

Algorithm	*Overlay Independent*	*Height Jitter*	σ	*Message Cost*	*Handles Departures*	*Number of Re-assignments*
Random Binary Tree	Yes	$\Theta(\log\log n)$	$\Theta(\log n)$	R	No	–
Adler et al [AHKV03]	No	$\Theta(1)$	$\Theta(1)$	$\Theta(R+\log n)$	No	–
Naor and Wieder [NW03]	Yes	$\Theta(1)$	$\Theta(1)$	$\Theta(R\log n)$	(Yes)	(Unknown)
Abraham et al [AAA$^+$03]	Yes	$\Theta(1)$	$\Theta(1)$	$\Theta(R\log n)$	No	–
Karger and Ruhl [KR03]	Yes	$\Theta(1)$	$\Theta(1)$	$\Theta(R\log n)$	Yes	$O(\log n)$
New Algorithms	**Yes**	**3**	**4**	$\Theta(R+\log n)$	**Yes**	**1**
	Yes	**2**	**2**	$\Theta(R+\log n)$	**Yes**	**2**
	Yes	**2**	$(1+\epsilon)$	$\Theta(R+\frac{1}{\epsilon^2}\log n)$	**Yes**	$2/\epsilon$

Table 1: R denotes the average number of messages required by the overlay routing layer. Typically $R=\Theta(\log n)$ or $R=\Theta(\log n/\log\log n)$, w.h.p. Height jitter denotes the number of different levels to which leaf nodes belong. σ denotes the ratio of the largest partition to the smallest partition. For [AHKV03], $R=\Theta(\log n)$ since the scheme is tied to a specific overlay routing topology (the hypercube).

The first approach [NW03, AAA$^+$03, KR03] is overlay-independent while the second is overlay-dependent [AHKV03]. Naor and Wieder [NW03] and Abraham et al [AAA$^+$03] proposed that a new host should choose $\Theta(\log n)$ random points from $[0,1)$, identify the managers of these points and split the largest manager into two. Karger and Ruhl [KR03] proposed a variation on the idea that supports departures as well. However, their variation necessitates $O(\log n)$ hosts to change their IDs in response to both arrivals and departures – a costly operation. Adler et al [AHKV03] analyzed an overlay-dependent scheme that is specific to hypercubes. The idea is to identify the manager of a random point in $[0,1)$, probe other managers it has established overlay connections with, and to split the largest of these managers into two. A scheme for handling departures exists, but it has not yielded to formal analysis yet. Features of our algorithm (see Table 1):

i) Generality: Our algorithm is independent of the overlay routing topology.

ii) Low cost: Our algorithm requires $\Theta(R+\log n)$ messages, which is cheaper than other overlay-independent schemes, each of which requires $\Theta(R\log n)$ messages. The overlay-dependent scheme in [AHKV03] requires $\Theta(\log n)$ messages but does not handle host departures.

iii) Host departures: [AHKV03] and [AAA$^+$03] do not handle departures. A full exposition of the departure-scheme in [NW03] has been deferred to the final version of their paper. Departures entail only one re-assignment in our scheme, for $\sigma=4$. The scheme in [KR03] requires $O(\log n)$ re-assignments for both arrivals and departures.

iv) Small partition balance ratios: We guarantee $\sigma=4$ w.h.p. without any re-assignment of existing IDs. We guarantee $\sigma=2$ by requiring at most one re-assignment. The idea can be extended to obtain smaller values of σ at the cost of changing the IDs of more hosts upon arrivals. Ours is the first algorithm that allows σ to be fine-tuned in such a fashion, with provable error guarantees. Existing schemes involve binary trees and cannot obtain $\sigma<2$.

v) Spectrum of schemes: The description of our algorithm permits a simple generalization yields a variety of schemes ranging from completely centralized (perfectly balanced binary trees) to completely decentralized (random binary trees).

vi) Overlay construction: Our algorithm enables *emulation* of various families of deterministic and randomized inter-connection network topologies. A host makes exactly *three sets* of links with other hosts.

vii) Estimation of n: Our algorithm permits a simple scheme for estimating n, the total number of participants currently in the system. In fact, a factor-4 estimate can be derived from a host's own ID. Sharper estimates can be obtained by modifying the arrivals/departure protocols at no extra cost. The resulting estimation scheme diminishes the number of sets of links required for emulation to only two.

2 The Algorithm

We present two ID selection algorithms. The first algorithm handles only host arrivals. The second algorithm (§4) handles host departures too, but is involved. The motivation for separating the two is that exposition of the key ideas (in §2.2, §2.4 and §3) is *cleaner*. Moreover, we feel that the complexity in the second algorithm is not inherent in the ID selection problem per se – it is an artifact of the tools we have used for analysis. Even without deletions, the algorithm is an improvement over existing overlay-independent ID selection algorithms, each of which requires $\Theta(R \log n)$ messages. Moreover, σ can be fine-tuned to $(1+\epsilon)$, as shown in §2.4, a feature unique to our algorithm.

Each host in the system is assigned an ID in $[0, 1)$, which can be visualized as a circle with unit perimeter. There is always a host with ID 0. Point $x \in [0, 1)$ will be managed by the host with the largest ID that does not exceed x. Each host is connected to its successor and its predecessor along the circle. Hosts make additional connections among themselves as a function of their IDs, thereby constructing an overlay routing network. We treat the overlay as a black-box which allows any member of the ring to identify the manager of any point $x \in [0, 1)$ in R messages, w.h.p. Typically, with n hosts in the system, $R = \Theta(\log n)$ or $R = \Theta(\log n / \log \log n)$.

2.1 Addition in a Binary Tree

Consider a binary tree with each internal node having degree two. The left and right branches emanating from an internal node are labeled 0 and 1 respectively. Leaf nodes have IDs associated with them. The sequence of 0s and 1s along the path from the root to a leaf node, treated as the binary expansion of a fraction in $[0, 1)$, constitutes the ID of that leaf. The *level* of a leaf node is the length of the path from the root. The root is at level 0 and there are at most 2^ℓ leaves at level ℓ.

Active Nodes: We mark a small fraction of internal nodes as *active*. Let n denote the number of leaf nodes. With $n = 2$, there is only one internal node, the root node, which is marked active. As more nodes arrive, the set of active nodes changes. At all times, we maintain the property that for every leaf node, exactly one internal node along the path from that leaf node to the root is active. Thus the set of active nodes constitutes a *frontier* such that sub-trees hanging below active nodes partition leaf nodes into disjoint groups.

Addition Algorithm: Upon arrival, a host selects its ID as follows: (a) *Random walk down the tree to reach leaf node* **r**: Start at the root. At each step, choose between the two children of an internal node, uniformly at random. Let **r** denote the leaf node reached. (b) *Perfect insertion in the sub-tree rooted at* **a**, *the active ancestor of* **r**: Imagine that each internal node is labeled with the number of leaf nodes in the sub-tree rooted at that node. Starting at **a**, we repeatedly move to the child with fewer leaves below it, breaking ties arbitrarily. We split the leaf node we reach, into two.

Having increased the number of leaf nodes by one, we check whether **a** should continue to be active, as follows. Let ϕ denote a monotonically non-decreasing function such that $\phi(\ell) \in [0, \ell]$ for non-negative integer ℓ. We maintain the invariant that whenever a new node is inserted at level ℓ under an active node **a**, then **a** is guaranteed to be at level $\phi(\ell)$. Keeping this invariant in mind, we mark **a** as non-active and we mark both its children as active iff two conditions are satisfied:

(i) No more nodes can be added at level ℓ in the sub-tree rooted at **a**, where ℓ is the level of the newly inserted node.

(ii) $\phi(\ell) \neq \phi(\ell + 1)$, where $\phi(\ell)$ is the level of **a**.

Different choices of ϕ result in a spectrum of algorithms. For example, if $\phi(\ell) = 0$ for all ℓ, we obtain a *perfectly balanced binary tree*. If $\phi(\ell) = \ell$, we obtain a *random binary tree*. We will use the following function in this paper: $\phi(\ell) = \max\{0, \ell - \lceil \log_2 \ell \rceil - c\}$, where c denotes a small constant that we will shortly fix.

2.2 Handling Host Arrivals

All internal nodes of the tree are conceptual. Only the leaf nodes correspond to actual host IDs. The "random walk down the tree" is equivalent to identifying **r**, the manager of a point chosen uniformly at random from the interval $\mathcal{I} = [0, 1)$. "Perfect insertion into the sub-tree rooted at **a**, the active ancestor of **r**", can be accomplished as follows: Let **r**

have an ℓ-bit ID. Let $S(\mathbf{r})$ denote the set of IDs that share the top $\phi(\ell)$ bits with $\mathbf{r}$. If $|S(\mathbf{r})| \geq 2^{\ell-\phi(\ell)}$, then we split $\mathbf{r}$. Otherwise, there exists an $(\ell-1)$-bit ID $\mathbf{s} \in S(\mathbf{r})$, which we split. *Splitting* a node $\mathbf{x}$ amounts to replacing $\mathbf{x}$ with two IDs: $\mathbf{x}0$ and $\mathbf{x}1$. Treated as fractions in $[0,1)$, $\mathbf{x}$ and $\mathbf{x}0$ are equivalent. The newly arrived host is assigned $\mathbf{x}1$. The message complexity of the algorithm is $\Theta(R+\log n)$, as shown in Theorem 2.2.

Optimizations: In practice, $S(\mathbf{r})$ could be maintained as the value associated with a hash key that includes $\mathbf{a}$ as a sub-key. Alternately, a specific node within the sub-tree rooted at $\mathbf{a}$ (for example, the leftmost child of the sub-tree) could be made responsible for this data structure. In either case, the data structure can be retrieved and updated in $\Theta(R)$ messages. However, the *congestion* (work done per node) caused by ID selection would no longer be uniform[2].

2.3 Analysis

We will prove that w.h.p., leaf nodes lie in at most three different levels and that a host arrival costs $\Theta(R+\log n)$ messages w.h.p. The three levels which leaves belong to are $[\log_2 n]$ and $[\log_2 n] \pm 1$, where $[x]$ denotes the integer closest to real number x, and n denotes the total number of hosts in the system.

The *phase* $\phi(\ell)$ of level $\ell \geq 0$ is defined as:

$$\phi(\ell) = \max\{0, \ell - \lceil \log_2 \ell \rceil - c\}$$

The *density* $\chi(\ell)$ for $\ell \geq 0$ is defined as:

$$\chi(\ell) = \begin{cases} 1 & \text{if } \ell = 0 \\ 2^{\ell-1} & \text{if } \phi(\ell) = 0 \text{ but } \ell \neq 0 \\ 2^{\lceil \log_2 \ell \rceil + c - 1} & \text{otherwise} \end{cases}$$

For $\ell \geq 0$, let *probability* $p(\ell) = 2^{-\phi(\ell)}$.

We define random variables $Y(\ell) = \sum_{i=0}^{\ell} \sum_{j=1}^{\chi(i)} y_{ij}$ where y_{ij} is a geometric random variable with parameter $p(i)$. The expected value of y_{ij} is $1/p(i)$. The expected value of $Y(\ell)$ is $\mathbf{E}Y(\ell) = \sum_{i=0}^{\ell} \chi(i)/p(i) = 1 + \sum_{i=1}^{\ell} 2^{i-1} = 2^{\ell}$

[2]In a real system, it is likely that the work associated with ID selection forms a small fraction of the total load on the system. Hence, non-uniform congestion may be acceptable.

What is our interest in $Y(\ell)$? Consider $\mathbf{a}$, an active node awaiting insertion of a new leaf at level ℓ. Our algorithm ensures that $\mathbf{a}$ must be at level $\phi(\ell)$. The probability that a newly-arrived host gets inserted as a leaf in the sub-tree rooted at $\mathbf{a}$ is $p(\ell) = 2^{-\phi(\ell)}$. Thus the total number of node arrivals before a leaf gets inserted below $\mathbf{a}$ happens to be a geometric variable with probability $p(\ell)$. Now $\chi(\ell)$ equals the number of leaf nodes inserted at level ℓ in the sub-tree rooted at $\mathbf{a}$ when $\mathbf{a}$ was active. In the past, each node along the path from the root to $\mathbf{a}$ has played the role of an active node. Thus the sum $Y(\ell)$ denotes the total number of node arrivals such that all leaf positions at level ℓ in a specific sub-tree rooted at level $\phi(\ell)$ are occupied.

Lemma 2.1

(a) With $N = 2^{\ell}$, $Pr[Y(\ell) > (1+\delta)N] < 1/N^2$, if constant c is chosen suitably (as a function of δ).

(b) With $N = 2^{\ell}$, $Pr[Y(\ell) < (1-\delta)N] < 1/N^2$ if constant c is chosen suitably (as a function of δ).

Proof: In Appendix. □

Lemma 2.2 *Let the total number of IDs be n.*

(a) If $2^{\ell-1} < n < \frac{7}{8}2^{\ell}$, then w.h.p., no leaf is at level $\ell+1$ or more (for a suitably large constant c).

(b) If $\frac{5}{4}2^{\ell} < n < 2^{\ell+1}$, then w.h.p., no leaf is at level $\ell-1$ or less (for a suitably large constant c).

Proof: (a) Let $N = 2^{\ell}$. Consider a specific host $\mathbf{r}$. (Host $\mathbf{r}$ is at level $\ell+1$ or more within $n < \frac{7}{8}2^{\ell}$ steps) $\Rightarrow (Y(\ell) < \frac{7}{8}N)$. Plugging $\delta = \frac{1}{8}$ in Lemma 2.1(b), we obtain $Pr[Y(\ell) < (1-\delta)N] < 1/N^2$ for suitably large c. Now, $1/N^2 = O(1/n^2)$. Summing over all n nodes in the system, we arrive at the claim.

(b) Let $N = 2^{\ell}$. Consider a specific host $\mathbf{r}$. (Host $\mathbf{r}$ is at level $\ell-1$ or less even when $n > \frac{5}{4}N$) $\Rightarrow (Y(\ell) > \frac{5}{4}N)$. Plugging $\delta = \frac{1}{4}$ in Lemma 2.1(a), we obtain $Pr[Y(\ell) > (1+\delta)N] < 1/N^2$ for suitably large c. Now, $1/N^2 = O(1/n^2)$. Summing over all n nodes in the system, we arrive at the claim.

As far as c is concerned, there is no conflict arising from cases (a) and (b) because c is bounded from below in both cases. Moreover, the constants $\langle \frac{7}{8}, \frac{5}{4} \rangle$ can be replaced with any $\langle r_1, r_2 \rangle$ satisfying $0 < r_1 < 1 < r_2$ and $r_2 < 2r_1$. □

Theorem 2.1 *With n leaf nodes, all leaf nodes lie in levels $\lfloor \log_2 n \rfloor$ and $\lfloor \log_2 n \rfloor \pm 1$, w.h.p. Thus $\sigma \leq 4$.*

Proof: Follows from Lemma 2.2. □

Leaf nodes belong to three different levels only when n hovers around a power of two. Otherwise, leaf nodes belong to only two different levels.

Theorem 2.2 *With n hosts in the system, a new host-arrival costs $\Theta(R + \log n)$ messages w.h.p.*

Proof: The cost of identifying $\mathbf{r}$, the manager of a random number in $[0, 1)$ costs R messages, w.h.p. Computing $S(\mathbf{r})$ (see §2.2 for its definition) entails $\Theta(\log n)$ messages if we use successor/predecessor links along the circle. This is because $|S(\mathbf{r})| \leq 2 \times 2^{\ell - \phi(\ell)} = \Theta(\ell)$ (from the definition of $\phi(\ell)$) and $\ell = \Theta(\log n)$ (Theorem 2.1). □

2.4 High-Degree Trees

The partition balance ratio for the scheme outlined above is 4 because all internal nodes have degree two and nodes are guaranteed to lie in at most three different levels. To obtain smaller values of σ, we stipulate that the degree of internal nodes that have leaf nodes as their children, is free to have any value in the range $[b, 2b - 1]$ for some $b \geq 2$. All other internal nodes still have degree exactly two. The smallest trees we consider have b leaf nodes. The degree constraint is reminiscent of B-trees [BM72]. However, our tree is quite different since only those internal nodes that are parents of leaves are allowed to have degree other than two.

The definition of active nodes is the same as before. The addition algorithm remains largely unaltered – when a node $\mathbf{x}$ becomes active for the first time, the degree of all internal nodes within the sub-tree rooted at $\mathbf{x}$, all of whose children are leaves, is exactly b. None of these nodes becomes degree $b + 2$ unless all of them are degree $b + 1$, and so on. Finally, enough leaves would have arrived so that all these nodes have degree $2b - 1$. Thereafter, insertions into the active sub-tree split internal nodes into pairs of internal nodes, each with degree b. Note that *every* insertion causes changes in the degree of some existing node. Each change in degree is concomitant with re-assignments of at least $b - 1$ and at most $2b - 1$ existing IDs.

Theorem 2.3 *If $\phi(\ell) = \ell - \lceil \log_2 \ell \rceil - c$, where $c = \Theta(\log \epsilon^{-2})$, the modified ID selection scheme guarantees $\sigma \leq 1 + \epsilon$ w.h.p.*

Proof: We present a brief sketch: Let us split the growth of n from $b2^\ell$ to $b2^{\ell+1}$ into b epochs. Each epoch sees 2^ℓ node arrivals. Using an argument similar to Lemmas 2.1(a) and 2.1(b), we can show that (a) if $(b + i - 1)2^\ell < n < \frac{7}{8}(b + i)2^\ell$, for $1 \leq i < b$, then w.h.p., no internal node has degree $b + i$ or more (for a suitably large value of c), and (b) if $\frac{5}{4}(b + i)2^\ell < n < (b + i + 1)2^\ell$ for $1 \leq i < b$, then w.h.p., no internal node has degree $b + i - 1$ or less (for a suitably large value of c). It follows that with high probability, the maximum value of $\sigma = 1 + 2/b$. If we let $\epsilon = 2/b$, then the value of c ought to be $c = O(\log \epsilon^{-2})$ for the theorem to be true. □

In terms of levels, for $b \geq 2$, leaves belong to two different levels only when n is very close to $b2^\ell$ for some ℓ; otherwise, leaves occupy just one level.

3 Emulation of Topologies

The guarantee that all leaf nodes belong to three different levels allows *emulation* of various families of deterministic and randomized inter-connection networks, in a straightforward fashion. We believe that our scheme in §3.1 is simpler than previous proposals [AAA+03, NW03, Man03].

A family of deterministic routing networks is typically defined over 2^k or $k2^k$ nodes where k is a positive integer [Lei92]. Examples are hypercubes, butterflies, shuffle-exchange networks, de Bruijn networks, etc. Recently, families of randomized routing topologies have been defined over 2^k nodes where k is a positive integer. For example, Symphony [MBR03] is an adaptation of Kleinberg's construction [Kle00]. Symphony, along with randomized variants of Chord and hypercubes [GGG+03, ZGG03] is studied in [MNW04].

3.1 Emulation with 3 Sets of Links

Families of inter-connection networks defined over 2^k nodes can be emulated as follows[3]. Recall that w.h.p., all leaf nodes (host IDs) lie in levels levels $\lfloor \log_2 n \rfloor$ and $\lfloor \log_2 n \rfloor \pm 1$ where n is the current number of hosts. A host at level ℓ constructs three

[3]Emulation of networks defined over $k2^k$ nodes can also be easily accomplished. We omit the details for lack of space.

sets of links corresponding to the top ℓ, $\ell-1$ and $\ell-2$ bits of its ID respectively. As a consequence, all hosts are guaranteed to have established links corresponding to the top $\lfloor \log_2 n \rfloor - 1$ bits of their respective IDs. Routing starts off by following the links at the lowest-numbered level at the source. If the message reaches some host which does not have links at that level, we switch to a level one higher than the current. With high probability, we are guaranteed to get *caught* in level $\lfloor \log_2 n \rfloor - 1$.

Having three sets of links per host does not necessarily triple the number of outgoing-links. Several high-degree parallel networks, like hypercubes and de Bruijn graphs, have a nice recursive structure such that the three sets of links have almost all the links in common. Even for randomized topologies like Symphony, the number of different links is only two in expectation. However, for the butterfly and its variants, the three sets are indeed quite different.

3.2 Emulation with only 2 Sets of Links

We borrow an idea from [Man03]. Each host runs a black-box called *Network Size Estimator*, whose goal is to maintain $\tilde{n}$, an estimate for n, the current number of participating hosts. We stipulate that $\tilde{n} \in n/(1 \pm \delta)$ where $\delta < 1/3$ w.h.p. Armed with $\tilde{n}$, a node computes two B-values: $B_1 = \lfloor \log_2 \tilde{n} \rfloor$ and $B_2 = \lceil \log_2 \tilde{n} \rceil$. When $\delta < 1/3$, with high probability, there are no more than three different B-values that nodes believe in, and there exists some B-value that is common to all nodes. In a practical system, network size estimates could be derived from random sampling where samples can be obtained by *snooping* on routing traffic. In §3.3, we show how a simple modification to the algorithm in §2 allows estimation to be done without snooping.

We call the set of nodes that share the top B_1 (and B_2) bits of a node, a *blob*. Thus a node belongs to exactly two blobs. Since B_1 and B_2 are successive integers, one of the blobs is a superset of the other. For inter-blob routing, a node establishes two sets of inter-blob links: one set corresponding to using its top B_1-bits as its blob ID and another set corresponding to using its top B_2-bits as its blob ID. For the other end of an inter-blob link, any node in the destination blob suffices. Routing starts off by following links corresponding to the smaller of the two B-values at the source. Routing switches to the next higher B-value if it encounters a node which believes in a different pair of B-values. This B-value is guaranteed to be common to all nodes. Once the destination blob has been reached, intra-blob routing takes $\Theta(1)$ hops since the size of all blobs is $\Theta(1)$ w.h.p.

An issue that was not addressed in [Man03] is the *flapping* of link-sets when $\tilde{n}$ hovers around a power of two. A simple modification to the emulation scheme prevents flapping. We stipulate that $\delta = 3-2\sqrt{2}$ for the Network Size Estimator. Consequently, the difference between $\log_2 \tilde{n}$ for two different hosts is at most $1/2$. We now introduce *hysteresis* to absorb small fluctuations in $\log_2 \tilde{n}$: A node switches from a pair of B-values $\langle b, b+1 \rangle$ to $\langle b+1, b+2 \rangle$ only when $\log_2 \tilde{n}$ crosses the boundary $b+1/2$ by *increasing* in value. The switch from $\langle b+1, b+2 \rangle$ to $\langle b, b+1 \rangle$ is made only when $\log_2 \tilde{n}$ crosses the boundary b by *decreasing* in value.

3.3 Estimating the Number of Leaves

How could a host derive a sharp estimate $\tilde{n}$ satisfying $\tilde{n} \in n/(1 \pm \delta)$ for a fixed $\delta < 3 - 2\sqrt{2}$, using only local information? A factor-4 approximation of n can be deduced from the number of bits in a host's own ID since all leaves are guaranteed to lie in levels $\lfloor \log_2 n \rfloor$ and $\lfloor \log_2 n \rfloor \pm 1$, w.h.p. It is possible to improve this estimate as follows.

Theorem 3.1 *Imagine n node arrivals where each node chooses a random number in $\mathcal{I} = [0, 1)$. If $X(k)$ nodes choose the same top k bits for their IDs such that $X(k) \geq 8(1+\delta)\delta^{-2} \ln n$, then $2^k X(k) \in n/(1 \pm \delta)$ with probability at least $1 - 2/n^2$.*

Proof: By application of Chernoff's inequality. □

Theorem 3.2 *Each host can estimate $\tilde{n} \in n/(1 \pm \delta)$ w.h.p., if c is large enough, by a modification of the ID selection algorithm (at no extra cost).*

Proof: A new host, upon joining the system, identifies the size of the sub-tree hanging below the *parent* of its active ancestor, and deduces $\tilde{n}$ from this size, as discussed below. The new host then informs all other hosts below the sub-tree about its estimate.

Let the newly-arrived host correspond to a leaf node at level ℓ. Let **a** denote the active ancestor of

the leaf node when it was created. Then $\mathbf{a}$ is at level $\phi(\ell)$. Consider $\mathbf{a}'$, the parent $\mathbf{a}$, which would be at level $\phi(\ell) - 1$. Let Z denote the total number of leaf nodes in the sub-tree rooted at $\mathbf{a}'$. Then $Z \geq \chi(\ell) = 2^{\lceil \log_2 \ell \rceil + c - 1} \geq \ell 2^{c-1}$. Now, $\ell \geq \log_2 n - 2$ since all leaves lie in the bottom three levels of the tree. Therefore, $Z \geq (\log_2 n - 2)2^{c-1}$. Each of these leaves chose the same top $\phi(\ell) - 1$ bits for their ID. In the past, for each $i \in [0, \ell - 2]$, $\chi(i)$ nodes chose their IDs such that the top $\phi(i)$ bits form a prefix of the ID of $\mathbf{a}'$. If we (conceptually) let each of these nodes carry out further coin tosses so as to decide which one of them would eventually have gone on to share the top $\phi(\ell) - 1$ bits with $\mathbf{a}'$, we would augment Z to obtain a quantity Z'. We claim that $\tilde{n} = 2^{\phi(\ell)-1} Z'$ is an unbiased estimate for n. In fact, if we fix c to the smallest integer that satisfies $(\log_2 n - 2)2^{c-1} \geq 8(1+\delta)\delta^{-2} \ln n$, we can apply Theorem 3.1 to claim that $\tilde{n} \in n/(1 \pm \delta)$ w.h.p. The construction in §3.2 works with any constant $\delta < 3 - 2\sqrt{2}$, which makes c a constant. □

Related Work: Estimation of $\log n$ is a sub-problem that often emerges in the context of DHTs. Viceroy [MNR02], Symphony [MBR03] and an optimal randomized protocol [Man03] require the estimate for constructing the overlay routing topology. Chord/CFS [DKK+01] requires the estimate to establish $\Theta(\log n)$ virtual IDs per node. Naor and Wieder [NW03] require the estimate to probe $\Theta(\log n)$ random points in $[0, 1)$ for ID selection.

Previous work has estimated network size when each host independently chooses an ID from $[0, 1)$ uniformly at random. Viceroy [MNR02] showed that a *constant-factor* approximation of $\log n$ could be deduced from two successive IDs. A scheme in [Man03] improves the estimate to *constant-additive* error in $\log n$. Recently, a new scheme for estimating $\log n$ within constant-factors has been devised by Horowitz and Malkhi [HM03].

4 Handling Host Departures

We now develop a variant of our earlier algorithm to handle host departures as well. The variation makes the resulting data structure amenable to analysis, albeit at the cost of increased complexity.

We create binary trees, as before. The ID of a leaf node is the sequence of 0s and 1s from the root to that node. There is always a 1-1 correspondence between hosts and leaf nodes in the tree, their IDs being the same. Further, when a host joins the system, it arrives with an infinite string of random bits. During the lifetime of a host, it can swap its bit-string with that of another host. Thus each host possesses exactly one bit-string at any time – this bit-string is not necessarily the one it joined the system with. The departure of a randomly chosen host is equivalent to deletion of a randomly chosen bit-string in the system. Swapping of bit-strings is different from *re-assignment* of host IDs, which occurs only when the tree itself undergoes structural changes. We now define formally, two simple procedures on sub-trees that our algorithm uses.

4.1 Perfect Insertion and Perfect Deletion

Definition of "perfect insertion of a newly-arrived host below internal node $\mathbf{x}$": Starting at $\mathbf{x}$, we repeatedly move down either the left or the right sub-tree, whichever has fewer leaf nodes, breaking ties arbitrarily. Let $\mathbf{r}$ denote the leaf node we reach. We *split* $\mathbf{r}$ into two by creating two leaf nodes. The newly-arrived host occupies the right leaf. The host that occupied $\mathbf{r}$ now occupies the left leaf.

Definition of "perfect deletion of a specific host (corresponding to some leaf node) below a specific internal node $\mathbf{x}$": Let the host correspond to leaf node $\mathbf{r}$ at level ℓ. If ℓ is the deepest level at which leaf nodes exist below $\mathbf{x}$, we delete both $\mathbf{r}$ and its sibling from the tree. The host at $\mathbf{r}$ leaves the system. The host at the sibling effectively "moves up", its ID possibly having been re-assigned since we chopped off its least-significant bit. If ℓ is not the deepest level below $\mathbf{x}$, we identify some pair of sibling leaf nodes that lie at the deepest level. We delete both of these leaf nodes. In terms of hosts, the host corresponding to the left sibling "moves up", its ID remaining effectively the same (we chopped off its least-significant bit, which was 0). The host corresponding to the right sibling is re-assigned – it takes up the ID corresponding to leaf node $\mathbf{r}$. The host that previously occupied $\mathbf{r}$ leaves the system. The bit-string in its possession also vanishes from the system.

4.2 Definitions and Overview

Each internal node belongs to one of four states: B, F, F* or A. The letters are acronyms for the words

$\mathbf{x}$ is in state F	$\Rightarrow$	$N(\mathbf{x}) \geq \psi(\mathbf{x})$	$\wedge$	$\Big(N(\mathbf{x}0) < \psi(\mathbf{x})/2$	$\vee$	$N(\mathbf{x}1) < \psi(\mathbf{x})/2\Big)$
$\mathbf{x}$ is in state F*	$\Rightarrow$	$N(\mathbf{x}) \geq \psi(\mathbf{x})$	$\wedge$	$\psi(\mathbf{x})/2 \leq N(\mathbf{x}0) < \psi(\mathbf{x}0)$	$\wedge$	$\psi(\mathbf{x})/2 \leq N(\mathbf{x}1) < \psi(\mathbf{x}1)$
$\mathbf{x}$ is in state A	$\Rightarrow$	$N(\mathbf{x}) \geq \psi(\mathbf{x})$	$\wedge$	$N(\mathbf{x}0) \geq \psi(\mathbf{x}0)$	$\wedge$	$N(\mathbf{x}1) \geq \psi(\mathbf{x}1)$

Figure 1: Three of the invariants maintained by our algorithm are listed above. For each invariant, the condition on the Right-Hand-Side AND *the condition that all ancestors of* $\mathbf{x}$ *are in state* A*, implies the Left-Hand-Side. This results in three additional invariants.*

BELOW, FRONTIER, FRONTIER* and ABOVE, respectively. We maintain the invariant that for every leaf node, exactly one of its ancestors is marked as F or F* at any time. Thus the union of F and F* nodes acts as a frontier, partitioning leaf nodes into disjoint groups. Every internal node that is *above* the frontier is in state A. All other internal nodes are *below* and in state B. The frontier "moves down" in response to additions and "moves up" in response to deletions. When a new host arrives, the state of an existing node either remains the same, or one of the following transitions takes place: B→F, B→F*, F→F*, or F*→A. In response to deletions, either the state remains the same, or one of the following transitions occurs: B←F, B←F*, F←F*, or F*←A.

For brevity, we will use $\mathbf{x}$ to denote both an internal node and the string of 0s and 1s from the root to that node. Thus $\mathbf{x}0$ and $\mathbf{x}1$ denote the left and the right child of $\mathbf{x}$, respectively.

Definitions: Let $N(\mathbf{x})$ denote the number of random bit-strings with prefix $\mathbf{x}$, currently in the system. The definition of ϕ remains unchanged: $\phi(\ell) = \max\{0, \ell - \lceil \log_2 \ell \rceil - c\}$. We define function ψ for integer $k \geq 0$, as follows: $\psi(k) \equiv 2^{x-\phi(x)}$, where $x = \max\{\ell \mid \phi(\ell) = k\}$. By definition, $\psi(k+1)$ equals either $\psi(k)$ or $2\psi(k)$, for all $k \geq 0$. For string $\mathbf{x}$, $\psi(\mathbf{x}) \equiv \psi(|\mathbf{x}|)$, where $|\mathbf{x}|$ denotes the length of string $\mathbf{x}$ (the same as the level of node $\mathbf{x}$).

We maintain the following invariant for every node $\mathbf{x}$ in state A, F or F*: *a bit-string is possessed by some leaf node below* $\mathbf{x}$ *iff* $\mathbf{x}$ *is a prefix of that bit-string.* An node in state B may or may not satisfy this invariant. Figure 1 describes six additional invariants that are always maintained. The addition and deletion algorithms in §4.3 and §4.4 basically show how all the invariants can be maintained in response to arrivals and departures of hosts.

4.3 Addition Algorithm

If there are fewer than $\psi(0)$ hosts in the system, all internal nodes are in state B – we maintain a complete binary tree at all times (using "perfect addition" and "perfect deletion" below the root node). When there are exactly $\psi(0)$ hosts in the system, the root node undergoes the transition B→F, or B→F*. If $N(\mathbf{0}) = N(\mathbf{1}) = \psi(0)/2$, then B→F* occurs; otherwise, B→F occurs. For the rest of the section, we assume that the root is not in state B.

When a new host arrives, it is in possession of an infinite random bit-string. Corresponding to the bits of this bit-string, we carry out a random walk down the tree. Let $\mathbf{r}$ denote the leaf node we reach. Let $\mathbf{a}$ denote the ancestor of $\mathbf{r}$ in state F or F*.

Case I ($\mathbf{a}$ is in state F): If $N(\mathbf{a}0) < \psi(\mathbf{a})/2$, we carry out "perfect insertion" below the sub-tree rooted at $\mathbf{a}1$. Otherwise, $N(\mathbf{a}1) < \psi(\mathbf{a})/2$ and we carry out "perfect insertion" below the sub-tree rooted at $\mathbf{a}0$. The bit-string of the newly-arrived host increments either $N(\mathbf{a}0)$ or $N(\mathbf{a}1)$. After the increment, if $\min\{N(\mathbf{a}0), N(\mathbf{a}1)\} \geq \psi(\mathbf{a})/2$, then $\mathbf{a}$ undergoes the transition F→F*; otherwise it remains in state F. Note that in state F*, the number of leaf nodes below $\mathbf{a}0$ and $\mathbf{a}1$ are *exactly* $N(\mathbf{a}0)$ and $N(\mathbf{a}1)$, respectively. This property is maintained in state F*, as described below.

Case II ($\mathbf{a}$ is in state F*): If $\mathbf{a}0$ is a prefix of the random bit-string of the newly-arrived host, we carry out "perfect insertion" below $\mathbf{a}0$. Otherwise, we carry out "perfect insertion" below $\mathbf{a}1$. After the insertion, if $N(\mathbf{a}0) \geq \psi(\mathbf{a}0)$ and $N(\mathbf{a}1) \geq \psi(\mathbf{a}1)$, we invoke procedure HANDLE($\mathbf{a}$), described below:

HANDLE($\mathbf{x}$) first re-distributes the bit-strings currently in possession of leaf nodes below $\mathbf{x}$: All bit-strings with prefix $\mathbf{x}0$ are made to lie below $\mathbf{x}0$, and all bit-strings with prefix $\mathbf{x}1$ are made to lie below $\mathbf{x}1$. The state of $\mathbf{x}$ then changes to A. At this

point, if $N(\mathbf{x}00) \geq \psi(\mathbf{x}00)$ and $N(\mathbf{x}01) \geq \psi(\mathbf{x}01)$, we invoke HANDLE($\mathbf{x}0$) recursively; else if $N(\mathbf{x}00) \geq \psi(\mathbf{x}0)/2$ and $N(\mathbf{x}01) \geq \psi(\mathbf{x}0)/2$, $\mathbf{x}0$ is assigned state F*; else $\mathbf{x}0$ is assigned state F. Node $\mathbf{x}1$ is dealt with similarly. We note that a recursive call is in fact, a rare event, as shown in §4.5.

The addition algorithm differs from that in §2.1 in that the transition F→A has been *delayed* by introducing an intermediate state F*. Plus, we are maintaining some random bit-strings along with hosts.

4.4 Deletion Algorithm

Let $\mathbf{r}$ denote a randomly-chosen host that departs. Let $\mathbf{a}$ denote the ancestor of $\mathbf{r}$ in state F or F*. Departure of $\mathbf{r}$ will decrement $N(\mathbf{a})$. Moreover, either $N(\mathbf{a}0)$ or $N(\mathbf{a}1)$ will also be decremented, depending upon the prefix of the random bit-string in possession of $\mathbf{r}$. In the cases below, the values of $N(\mathbf{a})$, $N(\mathbf{a}0)$ and $N(\mathbf{a}1)$ correspond to the *updated* values (following the decrement).

Case I: $N(\mathbf{a}) \geq \psi(\mathbf{a})$:

(a) $\big(\psi(\mathbf{a})/2 \leq N(\mathbf{a}0) < \psi(\mathbf{a}0)$ and $\psi(\mathbf{a})/2 \leq N(\mathbf{a}1) < \psi(\mathbf{a}1)\big)$: We carry out "perfect deletion" of $\mathbf{r}$ below $\mathbf{a}0$ or $\mathbf{a}1$, whichever happens to be the ancestor of $\mathbf{r}$. Node $\mathbf{a}$ remains in state F*.

(b) $\big(N(\mathbf{a}0) < \psi(\mathbf{a})/2$ or $N(\mathbf{a}1) < \psi(\mathbf{a})/2\big)$: We carry out "perfect deletion" of $\mathbf{r}$ below $\mathbf{a}$. Node $\mathbf{a}$ undergoes F←F* if its state was F* before the departure of $\mathbf{r}$.

Case II: $N(\mathbf{a}) < \psi(\mathbf{a})$: We carry out "perfect deletion" of $\mathbf{r}$ below $\mathbf{a}$. Internal node $\mathbf{a}'$, the parent of $\mathbf{a}$, undergoes the transition F*←A. Node $\mathbf{a}$, its *sibling*, and all descendants of its sibling which are presently in state F or F*, undergo B←F or B←F*, whichever is applicable. This cascade of state-transitions is the "inverse" of the recursive call to procedure HANDLE that was described earlier. Only the sibling changes state w.h.p.

4.5 Analysis

Lemma 4.1 *If $\frac{5}{4}2^{\ell} < n < 2^{\ell+1}$, then w.h.p., no leaf is at level $\ell - 1$ or less (for suitably large c).*

Proof: If $\phi(\ell) \neq \phi(\ell+1)$, the lemma follows from the following series of claims, each holding w.h.p.

A1: If $|\mathbf{a}| \leq \phi(\ell)$, then $N(\mathbf{a}) \geq \psi(\mathbf{a})$.
A2: If $|\mathbf{a}| < \phi(\ell)$, then $\mathbf{a}$ is in state A.
A3: If $|\mathbf{a}| = \phi(\ell)$, then $\mathbf{a}$ is in state A, F or F*.
A4: No leaf is at level $\ell - 1$ or less.

Proof of A1: Let $\eta = 2^{\ell}$. Then $N(\mathbf{a}) = B(\frac{5}{4}\eta, 2^{-|\mathbf{a}|})$, the sum of $\frac{5}{4}\eta$ Bernoulli variables, each with probability $2^{-|\mathbf{a}|}$. The expected value of $N(\mathbf{a})$ is $\frac{5}{4}2^{\ell-|\mathbf{a}|} \geq \frac{5}{4}\psi(\mathbf{a})$. Since $\psi(\mathbf{a}) = \Omega(\log \eta)$, application of Chernoff bound shows that the event $N(\mathbf{a}) < \psi(\mathbf{a})$ holds with probability at most $1/\eta^2$, for a suitably large c. A union bound on all n nodes makes the probability of failure at most $O(1/\eta) = O(1/n)$.

Claims A2 and A3 follow from A1 and the invariants in Figure 1. Claim A4 follows from the fact that all nodes in level $\ell - 1$ and less are occupied if all nodes in level $\phi(\ell)$ are in state A, F or F*.

If $\phi(\ell) = \phi(\ell+1)$, the lemma follows from the following series of claims, each holding w.h.p. (the overall proof is along the same lines as that above).

B1: If $|\mathbf{a}| \leq \phi(\ell) - 1$, then $N(\mathbf{a}) \geq \psi(\mathbf{a})$.
B2: If $|\mathbf{a}| < \phi(\ell) - 1$, then $\mathbf{a}$ is in state A.
B3: If $|\mathbf{a}| = \phi(\ell) - 1$, then $\mathbf{a}$ is in state A, F or F*.
B4: Let $\mathbf{a}'$ denote the sibling of $\mathbf{a}$. Then $N(\mathbf{a}) \geq \psi(\mathbf{a})/2$ and $N(\mathbf{a}') \geq \psi(\mathbf{a}')/2$.
B5: No leaf is at level $\ell - 1$ or less. □

Lemma 4.2 *If $2^{\ell-1} < n < \frac{7}{8}2^{\ell}$, then w.h.p., no leaf is at level $\ell + 1$ or more (for suitably large c).*

Proof: Consider internal node $\mathbf{a}$ with $|\mathbf{a}| = \phi(\ell)$. The lemma follows from the following claims:

C1: If $\phi(\ell) \neq \phi(\ell+1)$, then $N(\mathbf{a}) < \psi(\mathbf{a})$ w.h.p.
C2: If $\phi(\ell) = \phi(\ell+1)$, then $N(\mathbf{a}) < \psi(\mathbf{a})/2$ w.h.p.
C3: In either case, the "next insertion" below any F/F* node is going to be at level ℓ or less. Thus, there is no leaf at level $\ell + 1$ or above, w.h.p. □

Theorem 4.1 *With n hosts, all leaves lie in levels $\lfloor \log_2 n \rfloor$ and $\lfloor \log_2 n \rfloor \pm 1$. Thus $\sigma \leq 4$.*

Proof: Follows from Lemmas 4.2 and 4.1. □

4.6 Extensions

The algorithm outlined in §4.3 and §4.4 can be extended along the lines of §2.4 (high-degree trees) to achieve partition balance ratio $1 + \epsilon$, w.h.p. Further, an accurate estimate of n can be obtained by scaling the size of the sub-tree below any internal node in state F or F*. This enables emulation of a variety of inter-connection topologies with only two sets of links per host, as described earlier in §3.2.

4.7 Concluding Remarks

Both of our ID selection algorithms hinge upon the intuition that if a sub-interval of $[0, 1)$ has size $\Theta(\frac{\log n}{n})$, then $\Theta(\log n)$ out of n randomly-chosen points are guaranteed to fall into that sub-interval, w.h.p. This fact has been noted in an early paper on DHTs (Viceroy [MNR02]). However, the fact was never exploited to *perturb* the randomly-chosen points slightly to obtain fine-grained partition balance ratios. Four different groups [AAA+03, AHKV03, KR03, NW03] have since then developed various ID selection algorithms – the intuition in these algorithms is quite different.

Thanks to Mayank Bawa, Mayur Datar, Krishnaram Kenthapadi and Rajeev Motwani for reading the paper.

References

[AAA+03] I Abraham, B Awerbuch, Y Azar, Y Bartal, D Malkhi, and E Pavlov. A generic scheme for building overlay networks in adversarial scenarios. *Proc. Intl. Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.

[AHKV03] M Adler, E Halperin, R M Karp, and V V Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. *Proc. 35nd ACM Symposium on Theory of Computing (STOC 2003)*, p. 575–584, 2003.

[BM72] R Bayer and E M McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.

[DKK+01] F Dabek, M F Kaashoek, D Karger, R Morris, and I Stoica. Wide-area cooperative storage with CFS. *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, p. 202–215, 2001.

[FG03] P Fraigniaud and P Gauron. (brief announcement) An overview of the content-addressable network D2B. *Proc 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, p. 151–151, 2003.

[GGG+03] K P Gummadi, R Gummadi, S D Gribble, S Ratnasamy, S Shenker, and I Stoica. The impact of DHT routing geometry on resilience and proximity. *Proc. ACM SIGCOMM 2003*, p. 381–394, 2003.

[HJS+03] N J A Harvey, M Jones, S Saroiu, M Theimer, and A Wolman. Skipnet: A scalable overlay network with practical locality properties. *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, 2003.

[HM03] K Horowitz and D Malkhi. Estimating network size from local information. *Information Processing Letters*, 88(5):237–243, 2003.

[KK03] M F Kaashoek and D R Karger. Koorde: A simple degree-optimal hash table. *Proc. 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS 2003)*, p. 98–107, 2003.

[Kle00] J Kleinberg. The small-world phenomenon: An algorithmic perspective. *Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000)*, p. 163–170, 2000.

[KR03] D R Karger and M Ruhl. New algorithms for load balancing in peer-to-peer systems. IRIS Student Workshop, 2003.

[Lei92] F T Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes*. Academic Press/Morgan Kaufmann, 1992.

[LKRG03] D Loguinov, A Kumar, V Rai, and S Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: Routing distance and fault resilience. *Proc. ACM SIGCOMM 2003*, p. 395–406, 2003.

[Man03] G S Manku. Routing networks for distributed hash tables. *Proc. 22nd ACM Symp. on Principles of Distributed Computing (PODC 2003)*, p. 133–142, 2003.

[MBR03] G S Manku, M Bawa, and P Raghavan. Symphony: Distributed hashing in a small world. *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)*, p. 127–140, 2003.

[MNR02] D Malkhi, M Naor, and D Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. *Proc 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, p. 183–192, 2002.

[MNW04] G S Manku, M Naor, and U Wieder. Know thy neighbour's neighbour: The role of lookahead in randomized P2P networks. *Proc. 36th ACM Symposium on Theory of Computing (STOC 2004)*, 2004.

[MR95] R Motwani and P Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[NW03] M Naor and U Wieder. Novel architectures for P2P applications: The continuous-discrete approach. *Proc. 15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA 2003)*, p. 50–59, 2003.

[RD01] A I T Rowstron and P Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, p. 329–350, 2001.

[RFHK01] S Ratnasamy, P Francis, M Handley, and R M Karp. A scalable Content-Addressable Network. *Proc. ACM SIGCOMM 2001*, p. 161–172, 2001.

[SMK+01] I Stoica, R Morris, D Karger, M F Kaashoek, and H Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proc. ACM SIGCOMM 2001*, p. 149–160, 2001.

[ZGG03] H Zhang, A Goel, and R Govindan. Incrementally improving lookup latency in distributed hash table systems. *ACM SIGMETRICS 2003*, p. 114–125, 2003.

[ZHS+04] B Y Zhao, L Huang, J Stribling, S C Rhea, A D Joseph, and J D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

Appendix

Definitions from Section 2.3:

The *phase* $\phi(\ell)$ of level $\ell \geq 0$ is defined as:

$$\phi(\ell) = \max\{0, \ell - \lceil \log_2 \ell \rceil - c\}$$

The *density* $\chi(\ell)$ for $\ell \geq 0$ is defined as:

$$\begin{array}{rcll} \chi(\ell) & = & 1 & \text{if } \ell = 0 \\ & & 2^{\ell-1} & \text{if } \phi(\ell) = 0 \text{ but } \ell \neq 0 \\ & & 2^{\lceil \log_2 \ell \rceil + c - 1} & \text{otherwise} \end{array}$$

The *probability* $p(\ell)$ for $\ell \geq 0$ is defined as:

$$p(\ell) = 2^{-\phi(\ell)}$$

We define

$$Y(\ell) = \sum_{i=0}^{\ell} \sum_{j=1}^{\chi(i)} y_{ij}$$

where y_{ij} is a geometric random variable with parameter $p(i)$. The expected value of y_{ij} is $1/p(i)$. The expected value of $Y(\ell)$ is $\mathbf{E}Y(\ell) = \sum_{i=0}^{\ell} \chi(i)/p(i) = 1 + \sum_{i=1}^{\ell} 2^{i-1} = 2^{\ell}$

Lemma .3 *For all values of t and p,*

$$[\frac{2pe^t}{1-(1-2p)e^t}]^2 \leq \frac{pe^t}{1-(1-p)e^t}$$

Proof: The claim is true iff $4pe^t(1-(1-p)e^t) \leq (1-(1-2p)e^t)^2$ which simplifies to $0 \leq (1-e^t)^2$. □

Lemma .4 *If $\delta \in [0,1)$ and $p \in [0,1)$, then there exists $e^t < 1/(1-p)$ satisfying*

$$\frac{pe^t}{[1-(1-p)e^t]e^{(1+\delta)t/p}} \leq e^{-\delta^2/2}$$

Proof: Setting $e^{-t} = (1-p)(1+\delta)/(1+\delta-p)$, the expression above simplifies to $(1+\delta)\left[1 - \frac{p\delta}{1+\delta-p}\right]^{\frac{1+\delta-p}{p}}$. Using the fact that $(1-\frac{\delta}{m})^m \leq 1-\delta+\delta^2/2$ for positive $m > \delta$, the expression is no more than $(1+\delta)(1-\delta+\delta^2/2) = 1-\delta^2(1+\delta)/2 \leq e^{-\delta^2(1+\delta)/2} < e^{-\delta^2/2}$. It may be verified that $e^t < 1/(1-p)$ assuming $\delta \geq 0$ and $p \in [0,1)$. □

Lemma .5 *If $\delta \in [0, \frac{1}{2})$ and $p \in [0,1)$, then there exists $e^t > 1-p$ satisfying*

$$\frac{pe^{-t}}{[1-(1-p)e^{-t}]e^{-(1-\delta)t/p}} \leq e^{-5\delta^2(1-14\delta/15)/2}$$

Proof: Setting $e^t = 1 - \frac{p\delta}{1-\delta}$, the expression above simplifies to $\frac{1-\delta}{1-2\delta}\left(1 - \frac{p\delta}{1-\delta}\right)^{\frac{1-\delta}{p}}$. Using the inequality $(1-\frac{\delta}{m})^m < 1-\delta+\delta^2/2$ for $m > \delta$, the expression is no more than $1-5\delta^2(1-14\delta/15)/2 < e^{-5\delta^2(1-14\delta/15)/2}$. It may be verified that $e^t > (1-p)$ for $\delta \in [0, \frac{1}{2})$ and $p \in [0,1)$. □

Lemma .6 *Let $N = 2^\ell$. Let c be a constant satisfying $(\delta^2/2)2^{\lceil \log_2 \ell \rceil + c - 1} \geq 2 \log N$. Then*

$$Pr[Y(\ell) > (1+\delta)\mathbf{E}Y(\ell)] < 1/N^2$$

Proof: We employ the well-known Chernoff technique as outlined in [MR95].

Let $(*)$ denote $Pr[Y(\ell) > (1+\delta)N]$. Then

$$\begin{aligned}
(*) &= Pr[\exp(tY(\ell)) > \exp((1+\delta)tN)] \\
&= Pr[\exp(tY(\ell)) > \exp\left((1+\delta)t\sum_{i=0}^{\ell}\frac{\chi(i)}{p(i)}\right)] \\
&\leq [\mathbf{E}\exp(tY(\ell))] \,/\, \exp\left((1+\delta)t\sum_{i=0}^{\ell}\frac{\chi(i)}{p(i)}\right) \quad \text{by Markov's Inequality}
\end{aligned}$$

Now

$$\mathbf{E}\exp(tY(\ell)) = \Pi_{i=0}^{\ell}\Pi_{j=1}^{\chi(i)}\mathbf{E}\exp(ty_{ij})$$

where y_{ij} is a geometric variable with parameter $p(i)$. It turns out that $\mathbf{E}\exp(ty_{ij}) = \frac{p(i)e^t}{1-(1-p(i))e^t}$ provided $(1-p(i))e^t < 1$. Note that this introduces the constraint $e^t < 1/(1-p(i))$.

Therefore,

$$\mathbf{E}\exp(tY(\ell)) = \Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{1-(1-p(i))e^t}\right]^{\chi(i)}$$

which yields

$$\begin{aligned}
(*) &\leq \left(\Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{1-(1-p(i))e^t}\right]^{\chi(i)}\right) / \exp\left((1+\delta)t\sum_{i=0}^{\ell}\frac{\chi(i)}{p(i)}\right) \\
&= \Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{[1-(1-p(i))e^t]\exp\frac{(1+\delta)t}{p(i)}}\right]^{\chi(i)}
\end{aligned}$$

By repeated application of Lemma .3, it is possible to transform the above expression into

$$(*) \leq \Pi_{i=0}^{\ell}\left[\frac{p(i)e^t}{[1-(1-p(i))e^t]\exp\frac{(1+\delta)t}{p(i)}}\right]^{\eta(i)}$$

where $\eta(i) = 0$ or $\eta(i) = 1$ for $0 \leq i \leq \ell - 1$, and $\eta(\ell) \geq \chi(\ell)$.

For $i < \ell$, each of the terms in the product above is no more than 1. Ignoring them, we are left with the expression

$$(*) \leq \left[\frac{p(\ell)e^t}{[1-(1-p(\ell))e^t]\exp\frac{(1+\delta)t}{p(\ell)}}\right]^{\eta(\ell)}$$

We can now use Lemma .4 to claim that

$$(*) \leq e^{-\delta^2(1+\delta)\eta(\ell)/2}$$

Now $\eta(\ell) > \chi(\ell)$ where $\chi(\ell) = 2^{\lceil \log_2 \ell \rceil + c - 1}$. Recall that $N = 2^\ell$ and that we chose constant c such that $(\delta^2/2)(1+\delta)2^{\lceil \log_2 \ell \rceil + c - 1} \geq 2 \log N$. Substituting these values, we obtain

$$(*) \leq e^{-2\log N} = 1/N^2$$

□

Lemma .7 *Let $N = 2^\ell$. Let c be a constant satisfying $(5\delta^2/2)(1 - 14\delta/15)2^{\lceil \log_2 \ell \rceil + c - 1} \geq 2 \log N$. Then*

$$Pr[Y(\ell) < (1-\delta)\mathbf{E}Y(\ell)] < 1/N^2$$

Proof: Let $(**)$ denote $Pr[Y(\ell) < (1-\delta)N]$. Then

$$\begin{aligned} (**) &= Pr[-Y(\ell) > -(1-\delta)N] \\ &= Pr[\exp(-tY(\ell)) > \exp(-(1-\delta)tN)] \end{aligned}$$

Continuing in the same manner as the previous lemma, we finally obtain

$$(**) \leq \left[\frac{p(\ell)e^{-t}}{[1-(1-p(\ell))e^{-t}] \exp \frac{-(1-\delta)t}{p(\ell)}} \right]^{\eta(\ell)}$$

while introducing the constraint $e^{-t}(1-p(\ell)) < 1$, which is equivalent to $e^t > (1-p(\ell))$.

Using Lemma .5, we obtain

$$(**) \leq e^{-5\delta^2(1-15\delta/14)\eta(\ell)/2}$$

Substituting the value of c we chose, we obtain

$$(**) \leq e^{-2\log N} = 1/N^2$$

□