LAND: Locality Aware Networks for Distributed Hash Tables

Ittai Abraham, Dahlia Malkhi and Oren Dobzinski
The Hebrew University of Jerusalem, Israel
{ittaia,dalia,orend}@cs.huji.ac.il

July 2, 2003

Abstract

This paper proposes the first peer-to-peer network and lookup algorithm that has worst case constant distortion. The construction uses a constant expected number of links. The design lends itself to dynamic deployment, and has a simple and easy to verify proof. The construction embraces the two-tier architecture of current peer-to-peer networks, where stronger and stable nodes serve as ultra-peers, and other (e.g., transient home users) are regular peers.

1 Introduction

One of the main challenges of distributed systems is how to efficiently store and locate ever increasing amounts of content. The tremendous growth of the Internet illuminates this challenge. In the traditional basic web architecture, all requests reach the same server. This single service-point architecture may result in network congestion or server swamping as the number of requests increases.

The solution suggested by Karger et al. [6] is to use consistent hashing. Using a hash function, web page URLs are hashed, and from the hashed value a close-by cache server containing the page is accessed. Instead of having just one cache server (and suffer from scalability and fault tolerance problems), multiple caching servers act together to form a Distributed Hash Table (DHT). In the DHT paradigm, a lookup request is routed through the server network to the specific server that knows the answer to the lookup query.

DHTs are also at the focus of rising interest in file-sharing programs like Gnutella, Napster, and Freenet. These motivated a growing interest (e.g., see [16, 3, 17, 15, 8, 11, 10, 9, 1, 13, 12, 4, 2, 5]) in the field of distributed computing to the challenges of building DHTs: scale, dynamism, fault tolerance, decentralized control.

A case for locality-awareness. In many of the DHT works, one of the main measures of quality is the number of hops taken until the server with the required information is found. In most cases, logarithmic complexity (e.g., in [16, 3, 17, 11]) or polylogarithmic complexity (e.g., in [12]) is sought. If the only bound on reaching the target is that the number of hops is bounded, say by 4, then the lookup request could go, for example, from Boston through New Zealand, Brazil, France and finally to New York. Although the number of hops is small in this example, this clearly is not a desired outcome. While bounding the number of hops by a logarithm is important, several works [14, 15, 3, 17, 8, 2] have argued that a far more important measure is the total cost of communication between peers.

The natural way to model costs is to assume a cost function c that induces a metric space on the universe of servers. The main measure of a routing protocol is the *distortion*, namely, the ratio between the distance and the cost of a route. More precisely, let x be a lookup starting point, y be the target of the lookup (i.e., the closest node containing the searched object). Let $x = x_1, x_2, \ldots, x_k = y$ be the nodes traversed in the lookup route. Then the distortion is $\frac{c(x_1,x_2)+\ldots+c(x_{k-1},x_k)}{c(x,y)}$. The seminal work of Plaxton et al. [14] is one of the first works to provide a lookup protocol with analytical bounds on the distortion. They present a randomized scheme for a class of metric spaces representing realistic networks, in which the expected distortion in finding targets is constant. Several efforts were made to deploy this scheme, e.g., Tapestry [17] and Pastry [3]. These systems construct a dynamic DHT based on the principles of [14], yielding efficient, locality-aware lookup programs. Recently, Li and Plaxton introduce in [8] a simplified version of the PRR scheme, that yields lookup costs proportional to the diameter of the network (rather than proportional to the distance between x and y).

Our work builds on the successful approach introduced and deployed in [14, 17, 3, 15, 8]. We present the first constant distortion DHT, in which for all routes the cost ratio between the distance and the route is guaranteed to be constant. The guarantee of constant distortion is achieved while not impairing, and even improving, on other parameters of the network such as degree, memory requirements, adaptability, and fault tolerance.

A case for two-tiers. Almost all the file sharing systems used currently in Internet, e.g., Kazaa, Gnutella2, Fast Track, and others, use a hierarchical structure. Nodes are divided into nodes and super-nodes. While all nodes participate in forwarding and returning lookup requests, only the super nodes (whose reliability is higher) maintain the data itself and answer the lookup queries.

Most leading DHT constructions assume that all peers are equal [16, 17, 3, 15, 11]. In contrast, our approach enables to embrace the asymmetrical structure of real networks, and captures it in a two-tier architecture. The classification into the two tiers allows to take into account the difference between nodes in a number of domains: communication speed, persistence, CPU power and storage capacity, and so on. On the one hand, we have peers running on high performance computers that are up and connected 24×7, have high bandwidth links. On the other, we have a (potentially larger) number of home-users, whose participation is transient, and their links are slow and unreliable.

For completeness, we also provide a symmetrical construction using homogeneous nodes, which allows to contrast our work with other symmetrical DHTs.

A case for a low degree. As noted in several DHT works, e.g., in [15, 11, 1, 12, 5, 4, 13], the number of links maintained by each node affects the adaptability of the network to changes. In order for a lookup service to accommodate changes, when servers join or leave, only a small number of servers should change their state. Therefore, we desire to keep the number of connections on which a server can forward a lookup request constant, as these connections must be removed and the routing information changed when the other end leaves the network. Our primary, two-tier scheme maintains an expected constant number of outgoing links from each node. For regular nodes, the expected in-degree is also a constant. This is an important improvement over previous locality-aware DHTs which employ a logarithmic number of out-links per node [14, 3, 17, 2]. As for super-nodes, their in-degree is derived from their density in the network, and each supernode must be known to all of the regular nodes in its vicinity.

Overview of the scheme. As in several DHTs, nodes have virtual identifiers composed of string-identifiers over some base B. Objects also have virtual identifiers. Routing to a virtual address is done via *prefix routing*, namely, the i'th routing step fixes the i'th digit of the address. Likewise, objects publish their existence via prefix routing destined at their virtual address. Similar to [14], routing steps geometrically increase in cost as we fix more digits. Intuitively, this stems from the fact that there are expected n/B^i nodes matching i prefix digits in a network with n nodes. Thus, the more digits are fixed, the harder it is to find a suitable node to hop to. The total distance taken by i routing steps forms a geometric series that is dominated by the i'th step.

The novelty of our scheme lies in the choice of links, and in the analysis of the routing algorithm. Specifically, in addition to its identifier, each regular node has a *level* property between 1 and the length of identifiers. The links of the network are chosen so that a node of level i has outgoing edges allowing it to "fix" its $(i+1)$'st digit to any desired value. Unlike previous schemes, our choice of links **enforces** a distance upper bound on each stage of the route, rather than probabilistically maintaining it. This is done by having a node choose its links within an appropriately bounded distance. If no suitable endpoint is found for a particular link, we suggest a novel technique in which the node *emulates* the missing endpoint as a *virtual* *node*. The fact that a node has responsibility for "fixing" one target bit only is crucial for this emulation to work properly, as it allows bounding the amount of links emulated by each node to an expected constant.

Additionally, a node to which an object is published puts references to the object's location within its vicinity. Specifically, the node reached in the i'th step of publishing stores a reference on all level-i nodes matching the first i digits of its identifier within a certain distance. The purpose of these reference copies is to terminate the route quickly when the route from s to t already goes a distance roughly $c(s,t)$, in order to keep the distortion bounded by a constant. Here again, our scheme differs from previous methods in that the references are guaranteed to be maintained to within a certain distance, not probabilistically so.

Routing is done in three stages. Starting from a node, the first stage is a single step that proceeds to the closest supernode. The second stage performs *prefix-routing*, fixing one digit after another in the address of the target. It terminates when a node containing a reference to the object is reached. The last stage consists of simply following the reference pointers until the target is found.

Our analysis shows that the cost of a route from s to t is deterministically $O(c(s,t))$. Similar to [14, 3, 17], our locality analysis is given in a realistic network model whose metrics is *growth bounded* from above and from below.

Contribution. This paper presents a novel DHT construction that maintains locality in lookup operations. Our main design is a 2-tier system that utilizes the asymmetry of peers. We borrow heavily from the principles of [14], yet our design has the following important achievements:

- Most importantly, our constant distortion is guaranteed on all lookup routes. This bounds the cost of lookups in our scheme deterministically, while the cost bounding of [14] is only in expectation.
- Our construction is the first locality-aware DHT with only expected constant out-degree for nodes, compared with all other locality-aware schemes that have logarithmic degree.
- The main construction embraces the two-tier architecture of current peer-to-peer networks, where stronger and stable nodes serve as ultra-peers, and other (e.g., transient home users) are regular peers. A symmetrical construction is provided for completeness in Section 6.
- Our construction is simple, and our proofs short and intuitive. In contrast, proving the expected locality of [14] requires an involved and lengthy analysis. In our belief, the simplicity of our scheme and the elegance of its analysis may lead to improved practical deployments.

Organization. The rest of this paper is organized as follows. Our problem model and preliminary notations are described in Section 2. The DHT architecture of our scheme is described in Section 3. We give the full construction in Section 4. In Section 5 we analyze our results. Variations of our scheme, including a symmetric scheme for homogeneous nodes, are sketched in Section 6. The adaptation of our scheme for dynamic arrival and departure of nodes is described in Section 7. Finally, conclusions are drawn in Section 8.

2 Preliminaries

There are two types of nodes in the system, regular nodes and supernodes. Denote the set of regular nodes R, supernodes S , all nodes $V = R \bigcup S$, the number of all nodes $|V| = n$.

Let $c : V^2 \mapsto \mathbb{R}_+$ be the cost function, associated with pairs of nodes, that expresses the cost of communication between nodes. We assume c has positivity, reflexivity, triangle inequality, and symmetry. Thus $< V, c >$ forms a metric space. From here on, we refer to the cost as the *distance* between nodes.

Minimum density We assume that the minimal distance between every pair of nodes is 1.

Growth bounded metrics Growth bounded metrics are considered in several works, e.g., [14, 3, 17, 7], and reflect real-life internets. The set of nodes within distance r from x is denoted $N(x, r)$. For growth bounded metrics, we assume two constants $0 < \delta \leq \Delta$, such that for every node x and $r \geq 1$, we have

$$\delta |N(x, r)| \leq |N(x, 2r)| \leq \Delta |N(x, r)| \ . \tag{1}$$

Node identifiers Each node u (regular and super) has a string identifier denoted by $u.id$. Identifier strings are composed of M digits in radix $B = 2^b$. The radix B is chosen such that $B \geq \Delta^2$. For a network with n nodes the length of each identifier, M, is chosen such that $M = \lfloor x \rfloor$ for x satisfying $xB^x = n$.

Nodes have, in addition to their identifier, a *level* attribute, between 1 and M. Supernodes maintain two levels: one between 1 and M and the other is the level zero.

Let d be a k-digit identifier. Denote $d[j]$ as the prefix of the j most-significant digits, and denote d_j as the j'th digit of d. A concatenation of two strings d, d' is denoted by $d||d'$.

Notations and constants All notations and definitions used in our construction (including those mentioned above already) are summarized for convenience here.

$B \geq \Delta^2$.
$M = \lfloor x \rfloor$ for x satisfying $xB^x = n$.
A constant α is chosen such that $Be^{-\alpha} < 1$.
For identifiers d, d', $d[j]$ is the prefix of length j, d_j is the j'th digit, and $d||d'$ is their concatenation.
$N(x, r) = \{v \in V \mid c(x, v) \leq r\}$.
Let $A_i(x)$ denote the smallest ball around x containing $\alpha B^i M$ nodes.
Let $a_i(x)$ denote the radius of $A_i(x)$.
Denote $maxgrow = B^{\log_\delta 2}$, $mingrow = B^{\log_\Delta 2}$.

Useful properties The main properties implied by the growth-bound assumption are summarized in the following technical lemma. These properties suffice by themselves to uphold the LAND construction, and from here on we refer to the network properties only through them.

Lemma 2.1 *Let x and y be any two nodes, for any i such that $y \in A_i(x)$:*
(i) $A_i(x) \subseteq A_{i+1}(y)$.
(ii) $A_i(y) \subseteq A_{i+1}(x)$.
(iii) $a_{i+1}(x) \leq maxgrow\ a_i(x)$, where $maxgrow = B^{\log_\delta 2}$.
(iv) $a_{i+1}(x) \geq mingrow\ a_i(x)$, where $mingrow = B^{\log_\Delta 2}$.

Proof: Let $r = a_i(x)$ denote the radius of $A_i(x)$, so $A_i(x) = N(x, r)$. Since $y \in N(x, r)$ then (see Figure 1)

$$N(x, r) \subseteq N(y, 2r) \subseteq N(x, 3r) \ .$$

From the growth bounded assumption we can bound the number of nodes in $N(x, 3r)$ using $|N(x, r)|$ as follows:

$$|N(x, 3r)| \leq \Delta^2 |N(x, r)| = \Delta^2 |A_i(x)| = \Delta^2 \alpha B^i M \leq \alpha B^{i+1} M \ .$$

For (i), $N(x, 3r) \subseteq A_{i+1}(x)$, and so $A_{i+1}(y)$, the ball around y with $\alpha B^{i+1} M$ nodes, must include $N(y, 2r)$ and so must include $A_i(x)$. For (ii), $A_i(y) \subseteq N(y, 2r) \subseteq N(x, 3r) \subseteq A_{i+1}(x)$.

For (iii), $B^{\log_\delta 2} = 2^{\log_\delta B}$, thus $|N(x, B^{\log_\delta 2} r)| \geq \delta^{\log_\delta B} |N(x, r)| = \alpha B^{i+1} M$, so $A_{i+1}(x) \subseteq N(x, B^{\log_\delta 2} r)$.

Similarly, for (iv), $B^{\log_2 \Delta} = 2^{log_\Delta B}$, hence $|N(x, B^{\log_\Delta 2} r)| \leq \Delta^{\log_\Delta B} |N(x, r)| = \alpha B^{i+1} M$, so $A_{i+1}(x) \supseteq N(x, B^{\log_\Delta 2} r)$.

□

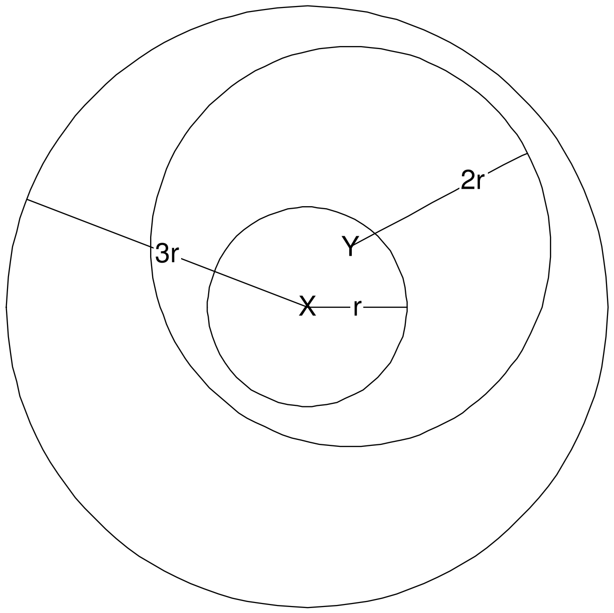

Figure 1: The circles $N(x, r)$, $N(y, 2r)$, and $N(x, 3r)$

3 LAND Architecture

The goal of our work is to support a lookup operation that locates nearest copies of objects, such that the nodes of the network share the lookup load evenly. Our method adopts the distributed hash table (DHT) approach that is employed in several recent systems [16, 3, 17, 11], with the distinction that only supernodes store objects, whereas regular nodes only assist in routing queries to supernodes.

More formally, let $\mathcal{A}$ be a set of objects. We make use of a uniformly distributed hash function H on object names. For any $A \in \mathcal{A}$, such that A is stored on some supernode s, reference information about A's location is stored on regular nodes whose identifiers have matching prefixes of various length to $H(A)$. Our network is able to locate a nearby copy of every $A \in \mathcal{A}$.

All nodes in the network take part in the routing algorithm, and pass queries to the nodes they have outgoing links to. Only supernodes hold objects, whereas all nodes store partial reference information about the location of objects. In this sense, the nodes form a *Distributed Hash Table*. Intuitively, a node with identifier $u.id$ and level ℓ has the responsibility for locating all hashed-names matching the ℓ most-significant bits of $u.id$.

4 The Routing Network

The identifier of a node is represented as radix $B = 2^b$ number. This means that each node has a B-ary identifier. The constant B is chosen such that $B \geq \Delta^2$. In addition a constant α is chosen such that $Be^{-\alpha} < 1$. We denote the maximum level as M chosen such that $MB^M = n$. The level attribute of regular nodes is chosen between 1 and M. In our construction we assume that the identifier and level of each node is randomly chosen uniformly and independently. Recall that $A_i(x)$ denotes the smallest ball around x containing $\alpha B^i M$ nodes. Recall that $a_i(x)$ denotes the radius of $A_i(x)$.

Every node v of level $v.level = \ell$ and id $v.id$ has outgoing links of three types *neighbor*, *publish*, *closest*. In addition there is a probability that v will also need to emulate some *virtual nodes* and their links. Each supernode s always emulates a level zero node with id $s.id$. Each node (and virtual node) maintains the following outgoing links.

neighbor: If $v.level = M$ then no neighbor links are formed. Otherwise, there are B neighbor links, denoted $L(0), ..., L(B-1)$. The i'th neighbor is selected as the closest node within $C_i(v) \cap A_{\ell+1}(v)$, where $C_i(v) = \{u \in V \mid u.id[\ell+1] = v.id[\ell]||i, \quad u.level = \ell+1\}$. The link $L(j)$ 'fixes' the $(\ell+1)$'st digit to j, namely, it connects to the closest node in level $\ell+1$, whose identifier matches $v[\ell]||j$, within the ball $A_{\ell+1}(v)$ (i.e., among the $\alpha B^{\ell+1}M$ closest nodes to v).

For any i, if $C_i(v) \cap A_{\ell+1}(v) = \emptyset$ then node v emulates a *virtual node* u that acts as the $v.L(i)$ endpoint. Node u's id is $u.id = v.id[\ell]||i$ and its level is $u.level = v.level + 1$. The real node that hosts u needs to maintain all the links that virtual node u requires as if u was a regular node (including the *publish* links described below).

Since a virtual node also requires its own neighbor links, it may be that the jth neighbor link of a virtual node u does not exist in $C_j(u) \cap A_{\ell+2}(u)$. In such a case v also emulates a virtual node that acts as the $u.L(j)$ endpoint.

Emulation continues recursively until all links of all the virtual nodes emulated by v are found (or until the limit of M levels is reached).

publish: If $|v.id| = M$ then the publish links in $v.P$ contain all other nodes that have the same identifier and level. Otherwise, the publish links $v.P$ are all the nodes (and virtual nodes) of level $v.level+1$ with the same first $v.level$ bits as $v.id$ which are inside the ball $A_{\ell+3}(v)$. Formally, $v.P = C(v) \cap A_{\ell+3}(v)$, where $C(v) = \{u \in V \mid u.id[\ell] = v.id[\ell], u.level = \ell+1\}$.

closest: The closest link points to the closest supernode: $v.closest = argmin_{s \in S}\{c(v,s)\}$.

4.1 Publish and lookup

The publishing of an object *obj* residing on a supernode t proceeds as follows. Starting with $w_0 = t$, move from a node w_{i-1} to w_i using the neighbor links by fixing the i'th bit to that of $H(obj)$ and moving to level i. Thus, node w_i satisfies: (i) $w_i.level = i$, and (ii) $w.i[i] = H(obj)[i]$. Continue until level M (i.e., until there are no more neighbor links to follow).

Each node w_i along the publishing route stores a reference to *obj* which points back to w_{i-1}. In addition, w_i stores such a reference on every node of w_i's publish links.

A lookup operation of an object $obj \in \mathcal{A}$ can be initiated by any node in the system, and its purpose is to find the supernode storing *obj*. The lookup operation from a node x proceeds in three stages if x is a regular node: The first is a single step to $x.closest$ (this step is skipped for supernodes).

The second stage fixes target bits one by one by moving up levels. The loop goes as follows: So long as the target was not found, then from the current node x_i of level i, first check if there is a reference to *obj* with a link to w_{i-1}. If so, move to w_{i-1} and continue with the third phase. Otherwise, if $H(obj)_{(i+1)}$ is j, continue at $x_i.L(j)$ (this might be a virtual node).

The third stage traverses from w_{i-1} backward to t using *obj*'s reference links.

The publish and lookup algorithms for a node u with level ℓ are provided in pseudo-code in Figure 2 below.

5 Analysis

5.1 Expected Degree

Lemma 5.1 *For every node v, the expected number of virtual nodes it emulates is constant.*

Proof: For any level $\ell \leq M$ the probability that the ith neighbor link will be found inside $A_{\ell+1}(v)$ is at least $1 - \left(1 - \frac{1}{B^{\ell+1}M}\right)^{\alpha B^{\ell+1}M} \geq 1 - e^{-\alpha}$.

For a node v with level $\ell < M$ and index $0 \leq i \leq M - \ell$ let $b_{\ell+i}$ be a random variable that counts the number of virtual nodes that v emulates with level $\ell + i$.

```
A super node $t$ that wants to store an object $obj$ initiates PUBLISH$(obj, t)$.

PUBLISH$(obj, w)$: // a publish request at node $u$ of object $obj$ coming from $w$
    store "for object with hash $H(obj)$ goto node $w$" on node $u$;

    send "for object with hash $H(obj)$ goto node $u$" to every node (possibly virtual) in $u.P$;

    If $u.level < M$
        $u.L(H(obj)_{u.level+1})$.PUBLISH$(obj, u)$; // Follow neighbor link that fixes bit according to $H(obj)$
```

```
A regular node $x$ that wants to lookup object $obj$ initiates LOOKUP$(obj, x)$ on $x.closest$.

LOOKUP$(obj, x)$: // a lookup at node $u$ to an object $obj$ originated from $x$.

    if $u$ stores $obj$
        return $obj$ to $x$;

    else if $u$ stores "for object with hash $H(obj)$ goto node $v$"
        $v$.LOOKUP$(obj, x)$;

    else if $u.level < M$
        $u.L(H(obj)_{u.level+1})$.LOOKUP$(obj, x)$; // Follow neighbor link that fixes bit according to $H(obj)$
```

Figure 2: The PUBLISH and LOOKUP algorithms.

So $b_\ell = 1$ and for $1 \le i \le M - \ell$ each of the $b_{\ell+i-1}$ nodes has B neighbor links with a probability of emulating each one bounded by $e^{-\alpha}$. Therefore $E[b_{\ell+i} \mid b_{\ell+i-1}] \le b_{\ell+i-1}Be^{-\alpha}$ and due to the independence of the identifiers $E[b_{\ell+i}] \le E[b_{\ell+i-1}]Be^{-\alpha}$. Thus by induction $E[b_{\ell+i}] \le (Be^{-\alpha})^i$. The expected total number of virtual nodes emulated by node v is bounded by:

$$E[\sum_{0 \le i \le M-\ell} b_{\ell+i}] \le \sum_{i=0}^{\infty} \left(Be^{-\alpha}\right)^i = \frac{1}{1 - Be^{-\alpha}} .$$

□

Lemma 5.2 *For every node v, the expected number of publish links is constant.*

Proof: Denote j the level of v. The probability that a node is on level $j+1$ (or of level j if $j = M$) and matches the first j bits of v is $1/(B^j M)$.

Further, we should consider the probability that a node emulates a virtual node of level $j+1$ with identifier matching $v[j]$, hence v also has a publish link to it. This probability is determined as follows. First, it is possible that a node of level j has a prefix matching $v[j]$ and does not have a close enough neighbor of level $j+1$ with prefix $v[j]$. As shown in Lemma 5.1 above, this probability is bounded by $(1/B^j M)Be^{-\alpha}$. Next, there might be a node of level $j-1$ with a prefix matching $v[j-1]$ that has neither a close-by node of level j matching $v[j]$, nor a close-by node of level $j+1$ matching $v[j]$. Thus, this node recursively emulates a level $j+1$ node with identifier prefix $v[j]$. The probability for this is at most $(1/B^{j-2}M)B(e^{-\alpha})^2$. And so on.

Putting the above together, the total probability that a node emulates a virtual node with level $j+1$ that matches $v[j]$ is bounded by

$$\sum_{i=0}^{\infty} \frac{1}{B^j M} \left(Be^{-\alpha}\right)^i = \frac{1}{B^j M} \frac{1}{(1 - Be^{-\alpha})} \ .$$

Hence, the expected number of level $j+1$ nodes (possibly virtual) among the $\alpha B^{j+3} M$ nodes that match $v[j]$ is bounded by

$$E[|v.P|] \leq \alpha B^{j+3} M \frac{1}{B^j M} \frac{1}{(1 - Be^{-\alpha})} = \frac{\alpha B^3}{1 - Be^{-\alpha}} \ .$$

□

Corollary 5.3 *The expected number of reference pointers for each object is $O(M) = O(\log n)$*

For the most part, the expected in-degree of nodes is constant by analogous arguments. However, the case of nodes that have links to/from supernodes is different, because the dispersal of supernodes is independent of regular nodes. For example, if there are very many supernodes, then level-1 nodes have high in-degree accordingly, due to neighbor and publish links. And vice-versa, if there are too few supernodes, then the supernodes have high in-degree by *closest* links. Ideally, $|R|/|S| = M$, then the expected in-degree of all regular nodes is constant. Under such as assumption, the following theorem easily follows from Lemma 5.1 and Lemma 5.2 above:

Theorem 1 *Assuming that supernodes are designated with probability $1/M$ at random, then the expected degree of all regular nodes is constant, and the expected degree of all supernodes is $O(M)$.*

5.2 Distortion

In this section we show that the worst case distortion of the lookup operation is constant. For the analysis of a lookup path, we denote the first node performing a lookup of an object A by x, and the (closest) target supernode containing A by t. Let $s = x.closest$.

Denote the sequence of neighbor nodes taken by the routing algorithm as $x_0, x_1, x_2, \ldots$, where $x_0 = s$ and x_i is a level i node. Similarly, let the sequence of publishing nodes taken from t be $t = w_0, w_1, w_2, \ldots$. Note that some of these nodes may be virtual nodes that are emulated by a non-virtual node.

Lemma 5.4 *For every $i \geq 0$, $x_i \in A_{i+1}(s)$ and similarly, $w_i \in A_{i+1}(t)$.*

Proof: By induction on i. For $i = 0$ we have $s = x_0$. Assume by induction that $x_{i-1} \in A_i(s)$. If x_i is emulated by the same node as x_{i-1} then we are done. Otherwise, by Lemma 2.1, $A_{i+1}(s) \supseteq A_i(x_{i-1})$. By construction, $x_i \in A_i(x_{i-1})$, and hence, $x_i \in A_{i+1}(s)$. (The case of w_i and t is identical). □

Corollary 5.5 *For every $i \geq 0$, the total distance of the path from $s = x_0$ through x_i is at most*

$$\frac{2mingrow}{mingrow - 1} a_{i+1}(s)$$

Proof: By Lemma 5.4 for every $0 < j \leq i$, $c(x_{j-1}, x_j) \leq 2a_{j+1}(s)$, and by Lemma 2.1(iv), $a_{j+1}(s) \leq mingrow^{-(i-j)} a_{i+1}(s)$. Hence, the total distance of the path from x_0 through x_i is at most

$$\sum_{j=0}^{i-1} c(x_j, x_{j+1}) \leq \sum_{j=1}^{i} 2a_{j+1}(s) \leq \sum_{j=0}^{i-1} 2mingrow^{-j} \ a_{i+1}(s) \leq 2a_{i+1} \sum_{j=0}^{\infty} mingrow^{-j} \leq \frac{2mingrow}{mingrow - 1} a_{i+1} \ .$$

□

Lemma 5.6 *Let k be an index such that $t \in A_k(s)$. Then x_k contains a reference to A.*

Proof: From Lemma 5.4, $w_{k-1} \in A_k(t)$. From Lemma 2.1, we have $A_{k+1}(s) \supseteq A_k(t)$ which implies that $w_{k-1} \in A_{k+1}(s)$. From Lemma 2.1, we have $A_{k+1}(s) \subseteq A_{k+2}(w_{k-1})$. Applying Lemma 5.4 again, $x_k \in A_{k+1}(s)$ thus $x_k \in A_{k+1}(w_{i-1})$. Since w_{k-1} is a level $k-1$ node it publishes a reference for object A in all the nodes with a prefix $w_{k-1}[k-1]$ of level k in the ball $A_{k+2}(w_{k-1})$ thus x_k must contain a reference of the type "for object with hash $H(A)$ goto node w_{k-1}". □

Let k be the first index such that $t \in A_k(s)$. By Lemma 5.6, we know that when the lookup path reaches x_k, it proceeds to $w_{k-1}, \ldots, w_0 = t$. It is left to see what is the total distance of the route $x, s = x_0, x_2, \ldots, x_k, w_{k-1}, w_{k-2}, \ldots, w_0 = t$.

Theorem 2 *The distortion of the path from x to t is constant.*

Proof: The first step of the route is from node x to its closest supernode s. By definition then, $c(x, s) \leq c(x, t)$.

The next part of the route is the path from s up to x_k. We now make use of the assumption that $t \notin A_{k-1}(s)$, hence $a_{k-1}(s) < c(s, t)$. From Lemma 2.1 (iii), $a_{k+1}(s) \leq maxgrow^2 a_{k-1}(s) < maxgrow^2 c(s, t)$. Putting together with Corollary 5.5, the length of the route from s to x_k is bounded by

$$\sum_{j=0}^{k-1} c(x_j, x_{j+1}) \leq \frac{2mingrow}{mingrow - 1} a_{k+1}(s) \leq \frac{2mingrow\ maxgrow^2}{mingrow - 1} c(s, t) \ .$$

By Lemma 5.4, $c(t, w_{k-1}) \leq a_k(t)$, by Lemma 2.1, $a_k(t) \leq a_{k+1}(s)$, and by Lemma 5.4 $x_k \in A_{k+1}(s)$. Hence,

$$c(x_k, w_{k-1}) \leq c(x_k, t) + c(t, w_{k-1}) \leq 2a_{k+1}(s) + a_{k+1}(s) \leq 3maxgrow^2 c(s, t) \ .$$

The third and last phase of the route is the traversal from $w_{k-1}, w_{k-2}\ldots$ back to $w_0 = t$. We get that the total distance of the path is bounded by $\frac{2mingrow}{mingrow-1} a_k(t)$. Since $a_k(t) \leq a_{k+1}(s)$, and by similar analysis as above, the cost of this route is bounded by

$$\sum_{j=0}^{k-2} c(w_j, w_{j+1}) \leq \frac{2mingrow}{mingrow - 1} a_{k+1}(s) \leq \frac{2mingrow\ maxgrow^2}{mingrow - 1} c(s, t) \ .$$

Hence, the total distortion is bounded by

$$1 + \frac{4mingrow\ maxgrow^2}{mingrow - 1} + 3maxgrow^2 \ .$$

□

6 Variations

In this section, we briefly sketch a number of variations on the basic problem model and its solutions. Full details of the various constructions will be provided in the full paper.

6.1 Homogenous nodes

In this section, we examine the principles of our construction in an environment containing homogeneous nodes only. This investigation is motivated by applications such as web caching. In this settings, it is not clear whether two-tier architectures are justified. This setting is the one presented by [14, 7], and thus, our results are directly comparable to theirs.

There are two possible approaches to accommodate homogeneous nodes. The first one simply allows placing data objects on all nodes, without further changes to the routing network. Lookup works without modification in this network, and its properties are almost unchanged. The only issue worth noting is the following. Since both the lookup procedure and the publishing procedure need to start with a node at level 0, this approach entails an inherent, minimal routing cost of approaching the closest level-0 node. This cost is not high, as one out of M nodes is expected to have level 0. Unless the distance between the initiating node and target point is very small, this additional cost would be unnoticeable.

Further, obvious balancing methods may be used to increase the proportion of nodes with level 0 in the network. Alternatively, additional information may be maintained by nodes at a reasonable cost about their proximity neighbors. This could be done so as to as to short-cut directly to close-by targets.

The second homogeneous approach is to increase the expected degree to $O(M) = O(\log n)$. This is done as follows. Each node has one identity but multiple levels, from 0 to M. As a consequence, nodes do not need their closest links anymore.

The links of node v include the $L(i)$ neighbors and publish links corresponding to all possible levels $0..M$. We denote the set of neighbor links for level ℓ by $\{v.L_\ell(i)\}$, where $0 \leq \ell \leq M, 0 \leq i \leq B-1$. The only difficult point to note is the emulation of virtual nodes. Say that node v finds no suitable target for one of its level-ℓ links $v.L_\ell(i)$. Then v emulates virtual node $v.L_\ell(i)$. However, we do not need for v to emulate $v.L_\ell(i)$ at all levels, because the virtual node exists only for the purpose of routing the phase ℓ of a lookup. Hence, virtual nodes have only one level (as in our original construction). More specifically, virtual node $u = v.L_\ell(i)$ has only neighbors $\{u.L_{\ell+1}(i)\}_{i=0..B-1}$ and level $\ell+2$ publish links.

The subtlety concerning virtual nodes is significant, as it indicates that LAND and the construction in [14] are not isomorphic in any way: A simple coallescing/separation of logarithmic clusters of nodes would not map between the two.

This subtlety is also crucial for maintaining the expected out-degree of nodes logarithmic in the homogenous construction. As before, the distortion is worst-case constant.

6.2 Content Addressable Networks (CAN)

In various DHTs, e.g., [15, 16, 11], each data object is moved to the node that owns its virtual name, rather than placing a reference to it there. In our terminology, if an object is named A, it is placed on the node y whose id has the longest matching prefix to $H(A)$. Thus, the lookup procedure looks for $H(A)$ directly, and once it is reached, no reference links are needed.

Our approach may support a CAN as follows: In a two-tier architecture, objects may reside only on supernodes. Thus, the target of each route (in the form $H(A)$) must be a supernode. There are no publish links, but rather, a node v of level ℓ chooses *direct* links to all supernodes in $A_{\ell+2}(v) \cap C(v)$, where $C(v) = \{u \mid u[\ell] = v[\ell]\}$. In words, direct links connect directly to supernodes in $A_{\ell+2}(v)$ whose first ℓ identifier digits match v's. Direct links are used for short-cutting a route directly to the supernode holding the object. More precisely, starting the lookup from a node x, let k be the first index such that $y \in A_k(x)$. Then after k steps, a node x_k on level k is reached that has a direct link to y. Analysis similar to the one above shows that this CAN construction has expected constant degree and maximum constant distortion.

In a homogeneous setting, we may support a CAN using the homogenous construction above with logarithmic expected degree.

6.3 Balancing supernodes

In order for the network to maintain its constant expected degree and logarithmic number of reference copies, supernodes need to constitute $1/M$ of each neighborhood. One obvious manner for keeping this balance is to assign supernode privileges to nodes at random with probability $1/M$.

However, in many situations, supernodes are restricted by additional criteria, such as throughput and uptime. In this case, if there are fewer supernodes than needed, then regular nodes may be allowed to assume level 0 (with some skewed probability), so that the in-degree of supernodes remains an expected logarithm. Conversely, if the ratio of supernodes to nodes exceeds $1/M$, the supernodes may be arranged in a hierarchy as follows. We keep only B^M supernodes at level 0, with all their links. Thus, regular nodes at level 1 have constant expected in-degree. We place the remaining super nodes in decreasing 'levels' $-1, -2, ..$, e.g., by putting B^M supernodes at each negative level. Nodes with negative levels have links to nodes one level higher, in a similar manner to positive levels. Closest links point from regular nodes to their closest supernode at any non-positive level. Thus, routing may start at a negative level. Likewise, publishing from a supernode holding an object starts with the level of that supernode. Consequently, routing may be lengthened in number of hops by twice the number of additional levels. This scheme keeps an expected constant degree of the network for any number of supernodes.

7 Dynamic node arrivals and departures

In this section we sketch how nodes may dynamically arrive and depart from the system. We assume that the rate of changes in the regular node population will be substantially greater than the rate of change in the supernode population. Thus, specially for regular nodes, the arrival and departure protocols should cause a minimal amount of network overhead.

We assume that once two nodes x, y connect (by x sending a message that arrives to y) they may exchange messages and discover the real distance $c(x, y)$ between them. In addition, new nodes arriving to the system initially know as an access point the closest supernode in the system. These are the same assumption as in, e.g., [17].

7.1 Node arrival

When a new node arrives to the system it needs to do three things: (1) acquire an id, (2) acquire a level, and (3) establish network links. Note that since regular nodes do not store content (only supernodes store objects), no transfer of content is necessary upon arrival.

Acquiring an identifier and level Each node chooses an identifier of M uniformly random radix B digits. In addition, each node chooses a random real number $r \in (0, 1]$. The level of a node will be $\lceil rM \rceil$. Note that due to a significant change in the number of nodes, the parameter M may change. In such a case, the level of a node may change, and if needed a uniformly random B digit is appended to its identifier.

Establishing network links Once the id and level of a node is known, it is left with the task of establishing links as defined in Section 4.

For a node x with level ℓ, the main difficulty is to find all the level $\ell + 1$ nodes with prefix $x.id[\ell]$ in the ball $A_{\ell+3}(x)$. Node x also needs to inform all nodes y of level $\ell - 1$ with prefix $x.id[\ell - 1]$ such that $x \in A_{\ell+2}(y)$. This can be done, again, by finding all nodes y in $A_{\ell+3}(x)$ with prefix $x.id[\ell - 1]$.

The LOCATE algorithm for a node v on level ℓ is as follows. Let $s = v.closest$ (by assumption, s is given to v at joining time). For every combination of digits $d_1, d_2 \in [0..B - 1]$, route from s to a node y such that $y.id[\ell + 2] = v.id[\ell]||d_1||d_2$. This routing is done in an identical manner to the routing phase of LOOKUP in Figure 2, i.e., using the $L(i)$ links. Let Y denote the set of nodes y reached by this procedure. By construction, every $y \in Y$ has level $\ell + 2$. Let $S(\ell + 3)$ be the set of level $\ell + 3$ nodes that appear as publish links of nodes in Y, i.e., $S(\ell + 3) = \bigcup_{y \in Y} y.P$. Obtain $S(\ell + 2)$ by taking all incoming publish links into Y from nodes of level $\ell + 2$. Then recursively, obtain $S(\ell + 1)$ by taking all publish links going into $S(\ell + 2)$. And so on, until we have $S(\ell - 1)$. From $S(\ell + 1)$, node v selects neighbor links whose distance from v does not exceed $a_{\ell+1}$, and keeps publish links whose distance from v does not exceed $a_{\ell+3}$. Then v informs nodes in $S(\ell - 1)$ about its arrival.

The LOCATE algorithm in depicted in pseudo-code in Figure 3.

```
LOCATE: // invoked by node $v$ of level $\ell$
    for every combination of digits $d_1, d_2 \in B$
      $v.closest.$SEARCH$(v.id[\ell]||d_1||d_2, \ell, v)$;

    wait for replies, accumulate in $SET$;
    set $v.L(i) = min_{u \in SET}\{u.level = \ell + 1 \wedge u.id[\ell + 1] = v.id[\ell]||i \wedge c(v, u) \leq a_{\ell+1}\}$;
    // emulate $v.L(i)$ if empty
    set $v.P = \{u \in SET \mid u.level = \ell + 1 \wedge u.id[\ell] = v.id[\ell] \wedge c(v, u) \leq a_{\ell+3}\}$;
    inform all level $\ell - 1$ nodes in $SET$ about $v$'s arrival;

SEARCH$(prefix, \ell, v)$: // search at node $u$ invoked by locate from node $v$

    if $u.level = \ell + 2$
      INLINKS$(v, 3)$; // return all in-links 3 levels backward

    else $u.L(prefix_{u.level+1})$.SEARCH$(prefix, \ell, v)$; // route on by fixing one more digit

INLINKS$(v, r)$: // inlinks at node $u$ invoked by locate from node $v$
    // recurse for $r$ levels searching for in-links

    let $I$ denote the incoming publish links of $u$ from level $u.level - 1$;
    send $v$ the set $I$;
    if $(r > 1)$
      for each $w \in I : w.$INLINKS$(v, r - 1)$;
```

Figure 3: The LOCATE algorithm.

7.2 Node departure

When a regular node x of level ℓ leaves the network, the level $\ell - 1$ nodes whose neighbor link contained x need to be updated, and x removed from their list. If x was a neighbor link of a node v, then v's next closest publish link becomes the neighbor link, unless this link is too far away in which case v emulates a virtual node. The links for this emulation are acquired using the LOCATE algorithm.

7.3 Supernode arrival and departure

In addition to being a regular node and a virtual level-0 node, supernodes must also discover all the closest regular nodes and change their *closest* pointers. Finding the closest nodes can be done by a depth bounded breadth first search.

When a supernode s is removed, the regular nodes that pointed to s as their *closest* must update their pointers. This can be done by a depth bounded breadth first search, using $L(i)$ links.

7.4 Fault tolerance

In this section we sketch how the LAND architecture has inherent fault tolerance in its network links. If a *neighbor* link of a node fails the node can immediately continue to send routing requests by using the closest *publish* link.

Using the publish links, a node can even initiate k multiple lookup paths on different links in parallel. Thus even if $k-1$ paths fail due to $k-1$ faulty nodes, one of the lookup paths will reach the supernode with the same id as the object hash. Note that in such a case the distortion may be higher, and constant distortion is not archived.

7.5 Analysis of the Locate algorithm

Lemma 7.1 *For a node x with level ℓ, Algorithm* LOCATE *finds all the appropriate level $\ell+1$ nodes in $A_{\ell+3}(x)$ and all appropriate nodes w with level $\ell-1$ such that $x \in A_{\ell+2}(w)$.*

Proof: Let $s = x.closest$. By applying Lemma 2.1, $A_{\ell+3}(x) \subseteq A_{\ell+4}(s)$. By Lemma 5.4 each node y that is routed to in algorithm LOCATE is in $A_{\ell+3}(s)$. In addition, each such node has level $\ell+2$ so its publish links cover all appropriate level $\ell+1$ nodes in $A_{\ell+2+3}(y)$. Applying Lemma 2.1 again we get $A_{\ell+3}(x) \subseteq A_{\ell+4}(s) \subseteq A_{\ell+5}(y)$.

For every w such that $x \in A_{\ell+2}(w)$, we have that $A_{\ell+2}(w) \subseteq A_{\ell+3}(x)$. By a similar reasoning to the above, we find all such w's. $\square$

Lemma 7.2 *For a node arrival:*
(i) The expected number of nodes that change their state is constant.
(ii) The expected number of messages sent is $O(\ell)$

Proof: Routing to each of the level $\ell+2$ nodes takes $\ell+2$ messages.

Once such a node is reached, a message is sent to each of its links. The expected number of links is constant (see Theorem 1), and all choices are independent. Thus recursively finding all appropriate links for node x will cause sending an expected constant number of messages.

The number of nodes that changes their state is the number of appropriate nodes in $A_{\ell+3}(x)$, the expected number of these nodes is constant. $\square$

8 Conclusions

LAND is the first peer-to-peer network and lookup algorithm that has worst case constant distortion. The network is a variation on the access scheme of [14], yet whereas PRR use a logarithmic number of links and achieve an expected distortion, our construction uses a constant expected number of links and achieves constant distortion in the worst case.

The deployment of algorithms such as [14] and LAND remains an active area of research, e.g., in [17, 3]. We envision that many of the insights and optimizations in these systems may be applied for a real-life implementation of LAND.

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 *International Parallel and Distributed Processing Symposium* (IDPDS 2003), April 2003, Nice, France.

[2] R. Chand and P. Felber. "A Scalable Protocol for Content-Based Routing in Overlay Networks". In IEEE International Symposium on Network Computing and Applications (NCA'03), Cambridge, MA, April 2003.

[3] P. Drushel and A. Rowstron. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". *Proceeding of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),* November 2001.

[4] P. Fraigniaud and P. Gauron. “The Content-Addressable Network D2B”. Technical Report 1349, LRI, Univ. Paris-Sud, France, January 2003.

[5] F. Kaashoek and D. R. Karger. “Koorde: A Simple Degree-optimal Hash Table”. In *2nd International Workshop on Peer-to-Peer Systems* (IPTPS ’03), February 2003, Berkeley, CA.

[6] 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”. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 654–663, May 1997.

[7] D. R. Karger and M. Ruhl. “Finding Nearest Neighbors in Growth-restricted Metrics”. ACM Symposium on Theory of Computing (STOC ’02), Montreal, May 2002.

[8] X. Li and C. G. Plaxton. “On name resolution in peer-to-peer networks.” In *Proceedings of the 2nd ACM Worskhop on Principles of Mobile Commerce* (POMC), pp. 82–89, October 2002.

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

[10] D. Malkhi. “Dynamic Lookup Networks: A position paper”. In *Proceedings of the International Workshop on Future Directions in Distributed Computing* (FuDiCo), LNCS Volume 2584, pp. 93–96. Bertinoro, Italy, 2002,

[11] D. Malkhi, M. Naor and D. Ratajczak. “Viceroy: A scalable and dynamic emulation of the Butterfly”. In *Proceeding of the 21 st ACM Symposium on Principles of Distributed Computing (PODC’02)*, July 2002.

[12] G.S Manku, M. Bawa, P. Raghavan. “Symphony: Distributed Hashing In Small World”. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[13] M. Naor and U. Wieder. “Novel Architectures for P2P Applications: the Continuous-Discrete Approach”. In *The Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA ’03)*, 2003.

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

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

[16] I. Stoica, R. Morrise, 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*, August 2001.

[17] B.Y. Zhao, J. D. Kubiatowicz and A.D. Joseph. “Tapestry: An infrastructure for fault-tolerant wide-area location and routing”. U.C. Berkeley Technical Report UCB/CDS-01-1141, April, 2001.