10 Locality-aware overlay networks

One of the main challenges of distributed systems is how to efficiently store and locate ever increasing amounts of content. The solution suggested by Karger et al. [3] is to use consistent hashing. We showed in the previous section how this consistent hashing method can be integrated into a decentralized overlay network to result in a dynamic, distributed hash table (or DHT) with low dilation and congestion. However, just minimizing the number of hops may not be a good idea in general since 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 [2, 7, 4, 5, 6] 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 peers. Given such a cost function, our goal is to design an overlay network so that the ratio between the direct cost of a source-destination pair and the cost of a route for it in the overlay network is as close as possible. To measure this ratio, we use a parameter called *stretch*. This is defined as follows.

Let u be a lookup starting point and w be the target of the lookup (i.e., the closest node containing the searched object). Let $u = v_1, v_2, \ldots, v_k = w$ be the nodes traversed in the lookup route. Then the *stretch* of that route is defined as $[c(v_1, v_2) + \ldots + c(v_{k-1}, v_k)]/c(u, w)$ and the stretch of the overlay network is the maximum over all pairs $(u, w) \in V^2$ of the minimum achievable stretch for (u, w) in that overlay network.

The seminal work of Plaxton, Rajaraman, and Richa [5] is one of the first works to provide a distributed lookup protocol with analytical bounds on the stretch. They present a randomized scheme for a class of metric spaces representing realistic networks, in which the expected stretch in finding targets is a (rather large) constant. Several efforts were made to deploy this scheme, e.g., Tapestry [7] and Pastry [2]. These systems construct a dynamic DHT based on the principles of [5], yielding efficient, locality-aware overlay networks. The best result obtained so far in this line of research is due to Abraham, Malkhi, and Dobzinski [1]. They present a $1 + \epsilon$-stretch DHT called *LAND* in which for all routes the cost ratio between the distance and the route is guaranteed to be at most $1 + \epsilon$. The guarantee of constant stretch is achieved while not impairing on other parameters of the network such as degree and memory requirements. We will present the LAND scheme in detail in this section.

10.1 Preliminaries

Let V denote the set of nodes in the system, where $|V| = n$, and let $c : V^2 \to \mathbb{R}_+$ be a function expressing the cost of communicating directly between any pair of nodes in V. We assume that c is positive, reflexive, symmetric and satisfies the triangle inequality. Thus, (V, c) forms a metric space. From here on, we refer to the cost as the *distance* between nodes. The set of nodes within distance r from node v is denoted $N(v, r)$.

We assume that the minimal distance between every pair of nodes is 1. Growth bounded metrics are considered (e.g., [10]) as a realistic model for the Internet. For *growth bounded metrics*, we assume a parameter Δ such that for every node v and $r \geq 1$ we have

$$|N(v, 2r)| \leq \Delta |N(v, r)| \ .$$

Each node u hosts an assembly of *routing entities* (or *routers*). A router r hosted by node u is denoted $u.r$. Each router is identified by a string $r.id$. When it is clear from the context, we sometimes refer to $u.r$ simply as r. Identifier strings are composed of M digits in radix $B = 2b$. 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 $M = \lceil x \rceil$ for x satisfying $B^x = n$. A router $u.r$ has an additional level property, denoted $u.r.level$, between 1 and $M + 1$.

Let s be a k-digit identifier. Denote $s[j]$ as the prefix of the j most significant digits, and denote s_j as the jth digit of s. A concatenation of two strings s, s' is denoted by $s \circ s'$.

For convenience, all definitions used in the LAND construction (including those mentioned above already) are summarized here.

- The radix B satisfies $B \geq \Delta^2$.
- The number of identifier-digits satisfies $M = \lceil x \rceil$ for x so that $B^x = n$.
- The constant α is chosen such that $Be^{-\alpha} < 1$.
- $A_i(v)$ denotes the smallest ball around node v containing $\min(\alpha B^i, n)$ nodes, and $a_i(v)$ denotes the radius of $A_i(v)$.
- Let $\gamma = B^{\log_\Delta 2}$, and note that $\gamma \geq 4$.
- For the desired stretch ϵ, choose d so that

$$\epsilon \geq \frac{1}{\gamma^d}\left(\frac{2\gamma}{\gamma - 1} + 2 + \frac{1}{\gamma} + \frac{1}{\gamma - 1}\right) .$$

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 10.1 *Let v and w be any two nodes. For any i with $w \in A_i(v)$:*

(i) $A_i(v) \subseteq A_{i+1}(w)$

(ii) $A_i(w) \subseteq A_{i+1}(v)$

(iii) $a_{i+1}(v) \geq \gamma a_i(v)$, *where* $\gamma = B^{\log_\Delta 2}$

Proof. Let $r = a_i(v)$ denote the radius of $A_i(v)$, so $A_i(v) = N(v, r)$ where $N(v, r) = \{u \in V \mid c(u, v) \leq r\}$. Since $w \in N(v, r)$ we get

$$N(v, r) \subseteq N(w, 2r) \subseteq N(v, 3r)$$

and from the growth bounded assumption it follows that

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

For (i), $N(v, 3r) \subseteq A_{i+1}(v)$, and so $A_{i+1}(w)$, the ball around w with αB^{i+1} nodes, must contain $N(w, 2r)$ and so must contain $A_i(v)$. For (ii), $A_i(w) \subseteq N(w, 2r) \subseteq N(v, 3r) \subseteq A_{i+1}(v)$. For (iii), notice that $\gamma = B^{\log_\Delta 2} = 2^{\log_\Delta B}$ and, hence, $|N(v, B^{\log_\Delta 2} r)| \leq \Delta^{\log_\Delta B} |N(v, r)| \leq \alpha B^{i+1}$. Therefore, $A_{i+1}(v) \supseteq N(v, \gamma r)$, which implies that $a_{i+1}(v) \geq \gamma a_i(v)$. $\square$

10.2 The LAND architecture

Our goal is to support a lookup operation that locates nearest copies of objects such that the nodes of the network share the lookup load evenly. All nodes in the network take part in the lookup process and pass queries to the nodes they have outgoing links to. (Copies of) objects may be stored at any node, but references to objects are mapped to nodes with the help of the distributed hash table (DHT) approach. More formally, let A be a set of objects. We make use of a pseudo-random hash function h that maps object names to identifiers in $[B]^M$ uniformly at random. For any (copy of an) object $obj \in A$ being stored in some node s, reference information about obj's location is stored in nodes whose identifiers match prefixes of various length of $h(obj)$. The object obj may be replicated in various locations in the network. As it will turn out, the LAND network is able to locate a nearby copy of every $obj \in A$ from any node in the network.

The basic entity in the LAND network is the router entity (simply referred to as 'router' in the following). Each router has an identifier and a level. The routers are connected in a manner similar to the butterfly graph, i.e., level ℓ routers have outgoing links only to level $\ell + 1$ routers. In LAND, each node in the system initially maintains $M + 1$ routers, each with a different level from 1 to $M + 1$. In addition, it is possible that some nodes will need to maintain additional shadow routers to ensure worst case locality. When we mention a router r, we interchangeably mean either the routing entity itself or the node that hosts the router. The precise meaning will be evident from the context.

More formally, each node v hosts an initial set of $M + 1$ routers denoted $R_1, ..., R_{M+1}$. Router $v.R_i$ has identity $v.R_i.id$ and level i, i.e., $R_i.level = i$. The identifiers of routers are represented as radix $B = 2b$ numbers. The identifiers of the initial routers are chosen uniformly and independently at random.

Let r be a router of level $r.level = \ell$ hosted by node v. The router r could be either the initial $v.R_\ell$, or a shadow router hosted by v, as we shall see below. Router r has outgoing links of two types, neighbor and publish links, denoted $r.L$ and $r.P$ respectively. These outgoing links are defined as follows:

Neighbor links

If $\ell \leq M$, then router r has B neighbor links, denoted $L(0), \ldots, L(B-1)$. The ith neighbor $r.L(i)$ is selected as the closest router within $C_i(r) \cap A_\ell(v)$ where

$$C_i(r) = \{u \in V \mid \exists s \text{ so that } u.s.id[\ell] = r.id[\ell-1] \circ i \text{ and } u.s.level = \ell + 1\} \ .$$

The link $L(i)$ fixes the ℓth digit of $r.id$ to i, namely, it connects to the closest node u that hosts a level $\ell + 1$ router $u.s$ that matches the id $r.id[\ell - 1] \circ i$ within the ball $A_\ell(v)$ (i.e., among the αB^ℓ closest nodes to v).

If $C_i(r) \cap A_\ell(v) = \emptyset$, then node v hosts a shadow router s with identifier $r.id[\ell - 1] \circ i$ and level $\ell + 1$. Node v maintains all of the links of the shadow router (including the publish links described below). Since a shadow router also requires its own neighbor links, it may be that the jth neighbor link of a shadow router s does not exist in $C_j(s) \cap A_{s.level}(v)$. In such a case v also hosts a shadow router that acts as the $s.L(j)$ endpoint. Shadow hosting continues recursively until all links of all the shadow routers hosted by v are found (or until the limit of $M + 1$ levels is reached).

Publish links

If $\ell \leq M$, the publish links $r.P$ are all the nodes hosting any level-$(\ell+1)$ router with the same first $\ell-1$ bits as $r.id$ which are inside the ball $A_{\ell+d+5}(v)$. Formally, $r.P = C(r) \cap A_{\ell+d+5}(v)$ where

$$C(r) = \{u \in V \mid \exists s \text{ so that } u.s.id[\ell-1] = r[\ell-1] \text{ and } u.s.level = \ell+1\} .$$

Publish and lookup

The publishing of an object obj residing on a node t proceeds as follows. Starting with a level-1 router $w_1.r$ (where $w_1 = t$), move from a node w_i using the neighbor links of the level i router $w_i.r$ by fixing the ith digit to that of $h(obj)$. This links to w_{i+1}, a node hosting a level $i+1$ router $w_{i+1}.r$ (this might be a shadow router in which case $w_i = w_{i+1}$) such that $w_{i+1}.r.id[i] = h(obj)[i]$. Thus, router $w_{i+1}.r$ has level $w_{i+1}.r.level = i+1$, and id $w_{i+1}.r.id[i] = h(obj)[i]$. Continue until the Mth digit (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 $w_i.r.P = C(r) \cap A_{i+d+5}(w)$.

A lookup operation of an object $obj \in A$ can be initiated by any node in the system, and its purpose is to find the closest node storing obj. The lookup operation from a node v proceeds in two stages. The first stage fixes target digits one by one. The loop goes as follows: Starting with a level-1 router at v denoted $v_1.r$, and so long as the target was not found, then from the current router $v_i.r$, first check if there is a reference to obj with a link to a node w_{i-1}. If so, move to w_{i-1} and continue with the second stage. Otherwise, continue at a node v_{i+1} with a router $v_{i+1}.r$ such that $v_{i+1}.r.level = i+1$ and $v_{i+1}.r.id[i] = h(obj)[i]$ (this might be a shadow router). The second stage traverses from w_{i-1} backward to t using obj's reference links.

The publish and lookup algorithms for a router u are provided in pseudo-code in Figure 1.

A node t that wants to store an object obj initiates $t.R_1.publish(obj, t, 1)$.

publish (obj, w, ℓ) **at router** $u.r$**:**
store "$obj; w$" on node u;
send "$obj; u$" to every node in $u.r.P$;
if $\ell \leq M$ then $u.r.L(h(obj)_\ell)$.publish$(obj, u, \ell+1)$;

A node v that wants to lookup object obj initiates $v.R_1.lookup(obj, v, 1)$.

lookup (obj, v, ℓ) **at router** $u.r$**:**
if u stores obj return obj to v;
else if u stores "$obj; w$" then w.lookup(obj, v, ℓ);
else if $\ell \leq M$ then $u.r.L(h(obj)_\ell)$.lookup$(obj, v, \ell+1)$;

Figure 1: The publish and lookup algorithms.

10.3 Analysis

We first bound the expected degree and then the stretch of the LAND construction.

Expected degree

Lemma 10.2 *For every initial router r hosted by a node u the expected number of shadow routers hosted by u due to r is constant.*

Proof. For any level $1 \leq \ell \leq M$, the probability that link $L(i)$ will be found inside $A_\ell(u)$ is at least

$$1 - \left(1 - \frac{1}{B^k}\right)^{\alpha B^k} \geq 1 - e^{-\alpha} .$$

For $0 \leq i \leq M - \ell$, let $b_{\ell+i}$ be a random variable that counts the number of level-($\ell + i$) shadow routers that u recursively emulates due to missing links. Such shadow routers are created if u incurs emulation of a level-($\ell + 1$) shadow router; one of that shadow router's links is also emulated by a level-($\ell + 2$) shadow router; and so on, up to level $(\ell + i)$. So $b_\ell = 1$, and for $1 \leq i \leq M - \ell$, each of the $b_{\ell+i-1}$ routers has B neighbor links with a probability of emulating each one bounded by $e^{-\alpha}$. Therefore,

$$\mathrm{E}[b_{\ell+i} \mid b_{\ell+i-1}] \leq b_{\ell+i-1} B e^{-\alpha}$$

and due to the independence of the identifiers,

$$\mathrm{E}[b_{\ell+i}] \leq \mathrm{E}[b_{\ell+i-1}] B e^{-\alpha} .$$

Thus, by induction, $\mathrm{E}[b_{\ell+i}] \leq (Be^{-\alpha})^i$. This implies that the expected total number of shadow routers incurred by router r is bounded by

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

□

Lemma 10.3 *For every router r the expected number of publish links $|r.P|$ is constant.*

Proof. Consider any router r of some node u and let $r.level = \ell$. The probability that a node v hosts an initial level-($\ell + 1$) router $v.r'$ that matches the first $\ell - 1$ bits of $r.id$ is at most $B^{-(\ell-1)}$.

Further, we need to consider the probability that a node v emulates a shadow router of level $(\ell + 1)$ with identifier matching $r.id[\ell - 1]$, hence r also has a publish link to it. Using the same arguments as in the proof of Lemma 10.2 above, for $0 \leq i \leq \ell$, the probability that a node v has a level-($\ell + 1 - i$) router with identifier-prefix $r.id[\ell - 1 - i]$ and needs to emulate a level-($\ell + 1$) shadow router with prefix $id[\ell - 1]$ (i.e., emulate recursively to depth i) is bounded by $B^{-(\ell-i-1)} e^{-i\alpha}$.

Hence, the total probability that a node hosts a level-($\ell + 1$) router (real or shadow) matching $r.id[\ell - 1]$ is bounded by

$$\sum_{i=0}^{\infty} \frac{1}{B^{\ell-1}} (Be^{-\alpha})^i = \frac{1}{B^{\ell-1}} \frac{1}{(1 - Be^{-\alpha})} .$$

Thus, the expected number of nodes among the $\alpha B^{\ell+d+5}$ nodes that satisfy this criterion is bounded by

$$\mathrm{E}[|u.P|] \leq \alpha B^{\ell+d+5} \frac{1}{B^{\ell-1}} \frac{1}{(1 - Be^{-\alpha})} = \frac{\alpha B^{d+6}}{1 - Be^{-\alpha}} .$$

□

As an immediate consequence of the above two lemmas, we get the following theorem.

Theorem 10.4 *The expected degree of all nodes is $O(M) = O(\log n)$.*

Corollary 10.5 *The expected number of reference pointers for each object is $O(M) = O(\log n)$.*

Stretch

Next we show that the worst case stretch of the lookup operation is $1 + \epsilon$. For the analysis of a lookup path, we denote the first node performing a lookup of an object obj by s, and the (closest) target node containing obj by t. Denote the sequence of steps taken by the routing algorithm as $v_1, v_2, v_3, \ldots$ where $v_1 = s$. Denote the relevant routers as $v_1.r, v_2.r, v_3.r, \ldots$ where $v_i.r.level = i$ and $v_i.r.id[i-1] = h(obj)[i-1]$. Similarly, let the sequence of publishing nodes taken from t be $t = w_1, w_2, w_3, \ldots$ and the sequence of relevant routers be $t = w_1.r, w_2.r, w_3.r, \ldots$ Hence, $w_i.r.level = i$ and $w_i.r.id[i-1] = h(obj)[i-1]$. Note that some nodes may repeat within this sequence due to shadow-router emulation. For ease of notation, we use below $v_0 = v_1 = s$.

Lemma 10.6 *For every $i \geq 1$, $v_i \in A_i(v_{i-1}) \subseteq A_{i+1}(s)$ and similarly, $w_i \in A_i(w_{i-1}) \subseteq A_{i+1}(t)$.*

Proof. By induction on i. For $i = 1$ we have $s = v_1$. Assume by induction that $v_{i-1} \in A_i(s)$. If $v_i.r$ is emulated then $v_i = v_{i-1}$ and we are done. Otherwise, by Lemma 10.1(ii), $A_{i+1}(s) \supseteq A_i(v_{i-1})$. By construction, $v_i \in A_i(v_{i-1})$, and hence, $v_i \in A_{i+1}(s)$. (The case of w_i and t is identical). □

Lemma 10.7 *For every $i \geq 1$, the total distance of the path from $s = v_1$ through v_i is at most*

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

Proof. By Lemma 10.6, for every $1 < j \leq i$, v_j is in the ball $A_j(v_{j-1})$ that is fully contained in the ball $A_{j+1}(s)$. Hence, its radius is at most $a_{j+1}(s)$, and therefore $c(v_{j-1}, v_j) \leq a_{j+1}(s)$.

By Lemma 10.1(iii), $a_{j+1}(s) \leq \gamma^{-(i-j)} a_{i+1}(s)$. Hence, the total distance of the path from v_1 through v_i is at most

$$\begin{aligned} \sum_{j=1}^{i-1} c(v_j, v_{j+1}) &\leq \sum_{j=1}^{i} a_{j+1}(s) \\ &\leq \sum_{j=0}^{i-1} \gamma^{-j} a_{i+1}(s) \leq \frac{\gamma}{\gamma - 1} a_{i+1}(s) \ . \end{aligned}$$

□

Lemma 10.8 *Let k be the first index such that $s \in A_{k+d+2}(t)$. Then v_k contains a reference to obj.*

Proof. From Lemma 10.6, $v_k \in A_{k+1}(s)$. Applying Lemma 10.1(ii) on $s \in A_{k+d+2}(t)$ gives $A_{k+d+2}(s) \subseteq A_{k+d+3}(t)$. Now, from Lemma 10.6, $w_{k-1} \in A_k(t)$. Applying Lemma 10.1(i) on $w_{k-1} \in A_{k+d+3}(t)$ gives $A_{k+d+3}(t) \subseteq A_{k+d+4}(w_{k-1})$. Combining the above yields $v_k \in A_{k+d+2}(s) \subseteq A_{k+d+3}(t) \subseteq A_{k+d+4}(w_{k-1})$. Router $w_{k-1}.r$ has publish links such that it publishes a reference for object obj in all the nodes within the ball $A_{k+d+4}(w_{k-1})$ containing a level-k router whose identifier matches the prefix $w_{k-1}.r.id[k-1]$. Thus, v_k must contain a reference of the type "$obj; w_{k-1}$". □

Using Lemma 10.8, we know that when the lookup path reaches v_k, it proceeds to $w_{k-1}, \ldots, w_1 = t$. It is left to see what is the total distance of the route $s = v_1, v_2, v_3, \ldots, v_k, w_{k-1}, w_{k-2}, \ldots, w_1 = t$.

Theorem 10.9 *The stretch of the path from s to t is $1+\epsilon$.*

Proof. The first phase of the route is the path from $s = v_1$ to v_k. Since $s \notin A_{k+d+1}(t)$, it holds that $c(s,t) > a_{k+d+1}(t)$ and therefore $N(s, 2c(s,t)) \supseteq A_{k+d+1}(t)$. This implies that $A_{k+d+1}(s) \subseteq N(s, 2c(s,t))$ because of node count, and thus $a_{k+d+1}(s) \leq 2c(s,t)$. With Lemma 10.7,

$$\sum_{j=1}^{k-1} c(v_j, v_{j+1}) \leq \frac{\gamma}{\gamma-1} \cdot a_{k+1}(s) \leq \frac{2\gamma^{1-d}}{\gamma-1} \cdot c(s,t) \ .$$

The second phase is the hop from v_k to w_{k-1}. Using the triangle inequality, it follows that

$$\begin{aligned} c(v_k, w_{k-1}) &\leq c(v_k, s) + c(s,t) + c(t, w_{k-1}) \\ &\leq a_{k+1}(s) + c(s,t) + a_k(t) \leq (2\gamma^{-d} + 1 + \gamma^{-d-1})c(s,t) \ . \end{aligned}$$

The third and last phase of the route is the traversal from $w_{k-1}, w_{k-2}, \ldots$ back to $w_1 = t$. Because of $a_{k+d+1}(t) \leq c(s,t)$ and from Lemma 10.7,

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

The theorem is proven by choosing $d = O(\log(1/\epsilon))$ such that

$$\epsilon \geq \frac{1}{\gamma^d}\left(\frac{2\gamma}{\gamma-1} + 2 + \frac{1}{\gamma} + \frac{1}{\gamma-1}\right) \ .$$

□

10.4 Dynamic node arrivals and departures

In this section we sketch how nodes may dynamically arrive and depart from the system. We assume that once two nodes v and w connect (by v sending a message that arrives at w) they may exchange messages and discover the real distance $c(v,w)$ between them.

Node arrival

When a new node arrives to the system it needs to do several things: (1) acquire an id for each of its routers, (2) establish network links for each of its routers, (3) acquire necessary object references.

Acquiring an identifier for each router. Each node chooses for each initial router, $R_1, \ldots, R_{M+1}$, an identifier of M radix B digits independently and uniformly at random. Note that due to a significant change in the number of nodes, the parameter M may change. In such a case, routers may need to add a new digit to each of their identifiers.

Finding the nearest neighbor. As part of the process of establishing router links, a node first needs to identify the closest neighbor it has in the network. Using the LAND architecture, the nearest neighbor is always found, and a load-balanced distributed nearest neighbor search will take an expected logarithmic number of steps. Algorithm find-closest for a node v is as follows. Let u be any node in the network known to v, e.g., an initial contact point. We denote $w_{M+1} = u$, and its level-$(M+1)$

initial router by $w_{M+1}.r = u.R_{M+1}$. For $\ell = M+1$ down to 2, take from among all incoming links into $w_\ell.r$ the router $w_{\ell-1}.r$ closest to v. By construction, $w_{\ell-1}.r$ is a level-$(\ell-1)$ router with $w_{\ell-1}.r.id[\ell-2] = w_{M+1}.r.id[\ell-2]$. At the end, set the closest node known to v, denoted $v.closest$, to w_1.

The find-closest algorithm is depicted in pseudo-code in Figure 2

A node v that wants to find its closest neighbor invokes $v.closest = v.R_{M+1}$.find-closest($v, M+1$).

find-closest(v, ℓ) at router $u.r$:
If $\ell = 1$ then send u to v and return;
Let S_i denote the set of all incoming links into $u.r$;
Let w among S_i be the closest to v;
$w.r$.find-closest($v, \ell-1$);

Figure 2: The find-closest algorithm.

Establishing network links. Once the id and level of a router is set, and the closest node to the node hosting it is known, the router is left with the task of establishing links as defined in Section 4. For a router $v.r$ with level ℓ, the main difficulty is to find all the level $\ell+1$ routers with prefix $v.r.id[\ell-1]$ in the ball $A_{\ell+d+5}(v)$. Router $v.r$ also needs to inform all routers $u.r$ of level $\ell-1$ with prefix $v.r.id[\ell-2]$ such that $v \in A_{\ell+d+4}(u)$. This can be done, again, by finding all routers $u.r$ in $A_{\ell+d+5}(v)$ with prefix $v.r.id[\ell-2]$. The locate algorithm for a router $v.r$ of level ℓ is as follows.

Let s be the closest node to v. For every combination of digits $b_1, b_2 \in [0, B-1]$, route from s to a level-$(\ell+2)$ router $u.r$ such that $u.r.id[\ell+1] = v.r.id[\ell-1] \circ b1 \circ b2$. This routing is done in an identical manner to the routing phase of lookup in Figure 1, i.e., using the $L(i)$ links. Let Y denote the set of routers $u.r$ reached by this procedure. Let $S(\ell+3)$ be the set of level-$(\ell+3)$ routers that appear as publish links of routers in Y, i.e., $S(\ell+3) = \bigcup_{y \in Y} y.P$. Obtain $S(\ell+2)$ by taking all incoming publish links into $S(\ell+3)$ from routers 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 $S(\ell+1)$, router $v.r$ selects neighbor links whose distance from v does not exceed $a_\ell(v)$, and keeps publish links whose distance from v does not exceed $a_{\ell+d+5}(v)$. Then v informs nodes in $S(\ell-1)$ about its arrival.

The locate algorithm is depicted in pseudo-code in Figure 3

Node departure

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

Analysis of dynamic algorithms

Lemma 10.10 *For a node v, Algorithm find-closest finds the closest neighbor of v.*

locate at router $v.r$ of level ℓ:
for every combination of digits $b_1, b_2 \in B$
  $v.closest.R_1$.search($v.r.id[\ell-1] \circ b1 \circ b2, \ell, v$);
wait for replies, accumulate in S;
set $v.r.L(i) = \text{argmin}_{u.r \in S}\{c(u,v) \mid u.r.level = \ell+1 \wedge u.r.id[\ell] = v.id[\ell-1] \circ i \wedge c(v,u) \leq a_\ell(v)\}$;
// emulate $v.r.L(i)$ if empty
set $v.r.P = \{u.r \in S \mid u.r.level = \ell+1 \wedge u.r.id[\ell-1] = v.r.id[\ell-1] \wedge c(v,u) \leq a_{\ell+d+5}(v)\}$;
inform all level-$(\ell-1)$ routers in S about v's arrival;

search($prefix, \ell, v$) at router $u.r$:
if $u.r.level = \ell + 2$ then
  for each $w \in r.P$ with $w.r'.level = \ell + 3$: $w.r'$.inlinks($v, 4$);
else $u.r.L(prefix_{u.level})$.search($prefix, \ell, v$);

inlinks(v, j) at router u.r:
// recurse for j levels searching for in-links
let I denote the incoming publish links of $u.r$;
if ($j > 0$) then
  for each $w.r \in I$: $w.r$.inlinks($v, j-1$);
else send v the set I;

Figure 3: The locate algorithm.

Proof. Denote by u the node at which find-closest is initiated by v. Denote $w_{M+1}.r = u.R_{M+1}$, and denote the sequence of routers traversed by the algorithm by $w_{M+1}.r, w_M.r, \ldots, w_1.r$. Let s denote the closest node to v in the network. We show by induction backward from $M+1$ to 1 that $w_i \in A_{i+2}(v)$ and $w_i.r$ is the closest router to v which satisfies $w_i.r.id[i-1] = w_{M+1}.r.id[i-1]$. The base obviously holds. We now prove the induction step.

Consider the routing path from s to $w_{M+1}.r.id$ using the regular digit-fixing routing method. Denote the router reached in the ith routing step by $y_i.r$, i.e., $y_i.r.level = i$ and $y_i.r.id[i-1] = w_{M+1}.r.id[i-1]$. By Lemma 10.6, $y_i \in A_{i+1}(s)$. Since $s \in A_1(v)$, by Lemma 10.1(ii), $A_{i+1}(s) \subseteq A_{i+2}(v)$, and so $y_i \in A_{i+2}(v)$. Hence, if there is any level-i router $c_i.r$ closer to v than $y_i.r$, then $c_i \in A_{i+2}(v)$ as well. By the induction hypothesis, $w_{i+1} \in A_{i+3}(v)$, and hence, by Lemma 10.1(i), $A_{i+2}(v) \subseteq A_{i+3}(v) \subseteq A_{i+4}(w_{i+1})$. From Lemma 10.1(i), this means $A_{i+4}(w_{i+1}) \subseteq A_{i+5}(c_i)$, and therefore, $c_i.r$ has a publish pointer to $w_{i+1}.r$. Therefore, Algorithm find-closest would find $c_i.r$ in this step, and in fact, $w_i.r = c_i.r$. This completes the proof. □

Lemma 10.11 *For a router $v.r$ with level ℓ, Algorithm locate finds all the appropriate level-$(\ell+1)$ routers in $A_{\ell+d+5}(v)$ and all appropriate routers $w.r$ with level $\ell-1$ such that $v \in A_{\ell+d+4}(w)$.*

Proof. Let s be the closest node. From Lemma 10.1 it follows that $A_{\ell+d+5}(v) \subseteq A_{\ell+d+6}(s)$. By Lemma 10.6, each router $y.r$ that is routed to in algorithm locate is in $A_{\ell+3}(s) \subseteq A_{\ell+d+6}(s)$. In addition, each such router has level $\ell+2$ so its publish links cover all appropriate level $\ell+3$ routers in $A_{\ell+2+d+5}(y)$. Applying Lemma 10.1 again we get $A_{\ell+d+5}(v) \subseteq A_{\ell+d+6}(s) \subseteq A_{\ell+d+7}(y)$. Hence, by following incoming links backwards down to level $\ell-1$, all appropriate level-$(\ell-1)$ routers within $A_{\ell+d+5}(v)$ are guaranteed to be found. For every w such that $v \in A_{\ell+d+4}(w)$, we have that $A_{\ell+d+4}(w) \subseteq A_{\ell+d+5}(v)$. By a similar reasoning to the above, we find all such w's. $\square$

Lemma 10.12 *For a node arrival:*

(i) The expected number of nodes that change their state is logarithmic.

(ii) The expected number of messages sent is $O(M^2)$

Proof. For each $\ell \in [1, M]$, routing to each of the level $\ell + 2$ routers takes $\ell + 2$ messages. Once such a router is reached, a message is sent to each of its links. The expected number of links is constant (see Theorem 5.1), and all choices are independent. Thus recursively finding all appropriate links for all M routers will cause sending an expected $O(M)$ number of messages. The number of nodes that change their state for each $\ell \in [1, M]$ is the number of appropriate routers in $A_{\ell+d+5}(v)$, the expected number of these routers is constant. $\square$

Maintaining the correct ball radius $a_i(v)$

In order to maintain the ball radius, the peers cannot count the exact number of nodes within a radius of r from them since this would not be scalable. Instead, a simple sampling trick works for any $i \geq 1$:

If $\alpha B^i = O(\log n)$, then node v can keep track of the exact number of nodes. Otherwise, v chooses a set of random prefixes of length $i - \log(c \log n)$ for some constant c and maintains an exact number of all nodes within a distance r whose i-router matches one of the prefixes. If there are αB^i nodes within a distance of r from v, then v should find, on expectation,

$$\frac{c \log n}{B^i} \cdot \alpha B^i = \alpha c \log n$$

many such nodes. This also holds with high probability since the router labels are chosen independently at random. Hence, if for a given radius $a_i(v)$ the number of nodes counter by v is less than $\alpha c \log n$, v needs to increase $a_i(v)$, and if it is more than $\alpha c \log n$, v needs to decrease $a_i(v)$.

References

[1] I. Abraham, D. Malkhi, and O. Dobzinski. LAND: Stretch $(1 + \epsilon)$ locality-aware networks for dhts. In *Proc. of the 23rd IEEE Symp. on Principles of Distributed Computing (PODC)*, pages 550–559, 2004.

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

[3] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of the 29th ACM Symp. on Theory of Computing (STOC)*, pages 654–663, 1997.

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

[5] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32:241–280, 1999. A preliminary version of this paper appeared in *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, June 1997.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01*, 2001.

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