Object Location in Realistic Networks

[Extended Abstract]

Kirsten Hildrum*
University of California, Berkeley
Berkeley, CA 94720, USA
hildrum@cs.berkeley.edu

Robert Krauthgamer†
IBM Almaden Research Center
San Jose, CA 95120, USA
robi@almaden.ibm.com

John Kubiatowicz*
University of California, Berkeley
Berkeley, CA 94720, USA
kubitron@cs.berkeley.edu

ABSTRACT

We devise an object location scheme that achieves a *guaranteed* low stretch in a wider and more realistic class of networks than previous schemes. The distinctive feature of our scheme is that it is inherently adaptive to the underlying topology. In particular, the system achieves $1 + \epsilon$ stretch (for arbitrarily fixed $\epsilon > 0$), with a neighbor list size that depends on the local density around the node (but not on the global growth rate bound). As a byproduct, our scheme has several advantages over existing ones, such as robustness to errors in network measurements, and simpler design choices of system builders, which may lead to improved and more robust deployments.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Distributed Data Structures, Graphs and Networks; F.2.2 [**Nonnumerical Algorithms and Problems**]: Routing and Layout; C.2.4 [**Distributed Systems**]: Distributed Applications

General Terms

Algorithms, Theory

Keywords

Networking, overlay, locality, peer-to-peer, distributed object location, DOLR, distributed hash table, DHT

*Research supported by NSF career award #ANI-9985250, NFS ITR award #CCR-0085899, and California MICRO award #00-049.

†Part of this work was done while this author was with the International Computer Science Institute and with the Computer Science Division of U.C. Berkeley, supported in part by NSF grants CCR-9820951 and CCR-0121555 and DARPA cooperative agreement F30602-00-2-0601.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SPAA'04, June 27–30, 2004, Barcelona, Spain.
Copyright 2004 ACM 1-58113-840-7/04/0006 ...$5.00.

1. INTRODUCTION

Peer-to-peer networks unite computers around the globe via the Internet to accomplish a common goal. Although each participant maintains links (via the Internet) to only a few others, the goal is to form an efficient and versatile overlay network. These highly scalable peer-to-peer networks have many potential applications, ranging from file sharing [5, 18, 27] to database query and indexing [14].

A central problem in these highly-scalable distributed systems is *object location* (aka *lookup*), or finding an object (e.g., a file or service) in the network.[1] One easy solution to this problem is to build a centralized directory that lists all the items and their locations. However, this would put a huge load, in terms of both computation and bandwidth, on that centralized directory. Moreover, the directory becomes a single point of failure.

A common solution to this problem is to maintain the directory via a *Distributed Hash Table* (*DHT*), which is a dictionary data structure implemented in a distributed way. A lookup for an object accesses this distributed dictionary to get a pointer to the object (or the object itself). Typical DHT algorithms, such as [1, 21, 23, 26, 28, 31], build an overlay network among the participants, where every participant maintains links to only a few other participants, called its *neighbors*, and the DHT operations are implemented by routing messages through the resulting overlay topology. A full-fledged DHT should support insertion and deletion of items, allow nodes to join and leave the overlay network, balance the load placed on different participants, and be resilient to failures.

1.1 Performance and locality

The efficiency of a DHT is often measured in terms of the number of neighbors per participant and the number of hops per message routed. These measures are well-suited to networks in which all possible overlay links have the same cost, in which case the total cost of an object location is proportional to the number of hops in a route. However, in many real-life networks, internode communication cost varies significantly. For instance, hops in a local area network are much cheaper (e.g., faster, more reliable and over-provisioned) than hops across the Pacific. Since popular objects are likely to be replicated, many lookups could be

[1]We consider only "structured" peer-to-peer networks, which are guaranteed to return a copy of the object if one exists. Commonly used unstructured networks include KaZaA, Gnutella, and Skype.

satisfied by a nearby copy of the object. This suggests that it is important to keep lookups local when possible; for example, for a request made in Berkeley for a file stored at IBM Almaden, routing through Stanford is reasonable, but routing through Australia is not.

Keeping lookups local is extremely important. First, for the searcher, it means that many lookups are much cheaper to perform (e.g., complete faster), because the searched objects have nearby replicas. Second, local lookups reduce the total bandwidth usage, decrease congestion at backbone routers, and limit the effect of faults at remote unrelated regions. Furthermore, in schemes that are locality-aware, a node's neighbors tend to be closer to the node (compared with random nodes), and thus maintenance traffic tends to put less stress on the network. These effects on the network performance may be particularly important since most peer-to-peer applications in use today, such as KaZaA, are bandwidth hogs.

Typical DHTs are limited in their ability to keep lookups local. They place objects in a location dependent on the object's ID. Since these IDs are hashes, these locations are typically random. This means that an object created and accessed from Stanford could be located in Australia, or vice-versa. As a result, most lookups are inherently non-local, and no cleverness of algorithms or data structures can change that. If most requests must cross the entire network (ie, going from Australia to Stanford), attempting to optimize the case when requests go from Berkeley to Stanford is useless. In this case, optimizations focus on efficient routing—that is, making the length of the overlay path between two nodes is as close as possible to the path in the underlying network. See for example [6, 9, 26], and an algorithm with provable bounds in [15].

In contrast to the DHT scenario, this paper considers a model in which the objects are placed arbitrarily; that is, our algorithm has no control over object placement. We also allow multiple copies of a given object to be placed in the network. Our challenge is really to effectively take advantage of good object placement.The measure of performance is *stretch*, the ratio between the total cost of a lookup route and the cost of direct link to the cheapest (for us, cheapest means "nearest") copy of an object.Thus, we present what is sometimes called a DOLR [7], or a *Decentralized Object Location and Routing* structure. In such a structure, performance is measured relative to the least possible cost of accessing a copy of the object.[2]

A DHT that stores pointers to objects can be used to locate objects, but with very high stretch. Consider a user in a dorm downloading `popularsong.mp3`. Even though a copy of the song may be available within the same dorm, the DHT will force a request to go to an arbitrary and probably distance location to find the nearby copy, incurring a high stretch. (Note that determining which copy is nearby is not a trivial problem.) A good object location system would find the in-dorm copy without sending a message that leaves the dorm.

[2]Essentially, our algorithm is working against an adversary that places objects in the network to maximize stretch obliviously, i.e., without being able to see random coin tosses (ID assignments) of our algorithm. We do not use this terminology because locating *nearby* objects is the most difficult, so our worst "adversary" is one who places objects near their accesses—something that is actually desirable!

Stretch guarantees. In a seminal paper [21], Plaxton, Rajaraman and Richa (PRR) pinpoint this issue of locality and measure it in terms of stretch. Their object location scheme achieves rigorous stretch guarantees, and had influenced both theoretical work and deployed systems.

In the PRR model [21], the cost of communication between two nodes u, v in the network is given by a distance function $d(u, v)$ that forms a metric space (X, d), i.e., satisfies triangle inequality. One motivation of this model is network latency, which may serve as a rough estimate for the distance between nodes. The PRR scheme achieved constant expected stretch, assuming that the node metric satisfies the following *growth rate* bound: Let $B(v, r)$ denote the set of nodes within distance r from v; then there are known constants $C \geq C' > 1$ such that $C' \leq \frac{|B(v,2r)|}{|B(v,r)|} \leq C$ for every node v and (effectively) every $r > 0$. The work of [21] inspired deployed systems such as Pastry [26] and Tapestry [32], and was followed by subsequent schemes that addressed various technical shortcomings. In particular, provably-good algorithms for node arrival and departure are devised in [11–13], a simplified scheme is given in [20], a fault-tolerant extension is provided in [10], and a much simplified scheme with $1 + \epsilon$ deterministic stretch and without the need for the lower bound C' is designed in [1]. Outside the context of peer-to-peer networks, several structures have been proposed for object location in general metric spaces. See for example, [4] and [22]. The distance oracles of [29] and the compact routing schemes of [2, 3] also address this problem, though they don't phrase their results in this context. These schemes achieve polylogarithmic space and polylogarithmic stretch. However, these structures are impractical in the peer-to-peer world because they require the underlying network to be fixed from the start. In addition, some cannot be constructed in a distributed fashion, and many are not even load-balanced, requiring a central server. Until our work, there was no low-stretch object location system for dynamic networks that did not require the corresponding metric space to be uniformly growth-restricted.

Limitations of previous systems. All the existing low-stretch schemes intrinsically rely on knowing a *global* growth rate upper bound C. These systems are based on assigning node IDs that are represented in some base $B \geq C$, and routing messages toward a given destination ID by fixing one digit at a time. It follows that the number of neighbors each node has to maintain is (at least) proportional to C. Since a larger number of neighbors immediately increases the amount of maintenance traffic, it is crucial for deployment that C is assigned a modest value.

In practice, however, it is likely that in many regions the global value C would be much larger than the actual local value,

$$\rho_{v,r} := \frac{|B(v, 2r)|}{|B(v, r)|},$$

which we call the *local growth rate* around node v at scale $r > 0$. This would be consistent, for example, with the transit-stub model [30] of the Internet, which features *stub domains* (modeling local networks) and *transit domains* (modeling backbone networks that interconnect stub domains); see Figure 1 for an illustration. Traffic between two nodes within a domain is routed inside the domain, and is typically fast compared to interdomain links. Each stub domain probably has an internal structure, but we cannot expect

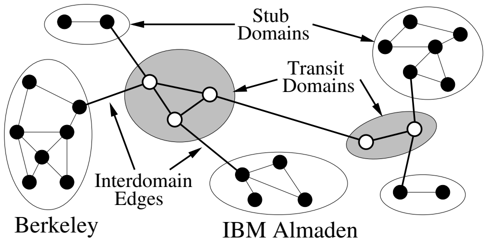

Figure 1: An illustration of a (non-hierarchical) transit-stub graph.

that the Berkeley stub domain would be similar to the IBM Almaden one. It is therefore only natural to require that the neighbors selection policy in the Berkeley stub domain is kept independent of the internal topology of unrelated domains such as IBM Almaden.

A partial remedy is to set C ignoring a few regions of extremely high local growth rate, giving up on low stretch in those regions. But even then significant gaps between the local growth rate and C might occur for the following reasons.

- The local growth rate is likely to vary significantly between different subnetworks—a local network at a small liberal arts college is probably very different from the one at a large research university.
- The local growth rate probably varies significantly between scales, because networks of different scale (e.g., local area networks vs. a nationwide network) have different technologies.
- Systems based on a global value C are designed with the value of C being determined before the network is first launched. In a dynamic system, this means that C must be increased by a large safety margin to guarantee future performance.

Large gaps between the local growth rate $\rho_{v,r}$ and the global value C might have a tremendous impact on the overall network. Many low-provisioned subnetworks would probably suffer from excessive maintenance traffic, resulting in a high network stress and poor network utilization.

It should be noted that low-stretch guarantees might require some dependency on $\rho_{v,r}$. Consider for example a subset S of k nodes, every two of which are at the same distance 1, and suppose all other nodes are at least ϵ away from every node of S. Then, a lookup whose source and destination are in S can achieve $1+\epsilon$ stretch only if it is performed in one hop, which means that every node $v \in S$ must have at least $k-1$ neighbors, where $k \simeq \rho_{v,\gamma}$ for any $\frac{1}{2} < \gamma < 1$. Notice that this scenario is likely to happen in large local area networks. See also [16, 17] for a related notion and similar lower bounds.

1.2 Our Results

We present a scheme that achieves a *guaranteed* low stretch object location without requiring a global growth rate bound C, and is thus effective in a wider and more realistic class of networks than previous schemes. The distinctive feature of our scheme is that it is inherently adaptive to the underlying topology, as its operations in any locale depend on the locale's properties. In particular, our system achieves the same $1+\epsilon$ stretch as LAND [1], with a neighbor list size does not depend on C, but only on the local growth rate $\rho_{v,r}$, on ϵ, on the *normalized diameter* of the network $D_N = D/d_{\min}$, where D is the largest distance between two points in the network and $d_{\min}$ are the largest and smallest pairwise distances in the network, respectively), and on the following bounds on the rate of change of $\rho_{v,r}$:

$$\delta_s := \sup\left\{\frac{\rho_{v,r}}{\rho_{v,2r}} : v \in X, r > 0\right\},$$

which upper bounds the change in growth rate over different but nearby scales, and

$$\delta_d := \sup\left\{\frac{\rho_{v,r}}{\rho_{u,r}} : u, v \in X, r > 0, \text{ and } d(u,v) \le r\right\},$$

which upper bounds the change in growth rate over different, but nearby (relative to the scale in question), nodes.

THEOREM 1.1. *There exists a randomized scheme that achieves object location with $1+\epsilon$ stretch, such that the expected number of neighbors of each node v is at most $f(\max_{v,r}\rho_{v,r}, \delta_s, \delta_d, \epsilon) \cdot \log^2 D_N$, for some function f.*

Our system design takes a direct approach toward achieving low stretch. As a byproduct, we get the following advantages (over existing schemes), which we believe will culminate in improved and more robust deployments of low stretch object location schemes.

- The system is adaptive to the local growth rate, which may be significantly smaller than the global upper bound C, thereby using less resources in many regions of the network.
- It operates without requiring a predetermined value C and it is therefore more practical in *any* underlying topology, since a single value of C might be hard to find. In fact, our approach identifies the algorithmic problem of estimating the distribution of the latencies between a node v and all other nodes in the network, and such an algorithm is sketched in Section 5.2.
- Our construction is robust to small errors in network measurements. Since accurate network measurements are difficult to make and require lots of bandwidth, this is important. While a node needs to count the number of nodes within a certain distance from it, having this number off by a factor of two will have only a slight effect.[3]
- Our scheme has low *node congestion*; that is, no node is unfairly burdened with requests from the rest of the network. In our scheme, if every node generates at most t requests, the expected node-congestion is $O(t \log D)$. For comparison, note that, by averaging, the expected node congestion in any scheme must be at least t times the expected number of hops.[3]
- The scheme lends itself to large local area networks, where the local growth rate is large, but many cheap hops are acceptable in practice; see Section A.

[3]This property may possibly be true also in existing schemes, but proving it would require an indirect and more complicated analysis.

In addition, because our system uses a form of prefix routing, the fault tolerance results of [10] apply naturally. In practice, one may also use a "hybrid" system that combines our scheme with a ring [9, 24–26].

Organization. The low stretch scheme is based on a construction of an overlay network with an $O(\text{diameter})$ routing data structure. To gain intuition, we outline the construction, focusing on a simplified case, in Section 2.2, and defer some details of the general case to Section 3. We then use a technique of LAND [1] to extend the routing algorithm to a low stretch object location in Section 4. Next, we use a technique of [11–13] to adapt the system to arriving and departing nodes in Section 5. Finally, we touch upon a heuristic improvement for large local area networks in Section A.

2. ALGORITHM OUTLINE

Our object location scheme is based on an $O(\text{diameter})$ routing overlay that does not rely on a global bound C, as summarized in the following theorem.

THEOREM 2.1. *There exists a randomized routing scheme, such that the expected number of neighbors of each node v is at most $f(\max_{v,r} \rho_{v,r}, \delta_s, \delta_d) \cdot \log^2 D_N$, for some function f, and any message travels a total distance of $O(D)$ en route to its destination ID.*

All previous low stretch object location schemes [1, 21, 32] (and their extensions) use prefix routing, i.e., every hop "corrects" one additional digit by matching it to the destination ID, and hence the ith hop in the routing is guaranteed to have ID that matches the destination ID in the first i digits. In all these systems, the digit size is determined in advance so as to accommodate for the growth rate of the network. In contrast, our scheme does not have a fixed digit size. Instead, we match a varying number of bits in each step, which is essentially like having a variable digit size. This leads to using, in every locality of the network, the optimal digit size for that locale. More precisely, the number of bits matched at a given step depends on the local growth rate $\rho_{v,r}$. As a result, it can vary both over the nodes and over the scales. For example, one message may match two bits at the first hop (scale) and five at the second, while another may match four at the first hop and three at the second; in fact, the total number of bits fixed in the first two hops need not be the same, unless the two messages have the same destination ID and they meet after two hops. See Figure 2 for illustration.

The rest of this section outlines the proof of Theorem 2.1, by focusing on a simplified case. Details of the general construction are given in Section 3. We then use this routing overlay in Section 4 to get $1+\epsilon$ stretch by additional "publish" pointers on nodes that are in the vicinity of the search path, similar to LAND [1]. The additional pointers increase storage by a factor that depends on the growth rate and on ϵ. Throughout, we shall assume $\rho_{v,r}$ is known exactly, although it can be easily seen that it suffices to know $\rho_{v,r}$ within a constant factor.

2.1 A Simple Scheme

Each node v in the overlay network is assigned a randomly chosen identifier, called an ID. These IDs are sufficiently long bit strings so that no two are the same. Suppose a node u wishes to send a message, given only the destination's ID. If, say, $d_{\min}$ and D are powers of 2, then the message will route through at most $\log D_N$ hops, where the ith hop is, by definition, no longer than $2^i d_{\min}$, as depicted in Figure 2(left). Thus, the total length of a routing path is at most $\sum_{i=0}^{\log D_N} 2^i d_{\min} \leq 2D$, i.e., twice the network diameter.

The routing is a variant of prefix routing, where instead of fixing one bit at a time, the number of fixed bits depends on the local growth rate at the corresponding scale. The last hop will fix all the bits, thereby guaranteeing arrival to destination. This is in contrast to previous systems, such as PRR [21], Tapestry [32], Pastry [26], and LAND [1], which choose a predetermined number of bits to fix at each step; their aim is to get the ith hop travel at most a distance of $2^i d_{\min}$, but this property is only indirectly guaranteed via the growth rate bound. Figure 2(center) depicts our bit fixing along two routing hops. Notice that in the dense region (represented by darker shading), more bits are fixed at the earlier hop, but after two hops the same number of bits have been fixed.

Routing Entities. We divide the routing state of a node into *routing entities*, and describe the routing as a transaction between entities. A routing entity corresponds to a level of the neighbor table in [1, 21, 26, 32]. A *scale* is a number $2^i \in [d_{\min}, 2D]$, where i is an integer. Each node hosts a *seed entity* for every scale 2^i, and may host additional *emulated* entities, as will be described below.

Each entity has an ID, denoted $E.\mathsf{id}$. The ID of a seed entity is the ID of the hosting node. To determine which messages E may accept, each entity E also has a *prefix requirement*, denoted $E.\mathsf{req}$, that we shall define later and a scale, denoted $E.\mathsf{scale}$. The neighbors of an entity E are all seed entities E' such that[4]

(a). $E'.\mathsf{scale} = 2 \cdot E.\mathsf{scale}$;

(b). E' is within distance $E.\mathsf{scale}$ of E; and

(c). $E'.\mathsf{id}$ agrees with $E.\mathsf{id}$ on the $E.\mathsf{req}$ first bits.

Routing Algorithm. A given message passes through entities of increasing scales. Denoting the ith entity in the route as E_i, the first entity E_0 is the minimum scale (i.e. scale $2^i \in [d_{\min}, 2d_{\min})$) seed entity of the node that generated the message. At the ith step, the entity E_i forwards the message to the entity E_{i+1} that is chosen as the entity E' whose ID prefix matches the destination ID in the most bits possible among all nodes meeting requirements (a), (b), and

(c$'$) $E'.\mathsf{id}$ agrees with the destination ID on the $E'.\mathsf{req}$ first bits.

The routing always terminates when the scale is the message reaches an entity E_i whose scale $E_i.\mathsf{scale}$ is greater than twice the diameter of the network D. Indeed, by definition, the previous entity E_{i-1} stores *all nodes in the network* (more precisely, seed entities of scale at least D) that agree with the destination ID on $E_{i-1}.\mathsf{req}$ bits, so the destination node must be in this list. Notice that the shorter $E.\mathsf{req}$, the more neighbors E will have to store. But if $E.\mathsf{req}$ is long, it can only accept routing requests for objects with a long ID match, which means that other nodes cannot use E as a neighbor quite as easily. Lemmas 2.1 and 2.2 show this tradeoff.

---

[4]Note that this asymmetric definition of being a neighbor provides a directed link from E to E'. The actual implementation of this may be bi-directional.

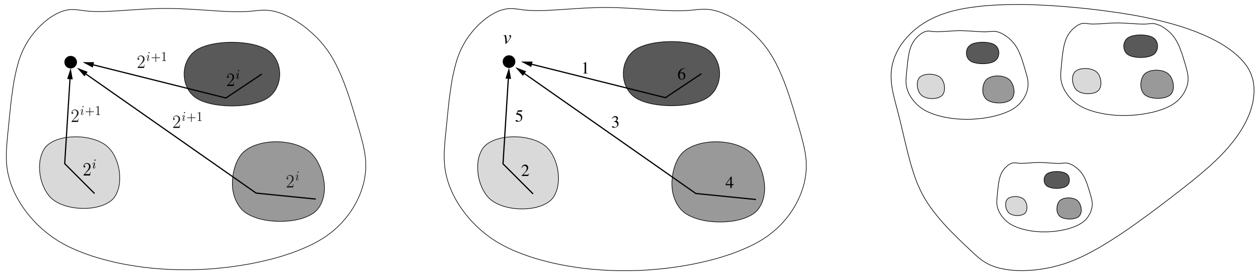

Figure 2: The leftmost picture depicts the ith and $i+1$st hops in the route of three messages, showing that when $d_{\min} = 1$, each ith hop travels distance at most 2^i. The center picture shows the number of (additional) bits that are fixed along a single hop in these routes. Darker shading represents a higher density of nodes, which leads to fixing more bits, but by the time the messages reach v, all routes fix a total of seven bits. The rightmost picture shows that the left two pictures are only a piece in a larger scheme.

Unfortunately, there is no guarantee for E_i that an entity matching (a),(b) and (c$'$) exists. For example, suppose an entity E_i with prefix requirement 11 is trying to route a message to ID 11011. If the nodes with prefix 110 are all more than 2^i away, E cannot use any of them in its neighbor table without violating the invariant that the ith hop does not travel too far away. In this case, where no suitable seed entity is found, we use a technique of LAND [1] and let the node create an *emulated* entity with the required ID prefix (e.g., 110) and scale (e.g., 2^{i+1}). This emulated entity stores all the information that a seed entity would.

Note that emulating entities is not quite the same as giving the node v another ID, as that would be the equivalent of creating an entire new node, while an emulated entity only corresponds to a single level of the routing table. Nonetheless, it is important the total number of emulated entities per node is small, and proving this is the main technical difficulty. A standard argument regarding branching processes shows that if the expected number of emulated entities that are *directly* generated by any single entity is upper bounded by a constant $\lambda < 1$ then the total number of emulated entities per seed entity is expected to be a constant. The intuition is that if that expectation was $\lambda \geq 1$, then each seed entity generates, in expectation, λ emulated entities, each of which generates λ more emulated entities, and so on, resulting in an exponential blow up in the total number of neighbors that poor node has to maintain.

The Prefix Requirement. The challenge in defining the prefix requirement is to create a self-organizing overlay network that adapts to the local growth rate, but still ensures that there is a routing path from any locale to any possible destination. A fast growing $E.\mathsf{req}$ (as a function of the scale) means that there will be few matching nodes inside the ball of radius $E.\mathsf{scale}$, and so many emulations will be required. On the other hand, a slow growing $E.\mathsf{req}$ means that there will be many matching nodes, and so E's neighbor table will be large. The crux is to show that one can define a prefix requirement that simultaneously satisfies these two competing needs.

2.2 Sketch of Analysis

Before considering the general case (in Section 3), we analyze the case where the local growth rate is "smooth" in the sense that $\rho_{v,r} \approx \rho_{w,r}$ whenever v and w are close, i.e., within distance r of each other. Notice that this does not preclude the growth rate from varying considerably between distant points. In this simplified case, we shall scale all the distances in the network so that the smallest distance is 1, and the largest (i.e., the diameter of the network) is D. Hence, a *scale* is a number 2^i for an integer $0 \leq i \leq \lceil \log D \rceil + 1$. we further assume here that all quantities are integral to avoid rounding notation, and denote by log a base 2 logarithm.

For each scale s entity E located at a node v, we set the prefix requirement $E.\mathsf{req}$ to be

$$\mathsf{pref}(v, s) := \log |B(v, s/2)| - a,$$

where a represents a universal comfort factor.[5] Setting a picks the tradeoff point between table size and the number of emulated entities. Note that $E.\mathsf{req}$ depends on the node's locale, unlike the analogous prefix requirement in [1,21,26,32]. We next show in Lemmas 2.1 and 2.2 that our prefix requirement indeed satisfies the two competing needs mentioned above.

LEMMA 2.1. *The expected table size of an entity of scale s on node v is $2^a \rho_{v,s/2}$.*

PROOF. There are $|B(v, s)|$ nodes in the ball of radius s around v. Each one matches v's required prefix with probability $2^{-\mathsf{pref}(v,s)} = \frac{2^a}{|B(v,s/2)|}$, so the entity's expected table size is $|B(v, s)| \cdot \frac{2^a}{|B(v,s/2)|} = 2^a \rho_{v,s/2}$. $\square$

LEMMA 2.2. *If a is chosen large enough, the expected number of directly emulated entities is less than* 1.

PROOF. Let $\mathsf{pref}(v, s)$ denote the prefix length required by v at scale s.

An entity E at scale s located on node v requires an emulated entity for every extension of $E.\mathsf{req}$ by $\mathsf{pref}(v, 2s) - \mathsf{pref}(v, s)$ bits where E knows of no suitable seed entity E' (in terms of ID, scale, and distance from E). Think of the $\mathsf{pref}(v, 2s) - \mathsf{pref}(v, s)$ bits as describing a digit to match. Intuitively, we match as many digits as possible so long as we can find a suitable neighbor in the ball around E of radius $E.\mathsf{scale}$.

Think of the $2^{\mathsf{pref}(v,2s)-\mathsf{pref}(v,s)} = \rho_{v,s}$ possible extensions as bins, and of the nodes within distance s from E as balls.

[5]In this simplified exposition a would actually have a weak dependence on the local growth rate. This is eliminated in our general scheme, using a more complicated prefix requirement function.

The assumption that $\rho_{v,r} \approx \rho_{w,r}$ means that $\mathsf{pref}(w, 2s) = \mathsf{pref}(v, 2s)$, and so if w as the right prefix and the right scale, its prefix requirement does not prevent it from being a neighbor of v.

We then wish to bound the expected number of empty bins. By Lemma 2.1, the expected number of balls is $2^a \rho_{v,s}$, and, clearly, each of them lands in a random bin. Notice that this is just a coupon collecting problem; so if there are N bins and more than $N \ln N$ balls, the expected number of empty bins falls below a constant less than one. Here, we only know the expected number of balls, but the argument is similar. It follows that the expected number of directly emulated entities is strictly less than some constant $\alpha < 1$ if $2^a \rho_{v,s} \geq \rho_{v,s} \ln \rho_{v,s}$, so a needs be larger than $\log_2 \ln \rho_{v,r}$. $\square$

The above shows how to build a system when the local growth rate $\rho_{v,r}$ is "smooth". The rest of the paper deals with the case where the local growth rate allowed to vary (in a bounded way) across scales and between nearby points in the network. The proof of this more general case follows the same basic outline as the above.

3. MORE GENERAL NETWORKS

In this section we prove Theorem 2.1 by constructing an O(diameter) routing overlay for the general case. The algorithm and terminology in this section is exactly the same as that in Section 2. In particular, recall that the routing state of each node was divided into entities.[6] The difference is that here we need to handle a more complicated network model, which may have significant changes in local growth rates, and hence we employ a less naive definition of the prefix requirement of an entity E and a more involved proof. In particular, we set $E.\mathsf{req}$ of an entity at node v at scale r to be

$$\mathsf{pref}(v, r) := \max\{0, \log |B(v, r/2)| - c \log(\rho_{v,r}) - a\}.$$

(We round down if it is not an integer.) Here and throughout, log denotes a logarithm to base 2; a and c are parameters; we will show in Section 3.2 that setting $c \geq 2$ and $a \geq \max\{3 + \log_2 \delta_d, 2 + \log_2(c + 2) + \log_2 \log_2(\delta_s \delta_d)\}$ are sufficient for our purposes, i.e., that the expected size of the neighbor table of a node is small. But first we bound the total distance traveled by a message before reaching its destination.

3.1 Guaranteed Delivery and O(D) Routing

Let D denote the diameter of the network (i.e., the maximum distance between two points), and let $d_{\min}$ denote the minimum distance between two points. Let $D_N = D/d_{\min}$ be the *normalized diameter*. We shall show that messages are delivered in $O(\log D_N)$ steps, and the total distance traveled is only $O(D)$. Before proving the efficiency, we prove that every message is delivered.

LEMMA 3.1. *The routing network described always delivers any given message.*

PROOF. Let the path of message m through the network be entities $E_1, E_2, \ldots, E_i$. At every step, the ID of E_i matches the destination ID in $E_i.\mathsf{req}$ bits. This is proved by induction on i. The base case is E_1, where $E_1.\mathsf{req} = 0$. The third routing property (c′) ensures that a message is sent only to an entity E' that matches the destination ID in at least $E'.\mathsf{req}$ bits, so the inductive step holds.

Finally, when $E.\mathsf{scale}$ is larger than the diameter D, the entity E stores a seed entity for *every* node ID (throughout the network) that match in $E.\mathsf{req}$ bits, so the next hop goes directly to the destination. If the message arrives at the destination at some intermediate scale, it remains there. $\square$

COROLLARY 3.1. *Every message is routed through at most $\log D + O(1)$ hops, and the total distance it travels is $O(D)$.*

PROOF. The first hop travels distance at most $2d_{\min}$, and this upper bound is doubled at every hop. The final hop is of length at most $2D$. The total distance is $\sum_{i=\lceil \log d_{\min} \rceil}^{\lceil \log D \rceil} 2^i = O(D)$. The number of hops is equal to the number of terms of the sum; this is $\lceil \log D \rceil - \lceil \log d_{\min} \rceil + 1 = \log D_N + O(1)$ hops. $\square$

3.2 Space Complexity

We turn to bounding the expected number of neighbors of a node. The high level argument, where dependency on $\rho_{v,r}, \delta_s, \delta_d$ is suppressed by assuming they are constant, goes as follows. By definition, each node hosts $O(\log D_N)$ seed entities. Now for some constant $\lambda < 1$, each seed entity is expected to *directly* generate at most λ emulated entities; where each of those is, in turn, expected to directly generate at most λ additional emulated entities, and so forth. By a standard branching process bound, the total number of emulated entities that a single seed entity is expected generate is at most $\frac{1}{1-\lambda} \leq O(1)$. Finally, the expected number of neighbors of each entity (seed or emulated) is $O(1)$, and therefore each node is expected to have a total of $O(\log D_N)$ neighbors.

Technically, this argument is flawed, because it does not account for certain correlations. First, the upper bound on the branching process requires certain events to be independent, which is not true in our case — the events of emulating a scale r and a scale $2r$ entities at the same node u depend on the IDs of nodes in $B(u, r)$ and $B(u, 2r)$, respectively, and these two balls are clearly non-disjoint. Furthermore, in some extreme cases (e.g., if the node u is in Hawaii) these two balls might be equal, and then any emulated entity at scale r generates (deterministically) an emulated entity at scale $2r$. In particular, this shows that λ might be as large as 1; as a result, the branching process bound might be as large as $O(\log D_N)$, increasing the final bound of the above high level argument to $O(\log^2 D_N)$, similar to Theorem 2.1. A second correlation is between the total number of emulated entities generated a single seed entity at a node u, and the number of neighbors of each of these emulated entities — these random variables both depend on the IDs of nodes around u.

In this extended abstract, we prove that the bounds mentioned in the above high level description hold for seed entities. Analyzing emulated entities is more complicated because their existence is correlated with the IDs of nearby nodes, and at the end of this section we briefly explain the technique used to analyze the corresponding conditional probabilities and expectation.

[6] A routing entity corresponds to a level of the neighbor table in PRR [21], Tapestry [32], or Pastry [26]. Since we later "duplicate" some levels of the neighbor table via emulation, we need a convenient way to talk about a level.

Lemma 3.2. *The expected number of neighbors for a scale s seed entity located at node v is $O(2^a \rho^c_{v,s} \rho_{v,s/2})$.*

Proof. A scale s seed entity E located at a node v has to store all the scale $2s$ seed entities matching $E.\mathsf{req}$ within distance s of v. There are $|B(v,s)| = \rho_{v,s/2}|B(v,s/2)|$ nodes within distance s of v, and the probability that any single one matches the prefix requirement is $\frac{2^a \rho^c_{v,s}}{|B(v,s/2)|}$, so the expected number of matching nodes in $B(v,s)$ is $2^a \rho^c_{v,s} \rho_{v,s/2}$. $\square$

Lemma 3.3. *If for every entity, the expected number of emulated nodes it directly generates is at most a constant $\lambda < 1$ independently of the other entities, then the expectation of the total number of entities generated from one seed entity (directly or indirectly) is at most $\frac{1}{1-\lambda} \leq O(1)$.*

Proof. For the purposes of this proof, we define an ith degree emulated entity recursively as follows. An ith degree emulated entity as an entity that created by a $i-1$st degree entity. Let a seed entity be a 0th degree emulated entity (that is, one that is not emulated at all). Let X_i be a random variable denoting the expected number of degree-i entities stemming from one seed entity. Then $\mathbb{E}[X_{i+1}|X_i] \leq X_i \cdot \lambda$, and taking expectation on both sides gives $\mathbb{E}[X_{i+1}] \leq \lambda \cdot \mathbb{E}[X_i]$. Since $X_0 = 1$, we have that $\mathbb{E}[X_i] \leq \lambda^i$, and thus $\sum_i \mathbb{E}[X_i] \leq 1/(1-\lambda)$. $\square$

Lemma 3.4. *The expected number of entities directly emulated by a seed entity is at most $1/e$, if $c \geq 2$ and $a \geq \max(3 + \log \delta_d, 2 + \log(c+2) + \log\log \delta_s \delta_d)$.*

Proof. Let E be a seed entity of scale r at node v. Consider a message m that may be routed through E. By definition, this message's destination ID must agree with with $E.\mathsf{id}$ on the first $\mathsf{pref}(v,r) = \max\{0, \log_2|B(v,r/2)| - c\log_2(\rho_{v,r}) - a\}$ bits. Now fix a node $w \in B(v,r)$ and let's see whether E can forward this message m to w's scale $2r$ seed entity $w.\mathsf{E}[2r]$. This is possible if the destination ID matches $w.\mathsf{id}$ on the first $\mathsf{pref}(w,2r)$ bits. Since w's ID is random, this event happens with probability $2^{-\mathsf{pref}(w,2r)}$. Hence,

$$\Pr[E \text{ has to emulate to route } m] \leq \prod_{w \in B(v,r)} (1 - 2^{-\mathsf{pref}(w,2r)}).$$

Letting $\hat{w}$ be the node $w \in B(v,r)$ whose prefix requirement $\mathsf{pref}(w,2r)$ is maximal, we get the upper bound

$$\Pr[E \text{ has to emulate to route } m] \leq (1 - 2^{-\mathsf{pref}(\hat{w},2r)})^{|B(v,r)|}.$$

Let Y_E be a random variable representing the number of entities directly emulated by E. In order to upper bound $\mathbb{E}[Y_E]$, we need to consider all possible messages m that may be routed through E. The point is that the many possible destination IDs can be grouped into relatively few distinct events. Notice that our arguments above for the message m actually depend only on the first $\mathsf{pref}(\hat{w},2r)$ bits of the destination ID, while the first $\mathsf{pref}(v,r)$ bits of the destination ID must be the same as those of $v.\mathsf{id}$. Thus, we can group the possible destination IDs by their contents in positions $\mathsf{pref}(v,r)+1, \ldots, \mathsf{pref}(\hat{w},2r)$. By the analysis above, the probability that (the destination IDs of) a single group requires an emulated entity is at most $(1 - 2^{-\mathsf{pref}(\hat{w},2r)})^{|B(v,r)|}$.

There are at most $2^{\mathsf{pref}(\hat{w},2r) - \mathsf{pref}(v,r)}$ groups, and hence

$$\begin{aligned} \mathbb{E}[Y_E] &\leq 2^{\mathsf{pref}(\hat{w},2r) - \mathsf{pref}(v,r)} \cdot (1 - 2^{-\mathsf{pref}(\hat{w},2r)})^{|B(v,r)|} \\ &\leq e^{\mathsf{pref}(\hat{w},2r) - \mathsf{pref}(v,r) - |B(v,r)|/2^{\mathsf{pref}(\hat{w},2r)}}. \end{aligned}$$

It remains to upper bound the righthand side by $1/e$. This requires some technical calculations, whose intuition is the following. Suppose that any ball of radius r in the network contains about r^d nodes, for all r. If we were to define $\mathsf{pref}(v,r) = \log|B(v,r)|$, then $\mathsf{pref}(\hat{w},2r) - \mathsf{pref}(v,r) \simeq \log[(2r)^d/r^d] = d$, and $|B(v,r)|/2^{\mathsf{pref}(\hat{w},2r)} \simeq |B(v,r)|/|B(\hat{w},2r)| \simeq 1/2^d$, which yields a poor upper bound $\mathbb{E}[Y_E] \leq 2^d \exp\{-2^{-d}\}$. But we can easily overcome this by setting $\mathsf{pref}(v,r) = \log|B(v,r)| - 3d$, and then Y_E is at most $2^d \cdot \exp\{-2^{-d+3d}\} = o(1)$.

Now, to the actual calculations. Notice that $B(\hat{w},r) \subseteq B(v,2r)$, and that $|B(v,2r)| = \rho_{v,r}\rho_{v,r/2}|B(v,r/2)|$. Hence,

$$\begin{aligned} \mathsf{pref}(\hat{w},2r) - \mathsf{pref}(v,r) &\leq \log\frac{|B(\hat{w},r)|}{|B(v,r/2)|} + c\log\frac{\rho_{v,r}}{\rho_{\hat{w},2r}} + 1 \\ &\leq \log(\rho_{v,r}\rho_{v,r/2}) + c\log\frac{\rho_{v,r}}{\rho_{\hat{w},2r}} + 1 \end{aligned}$$

Second, by the definition of the prefix requirement we have

$$\begin{aligned} |B(v,r)|/2^{\mathsf{pref}(\hat{w},2r)} &= |B(v,r)| \cdot \frac{2^{a-1}(\rho_{\hat{w},2r})^c}{|B(\hat{w},r)|} \\ &\geq \frac{2^{a-1}(\rho_{\hat{w},2r})^c}{\rho_{v,r}} \end{aligned}$$

Noticing that $\rho_{v,r/2} \leq \delta_s \rho_{v,r}$ and $\rho_{v,r} \leq \delta_d \rho_{\hat{w},r} \leq \delta_d \delta_s \rho_{\hat{w},2r}$, we obtain

$$\begin{aligned} \mathbb{E}[Y_E] &\leq \exp\{\log(\delta_s(\delta_d\delta_s\rho_{\hat{w},2r})^2) + c\log(\delta_d\delta_s) \\ &\quad + 1 - 2^{a-1}(\rho_{\hat{w},2r})^{c-1}/(\delta_d\delta_s)\} \\ &\leq \exp\{2\log\rho_{\hat{w},2r} + (c+3)\log(\delta_d\delta_s) \\ &\quad + 1 - 2^{a-1}(\rho_{\hat{w},2r})^{c-1}/(\delta_d\delta_s)\} \end{aligned}$$

To conclude that the righthand side is at most $1/e$, it suffices to show that

$$\begin{aligned} 2\log\rho_{\hat{w},2r} - 2^{a-3}(\rho_{\hat{w},2r})^{c-1}/(\delta_d\delta_s) &\leq 0 \\ (c+3)\log(\delta_d\delta_s) - 2^{a-3}(\rho_{\hat{w},2r})^{c-1}/(\delta_d\delta_s) &\leq 0 \\ 1 - 2^{a-2}(\rho_{\hat{w},2r})^{c-1}/(\delta_d\delta_s) &\leq -1 \end{aligned}$$

Rearranging these three inequalities we get

$$\begin{aligned} \delta_d\delta_s\log\rho_{\hat{w},2r} &\leq 2^{a-4}(\rho_{\hat{w},2r})^{c-1} \\ (c+3)(\delta_d\delta_s)\log(\delta_d\delta_s) &\leq 2^{a-3}(\rho_{\hat{w},2r})^{c-1} \\ 2\delta_d\delta_s &\leq 2^{a-2}(\rho_{\hat{w},2r})^{c-1} \end{aligned}$$

Since $\rho_{\hat{w},2r} \geq 1$, the third inequality holds whenever $a \geq 3 + \log_2(\delta_d\delta_s)$. Since $\log\rho_{\hat{w},2r} \leq \rho_{\hat{w},2r}$, the first inequality holds whenever $c \geq 2$ and $a \geq 4 + \log_2(\delta_d\delta_s)$. Since $\rho_{\hat{w},2r} \geq 1$, the second inequality holds whenever $a \geq 3 + \log(c+3) + \log(\delta_d\delta_s) + \log\log(\delta_d\delta_s)$.

To conclude, it suffices that $c \geq 2$ and $a \geq 3 + \log(c+3) + \log(\delta_d\delta_s) + \log\log(\delta_d\delta_s)$. $\square$

Analyzing emulated entities. Emulated entities located at the same node are correlated, and therefore the branching process bound cannot be applied to the total number of emulated entities that a seed entity generates. We overcome this by considering a branching process over a subset of the scales, which we call *marked scales*. These marked scales are defined only for the sake of analysis – the algorithm does not change. Starting with the scale of the seed entity, which is always marked, iteratively increase the current scale r by a factor of 2. If $|B(v,r)|$ is at least twice as large as the ball at the last marked scale, we mark scale

r. Let us denote the subset of marked scales by $\{m_i\}_i$; thus, $\frac{|B(v,m_i)|}{|B(v,m_{i-1})|} \geq 2$, and $\frac{|B(v,m_i/2)|}{|B(v,m_{i-1})|} < 2$. Now a slight generalization of Lemma 3.4 can bound the number of emulated entities at a scale $r \in [m_i, m_{i+1})$ conditioned on the number of emulated entities in the marked scale m_{i-1} (or the seed entity, if $i = 0$), and this is enough to prove an analogue to the branching process. The main points in changing Lemma 3.4 are bounding the term corresponding to $\mathsf{pref}(\hat{w}, 2r) - \mathsf{pref}(v, r)$, and arguing that the IDs of at least half the nodes in the ball corresponding to $B(v,r)$ are independent of the data we conditioned on.

Another issue is the expected number of neighbors of an emulated entity. Fixing two seed entities, E at node u and E' at node u', we upper bound the expectation of the total number of links that all the emulated entities generated (directly or indirectly) by E have to the entity E'. The main point is that the expected number of the former entities (which are all of scale $E'.\mathsf{scale}/2$) can be bounded, even if we condition on a fixed ID for u' (because for Lemma 3.4 it suffices that all nodes but one have random IDs), and now the randomness in the ID of E' makes the number of entities that link to E' have small expectation.

3.3 Load balance

We now consider the load balance of our scheme, measured as the number of message routed through any single node, when every network node is the source of at most one message to a random ID. The bound scales linearly by linearity of expectation, if every node is the source of at most t messages.

LEMMA 3.5. *Consider a routing where every node is the source of at most one message with a fixed destination ID. The total number of messages expected to route through any single node w (over the random choices of node IDs) is at most $O(2^a \rho_{w,r}^{c+1} \rho_{w,r/2} \log D)$.*

PROOF. Fix a node $w \in X$, and consider a message sent from source node u toward a given destination ID. The number of hops in our routing scheme is $O(\log D)$ by Lemma 3.1. Let v_r be the node visited in the scale r hop along the route. We know that $d(v_{r/2}, v_r) \leq r$, and thus $d(u, v_r) \leq r + r/2 + \cdots < 2r$. Thus, w can only be the ith hop in the route if $u \in B(w, 2r)$. Furthermore, for w to be the scale r hop in the route it is necessary that it matches the destination ID on a prefix of length $\mathsf{pref}(w, r)$. We shall consider only hops visiting a seed entity at w, because if the route can contain an emulated entity then it must also contain at least one seed entity at the same node. Now for a seed entity, since w's ID is random, the prefix match happens with probability $1/2^{\mathsf{pref}(w,r)}$. Therefore, the expected number of source nodes (and thus messages) that use w as their ith hop is at most

$$|B(w, 2r)| \cdot \frac{1}{2^{\mathsf{pref}(w,r)}} \leq \frac{|B(w, 2r)| \cdot 2^a \rho_{w,r}^c}{|B(w, r/2)|} = 2^a \rho_{w,r}^{c+1} \rho_{w,r/2}.$$

□

4. OBJECT LOCATION

This section shows how to use the overlay network construction described in the previous section to do low-stretch object location. The routing algorithm and its geometrically increasing hop lengths will play a key role in achieving low stretch. We use the notation of Section 3.

Objects are placed in the network by their *publisher* node. An object location data structure (DOLR) such as the one presented here determines where to place pointers to the objects so that a *searcher* node is able to locate object copies efficiently. We measure the efficiency of an object-location request in terms of *stretch*, the ratio between the total distance traveled searching for the object and the distance to the closest copy of the object. It is possible to achieve a stretch of one by placing pointers to the object at all potential searchers; this is too much. In this section, we show how to place a small number of pointers and get $O(1)$ stretch. By placing more pointers in a manner similar to LAND [1], $1 + \epsilon$ stretch is possible.

The routing algorithm forms the basis for the object location algorithm. The name of the object is hashed into the same namespace as the node IDs, and the publisher of the object routes toward the object's ID. The node with the ID most closely matching the object's ID is the *root*. The publisher then places pointers along the routing path from the object's location to object's ID (i.e., the root). To search for the object, the searcher routes toward the object's ID. Suppose that a searcher and a publisher are within distance r of each other. If they happen to have the same scale-r routing hop when routing toward the object's ID, then the searcher finds a pointer to the object at that hop and is able to shortcut the rest of the routing. In the worst case, however, the first hop they share might be of a scale much larger than r (such as the network's diameter).

To solve this, the publisher places pointers to the object at any possible scale-r routing entity that a searcher within distance r might visit when looking for the object. This yields $O(1)$ stretch, as we describe below. We call this new set of neighbors "publish neighbors," and the old set of neighbors are "routing neighbors".

More precisely, for an entity E at scale $E.\mathsf{scale}$ with prefix requirement $E.\mathsf{req}$, maintain links to all scale $E.\mathsf{scale}$ seed *and emulated* entities E' within distance $5 \cdot E.\mathsf{scale}$ of E matching in $\min\{E'.\mathsf{req}, E.\mathsf{req}\}$ bits. Objects are published by routing toward the object's root. At every hop, pointers to the object are placed on the publish neighbors of the current entity. Search routes toward the object's root, checking for a pointer to the object at each entity en route.[7] We now prove that this results in $O(1)$ stretch.

LEMMA 4.1. *If node y searches for an item published by node x, the total length of the search path before a pointer to the object is found is at most $18d(x, y)$.*

PROOF. Let x_r be the scale r hop on the path toward the root from x, and let y_r be the scale r hop on the path toward the root from y. Choose r^* such that $r^*/2 \leq d(x, y) \leq r^*$. The key part of the proof is to note that $d(x_{r^*}, y_{r^*}) \leq 5r^*$. Figure 4 shows why this is so. In particular, notice that $d(x, x_{r^*}) \leq 2r^*$, and $d(y, y_{r^*}) \leq 2r^*$, then $d(x_{r^*}, y_{r^*}) \leq d(x, y) + d(x, x_{r^*}) + d(y, y_{r^*}) \leq 5r^*$. When x_{r^*} received the publish request for the item, it sent an object pointer to y_{r^*} (since y_{r^*} is within distance $5r^*$ and matches in the right number of bits). Thus, when y searches for an object, it has

[7] An alternative would be to keep the publishing only along the path to the root and have the searcher look in many places at every scale. (See Awerbuch and Peleg [4] and their discussion of read and write sets.)

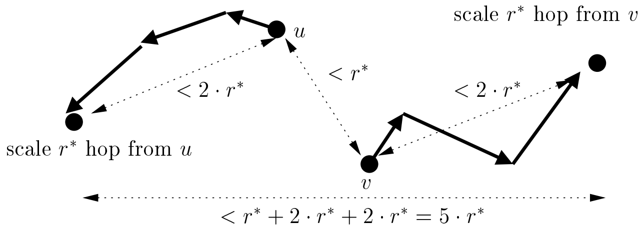

Figure 3: The routes of messages that have the same destination ID. If the two source nodes are within distance r^*, their corresponding scale r^* hops are within $5r^*$ of each other.

to pay at most the distance to y_{r^*} and then the distance from y_{r^*} to the object. The first distance is bounded by $2r^*$, and the second distance is at most $d(y_{r^*}, x_{r^*})+d(x_{r^*}, x)$. Hence, the overall the length is at most $2r^* + 5r^* + 2r^* \leq 9r^*$, and since $d(x, y) \geq r^*/2$, the stretch is at most 18. □

It remains to bound the expected number of publish neighbors a node has to maintain. Similar to the discussion in Section 3, we analyze below only the case of seed entities; the analysis can then be extended to emulated entities using similar techniques to those sketched in Section 3.

LEMMA 4.2. *The expected number of publish neighbors for a scale s seed entity at node v is $O(2^a \delta_d^{5+c} \rho_{v,8s} \rho_{v,s}^{c-1} \delta_s^{10+3c})$.*

PROOF. A scale s entity E located at a node v has to store all the nodes matching $E.\mathsf{req}$ within distance $5s$ of v. There are $|B(v,s)| = \rho_{v,s/2}|B(v,s/2)|$ nodes within distance s of v, so there are $\rho_{v,s/2}\rho_{v,s}\rho_{v,2s}\rho_{v,4s}|B(v,c)|$ within $8s$ of v, which bounds the number within $5s$ of v. The probability that any single one matches the prefix requirement of E is $2^{-\mathsf{pref}(v,s)} = \frac{2^a \rho_{v,s}^c}{|B(v,s/2)|}$, so the expected number of matching nodes in $B(v, 8s)$ is $2^a \rho_{v,4s}\rho_{v,2s}\rho_{v,s}^c\rho_{v,s/2}$.

However, this only counts nodes that match v's prefix requirement. But recall that v needs to connect to all entities u such that match in $\min\{\mathsf{pref}(u, s), \mathsf{pref}(v, s)\}$, so the number of entities actually stored by v can be a factor of $2^{\mathsf{pref}(v,s)-\mathsf{pref}(u,x)}$ greater.

Consider some entity E located at a node u such that $d(u, v) \leq 8s$. Since $|B(u, 16s)| \geq |B(v, 8s)|$, and using the definition of δ_s and δ_d, we get

$$\begin{aligned}
&\mathsf{pref}(v,s) - \mathsf{pref}(u,s) \leq \\
&\leq \log \frac{|B(u,16s)|}{|B(v,8s)|} + \log \frac{|B(v,s/2)|}{|B(u,s/2)|} - c \log \frac{\rho_{v,s}}{\rho_{u,s}} \\
&\leq \log \frac{\rho_{u,8s}\rho_{u,4s}\rho_{u,2s}\rho_{u,s}\rho_{u,s/2}}{\rho_{v,4s}\rho_{v,2s}\rho_{v,s}\rho_{v,s/2}} + c \log \rho_{u,s} \\
&\leq \log \frac{(\delta_d \rho_{v,8s})^5 \delta_s^{10}}{\rho_{v,4s}\rho_{v,2s}\rho_{v,s}\rho_{v,s/2}} + c \log \delta_s^3 \delta_d \rho_{v,8s} \\
&\leq \log \frac{\delta_d^{5+c} \rho_{v,8s}^{5+c} \delta_s^{10+3c}}{\rho_{v,4s}\rho_{v,2s}\rho_{v,s}\rho_{v,s/2}}.
\end{aligned}$$

Thus, v may need to match a factor of $\frac{\delta_d^{5+c}\rho_{v,8s}^{5+c}\delta_s^{10+3c}}{\rho_{v,4s}\rho_{v,2s}\rho_{v,s}\rho_{v,s/2}}$ more nodes than the $2^a \rho_{v,4s}\rho_{v,2s}\rho_{v,s}^c\rho_{v,s/2}$ that match its prefix requirement, which is at most $O(2^a \delta_d^{5+c} \rho_{v,8s} \rho_{v,s}^{c-1} \delta_s^{10+3c})$.

This counts the number of seed entities stored. But at a given scale, in expectation, each seed entity hosts only a constant number of emulated entities, so the bound still holds even when emulated entities are considered. □

Counting the number of publish neighbors from an emulated entity is more complicated, but can be done using the ideas mentioned in Section 3.

Finally, notice that we can reduce the number of publish neighbors if we make the search algorithm more extensive. If nodes both push pointers to their publish neighbors *and* search for objects on their publish neighbors, it is sufficient for each node to keep only those entities in $B(v, 5s)$ that match in $\mathsf{pref}(v, s)$ bits, resulting in a total of $O(2^a \rho_{v,4s}\rho_{v,2s}\rho_{v,s}^c\rho_{v,s/2})$ publish neighbors for a seed entity. Using the notation of Lemma 4.1, suppose that a search request from y found a publish pointer at y_{r^*} search in the old scheme. Then in the new scheme, either it finds a pointer at y_{r^*} as before, or y_{r^*} searches for the item on x_{r^*}.

5. DYNAMIC SYSTEM

In order to be practical, this system must be able to adapt to arriving and departing nodes. To build a table for a node v and scale r, we need

1. The number of nodes within distance r of v.
2. The local growth rate around v at scale r (i.e., $\rho_{v,r}$).
3. All the entities matching $\mathsf{pref}(v, r)$ within r of v.

Items (1) and (2) are needed to determine $\mathsf{pref}(v, r)$. Notice that they are equivalent; given (1) for all r, one can compute (2), and vice-versa. While the problem of computing (1) and (2) is new, other systems [1, 13, 26, 32] have considered (3).

A first approach to estimating (1) and (2) is to find a physically nearby node, and copy the values from there. Intuitively, this should work well, though it may be hard to prove performance results, especially as the network evolves over time. Section 5.2 gives a more rigorous technique.

Likewise, finding approximate neighbor lists can also be done easily, if, as is typical in these systems, the data structures need only be close to the correct ones. In this case, the algorithms of [26, 32] can be used. These algorithms start with a physically nearby node as before, copying its scale r_0 (where r_0 is the smallest scale) neighbor table. Using this list, the node routes toward its own ID, and takes that node's scale $2r_0$ neighbors as its $2r_0$ scale neighbors, and so on. This would be complicated by the different prefix requirements, but some immediate heuristics to deal with this suggest themselves. These lists can be optimized by asking all the scale r neighbors for their neighbors measuring the distances to these neighbors of neighbors, and updating the neighbor table appropriately.

The rest of this section sketches provable techniques for finding (1)-(3). We first show how to find all nodes with a given prefix within distance d of a particular starting point, solving problem (3), and then explain how this primitive can be used for finding the growth rates and the number of nodes in the ball of radius r.

5.1 Neighbors

The basic idea is to use the backward-routing techniques first described in [13] and later refined in [11, 12]. (Because of the emulated nodes, the algorithm also resembles [17].) We sketch the algorithm here. Consider particular entity E. Its *children* are the entities that have E as a neighbor, and its grandchildren are the entities that have children of E as a neighbor, and so on. Also, if an entity E generates and entity E', we say that E is a child of E'. Define descendants analogously as all entities on the routing path to entity E. The set of descendants form a tree. Note that all scale $d_{\min}$ entities are descendants of every scale D entity.

LEMMA 5.1. *All the descendants of a scale r entity are within distance r of the entity.*

PROOF. The proof is by induction. For the base case, consider an entity at scale $d_{\min}$. It has no children, and so all its descendants are within $d_{\min}$. Now consider an entity E at scale r. Its children are all within $r/2$ by construction. If it has a child further than $r/2$ away, that child could not keep E as a neighbor. Further, we know (by the inductive hypothesis) that all the descendants of those child-entities are within $r/2$, and so they are within $r/2 + r/2 = r$ of E. □

To build the table for a node v, we need an algorithm to find all entities within distance d of v. The above suggests the following algorithm. Start with any scale D entity E (recall that D is the largest possible scale). Then, the set S_r will contain all the scale r descendants of E that are ancestors of entities within d of v. Start with $S_D = \{E\}$. Then, given S_{2r}, we find S_r as follows:

1. For each entity e in S_{2r}, put the children of e (all of which have scale r) in S_r.
2. For each entity e in S_r, measure $d(e, v)$ (the distance to v). If this distance is more than $r + d$, then remove e from S_r.

LEMMA 5.2. *If u is within distance d of v, the above algorithm never removes an ancestor of u.*

PROOF. Denote u's scale r ancestor by E_u^r. By Lemma 5.1, we know that $d(u, E_u^r) \leq r$. By assumption, $d(u, v) \leq d$, so by the triangle inequality, $d(v, E_u^r) \leq r + d$, so for each r, E_u^r is kept in the set S'. □

The above algorithm finds all nodes within d starting with *any* scale D entity. However, recall that v needs to find all the nodes with a given prefix within d. Following nearly the same algorithm, we can get this for free by carefully selecting the starting entity so that its ID matches the desired ID. Suppose, more precisely, that we want all the nodes within d of v that match a particular ID α in at least k bits. Then S_D is initialized to contain a scale D entity matching α in at least k bits. Second, we add an additional step to the algorithm.

3. For each e in S_r, if e does not match α in at least k bits, remove e from S_r.

Notice that a search for the neighbors at r finds the scale $r' > r$ neighbors with only a little additional work, so by essentially running this algorithm once, a node v can find neighbors for all the entities it hosts. Using techniques from [11–13], one can bound the work in terms of the growth rate and the number of neighbors returned.

5.2 Finding growth rate

The previous sections sketch an implementation of a primitive that can be used to find all entities with a specified prefix within a given distance efficiently. Because the prefixes are assigned randomly, this function can be used to get a random sample of the nodes at the rate $1/2^i$ for any i by picking a prefix of length i.

This primitive can be used to estimate the number of nodes in $B(v, r)$ and $\rho_{v,r}$. Pick a prefix of length i. Then to get an estimate of the number of nodes in $B(v, r)$, count the number of nodes (ignore any non-seed entities) matching in i bit prefix and multiply by 2^i. The error in this technique can be made small if i is chosen so that there are enough nodes matching the prefix in $B(v, r)$. Estimates of the size of $B(v, r)$ and $B(v, 2r)$ can be used to calculate the $\rho_{v,r}$.

Other work [8, 19] has shown how to sample randomly from a peer-to-peer network, but these techniques give a random sample from anywhere in the network, rather than a sample from within a certain radius, so their results do not seem applicable here.

6. CONCLUSION

We have shown how to build a low-stretch object location system that is sensitive to the *local* growth rate. This is useful not only in situations were the growth rate changes but also in situations where the growth rate is fixed, but unknown or hard to determine.

Though we require more space when the growth rate is high, there are some indications that our system is close to optimal. Defining optimality in this case and proving lower bounds is an interesting open problem. In addition, the similarity between the object location structure and the nearest-neighbor search algorithm of [17] suggests that the two problems (low-stretch object location and neighbor search) may be closely related.

Acknowledgments

Thanks to Dahlia Malkhi, Satish Rao, and Sean Rhea for their insightful comments.

7. REFERENCES

[1] I. Abraham, D. Malkhi, and O. Dobzinski. LAND: Stretch $(1 + \epsilon)$ locality-aware networks for DHTs. In *15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 543–552, Jan. 2004.

[2] M. Arias, L. J. Cowen, and K. A. Laing. Compact roundtrip routing with topology-independent node names. In *Proceedings of 22nd Annual Symposium on Principles of Distributed Computing*, pages 43–52, 2003.

[3] M. Arias, L. J. Cowen, K. A. Laing, R. Rajaraman, and O. Taka. Compact routing with name independence. In *Proceedings of 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 184–192, 2003.

[4] B. Awerbuch and D. Peleg. Concurrent online tracking of mobile users. In *Proc. of SIGCOMM*, pages 221–233, 1991.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of SOSP*, pages 202–215, 2001.

[6] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a dht for low latency and high throughput. In *Proceedings of the First Symposium on Networked Systems Design and Implementation*, pages 85–98, Mar. 2004.

[7] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. . Stoica. Towards a common API for structured P2P overlays. In *Proceedings of IPTPS*, pages 33–44, 2003.

[8] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, 2004.

[9] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. of SIGCOMM*, pages 381–394, 2003.

[10] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *17th International Symposium on Distributed Computing*, pages 321–336, 2003.

[11] K. Hildrum, J. Kubiatowicz, S. Ma, and S. Rao. A note on finding the nearest neighbor in growth-restricted metrics. In *Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 553–554, 2004.

[12] K. Hildrum, J. Kubiatowicz, and S. Rao. Another way to find the nearest neighbor in growth-restricted metrics. Technical Report UCB/CSD-03-1267, UC Berkeley, 2003.

[13] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *14th ACM SPAA*, pages 41–52, 2002.

[14] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, Sept. 2003.

[15] D. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proc. of the 34th ACM Symp. on Theory of Comp.*, pages 741–750, 2002.

[16] R. Krauthgamer and J. R. Lee. The black-box complexity of nearest neighbor search. In *31st International Colloquium on Automata, Languages and Programming*. July 2004. To appear.

[17] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 791–801, Jan. 2004.

[18] J. Kubiatowicz et al. OceanStore: An architecture for global-scale persistent storage. In *Proc. of ACM ASPLOS*, pages 190–201. ACM, 2000.

[19] C. Law and K.-Y. Siu. Distributed construction of random expander networks. In *Proceedings of IEEE INFOCOM*, 2003.

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

[21] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32:241–280, 1999.

[22] R. Rajaraman, A. W. Richa, B. Vocking, and G. Vuppuluri. A data tracking scheme for general networks. In *Symposium on Parallel Algorithms and Architectures*, pages 247–254, 2001.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. of SIGCOMM*, pages 161–172, 2001.

[24] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. To appear in USENIX '04 Annual Technical Conference.

[25] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. Technical Report UCB//CSD-03-1299, UC Berkeley, 2003.

[26] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware*, pages 329–350, 2001.

[27] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of SOSP*, pages 188–201, 2001.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM*, pages 149–160, 2001.

[29] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of the 33th Annual ACM Symp. on Theory of Comp.*, pages 183–192, July 2001.

[30] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM*, pages 594–602, 1996.

[31] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*, 2003. Special Issue on Service Overlay Networks, to appear.

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

APPENDIX

A. TRADING STRETCH AND STORAGE IN CERTAIN SUBNETWORKS

In a large local area network (LAN), where the local growth rate is large but hops are cheap, it may be acceptable to perform relatively many hops and give up on the low stretch guarantee for nearby objects. This can be easily achieved in our scheme by a simple modification – routing at sufficiently low scales proceeds by fixing only a single bit at every hop. This requires that every node maintains a seed entity for every single bit that may be fixed, but the number of hops in a route would be logarithmic in the number of nodes in the LAN. For example, if a LAN contains 1024 nodes, then instead of fixing 10 bits at the first hop, each of the first 10 hops will fix only on additional bit, so routing to any node in the LAN would take only 10 hops, and every node only maintains 10 neighbors, instead of 1024.

Further, in a LAN, because the distances within in the LAN are small compared to to the distance to the outside of the LAN, the higher scales are unaffected. Proving this in a more general metric becomes difficult, however.