Efficient Lookup on Unstructured Topologies

Ruggero Morselli, Bobby Bhattacharjee, Aravind Srinivasan
Department of Computer Science
University of Maryland
College Park, MD 20742
{ruggero,bobby,srin}@cs.umd.edu

Michael A. Marsh
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742
mmarsh@umiacs.umd.edu

ABSTRACT

We present LMS, a protocol for efficient lookup on unstructured networks. Our protocol uses a virtual namespace *without imposing specific topologies*. It is more efficient than existing lookup protocols for unstructured networks, and thus is an attractive alternative for applications in which the topology cannot be structured as a Distributed Hash Table (DHT).

We present analytic bounds for the worst-case performance of our protocol. Through detailed simulations (with up to 100,000 nodes), we show that the actual performance on realistic topologies is significantly better. We also show in both simulations and a complete implementation (which includes over five hundred nodes) that our protocol is inherently robust against multiple node failures and can adapt its replication strategy to optimize searches according to a specific heuristic. Moreover, the simulation demonstrates the resilience of LMS to high node turnover rates, and that it can easily adapt to orders of magnitude changes in network size. The overhead incurred by LMS is small, and its performance approaches that of DHTs on networks of similar size.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; G.3 [**Mathematics of Computing**]: Probability and Statistics; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Design, Performance, Theory

Keywords

Peer-to-peer networks, lookup protocols, distributed algorithms, random walks

PODC'05, July 17–20, 2005, Las Vegas, Nevada, USA.

1. INTRODUCTION

Large peer-to-peer networks require efficient object lookup mechanisms. Flooding searches, while adequate for small networks, quickly become untenable as networks grow larger. Consequently, research has progressed along two avenues: more efficient lookup protocols that operate on general (unstructured) topologies, and fundamentally new lookup strategies that achieve bounded worst-case performance by adding constraints to the network topology. The latter, while very efficient, might not be usable when the peer-to-peer application places its own constraints on the topology. This is especially the case when the links in the topology have particular significance, such as expressing statements of trust or belief between peers. From an algorithmic perspective, this sort of network topology appears to be unstructured.

We approach this issue by considering a model where a graph G is given (the topology), such that each peer in the system corresponds to a vertex u of G and can only send messages to peers that correspond to neighbors of u in G. Developing distributed protocols in this model is not only an interesting theoretical problem, but has several practical applications, such as that outlined in Section 6. In this paper, we present the Local Minima Search (LMS) protocol for unstructured topologies. LMS extends the notion of random walks, which to date have proved the most promising approach to improving search performance on unstructured networks [10, 7, 3]. In addition, LMS borrows the ideas of *namespace virtualization* and *consistent hashing* employed by constrained-topology protocols such as distributed hash tables (DHTs) [19, 22, 21, 11]. Namespace virtualization maps both peers and objects to identifiers in a single large space.

In LMS, the owner of each object places replicas of the object on several nodes. Like in a DHT, LMS places replicas onto nodes which have IDs "close" to the object. Unlike in a DHT, however, in an unstructured topology there is no mechanism to route to the node which is the closest to an item. Instead, we introduce the notion of a *local minimum*: a node u is a local minimum for an object if and only if the ID of u is the closest to the item's ID in u's *neighborhood* (those nodes within h hops of u in the network, where h is a parameter of the protocol, typically 1 or 2). In general, for any object there are many local minima in a graph, and replicas are placed onto a subset of these. During a search, random walks are used to locate minima for a given object, and a search succeeds when a minimum holding a replica is located. While DHTs typically provide a worst-case bound of $O(\log n)$ steps for lookups in a network of size n, LMS pro-

vides a worst-case bound of $O(T(G) + \log n)$, where $T(G)$ is the *mixing time* of G (the time by which a random walk on the topology G approaches its stationary distribution). $T(G)$ is $O(\log n)$ or polylogarithmic in n for a wide range of randomly-grown topologies. This "$O(T(G)+\log n)$" is typically in the $6-15$ range in our experiments with networks of size up to $100,000$. Let d_h be the minimum size of the h-hop neighborhood of any node in G. LMS achieves its performance by storing $O(\sqrt{n/d_h})$ replicas, and with a message complexity (in its lookups) of $O(\sqrt{n/d_h} \cdot (T(G) + \log n))$. This is notably worse than DHTs, but is a considerable improvement over other (essentially linear-time) lookup techniques in networks that cannot support a structured protocol, and a vast improvement over flooding-based searches. Furthermore, as we shall see, LMS provides a high degree of fault-tolerance.

Note that the LMS protocol provides a *probabilistic* guarantee of success. Depending on the number of replicas placed and the number of search probes, an object may not be found even if it exists in the network, though the probability of failure can be made arbitrarily small. In Section 4, we derive expressions for the number of necessary replicas and probes for specific probabilities of success, for arbitrary graphs G. In particular, we deal with the following problem. Let D be an arbitrary distribution on a set of size k. Suppose we sample $r+s$ times independently from D; what is a good upper bound on the probability that the multiset of the first r samples, is disjoint from the last s samples? We present an upper-bound of $\exp(-\Omega(s \cdot \min\{r/k, 1\}))$, assuming without loss of generality that $s \leq r$. At first sight, this may appear related to the "birthday paradox", and D being the uniform distribution may appear to be the worst case. This turns out to not be true — this problem is more complex, as is illustrated in Section 4. Our proof approach also facilitates a proof of the fault-tolerance of LMS.

The object placement component of LMS distrubutes replicas randomly throughout the network. This means that even if the LMS lookup is not used (e.g. for multi-attribute searches when the object id-based lookup is not applicable) flooding will succeed with high probability even with relatively small bounded propagation. LMS can also be augmented with index-based searches in the same manner as DHTs. This flexibility suggests that LMS could become the underlying platform of choice for building wide-area applications.

We start with a description of related work in Section 2. The primary contribution of this paper is the base LMS protocol described in Section 3, for which we derive expected and worst-case performance bounds in Section 4. A particularly novel feature of our analysis is that the lower bounds hold regardless of G. We also analyze, in Section 5, the performance of LMS over realistic topologies via a set of comprehensive simulations on networks with 10^5 peers and over a testbed with 512 peers. In Section 6, we finally conclude.

2. RELATED WORK

Gnutella [8] is probably the most-studied unstructured peer-to-peer network; much work has been done to understand Gnutella dynamics [18, 20, 4]. The original Gnutella search protocol was based on naive flooding. An improved flooding algorithm is discussed in [9]; it reduces the number of messages per search while still reaching the entire network. More sophisticated search techniques based on random walks are described in [10]. Their analysis yields a search cost of $\frac{n}{r}$, where n is the size of the network and r is the number of replicas, although it is based on the assumption that replicas are placed on uniformly random nodes, while the paper does not describe any placement protocol. As a comparison, LMS achieves a search cost of $O(\frac{n}{rd_h} \log n)$. In Section 5 we directly compare LMS to the best protocols described in [10].

The Gia system [3] improves on Gnutella using: topology adaptation, flow control and biased random walks. We do not compare LMS and Gia, because our model does not allow topology adaptation and, by design, the LMS substrate is not responsible for node heterogeneity and node congestion, without which flow control and biased random walks are of no use. Note that one can implement the latter two techniques on top of LMS.

Highly-efficient DHT lookup algorithms (e.g., [21, 19, 22, 15, 11]) impose structure on the topology in order to achieve their performance bounds. Beehive [14] is a replication framework that can be built on top of a DHT. These systems employ the same namespace virtualization of which we make use. A virtualized namespace has previously been used on an unstructured topology in Freenet [5], though for a different reason.

In [1] the authors build a P2P system where publishing involves storing on s random nodes in the graph and searching involves querying s independently chosen random nodes. Unlike LMS, [1] requires that the nodes may modify the topology of the neighborhood graph. The authors prove a result that is similar to Theorem 4.1(i) in this paper, although with a different technique; we thank a referee for bringing this result to our attention. Other differences are: (a) LMS restricts replica placement to local minima, which improves efficiency significantly; (b) our Theorem 4.1 is generalized beyond the special case in [1]; and (c) our proof technique for Theorem 4.1 also helps us show that LMS is robust under the loss of some replicas.

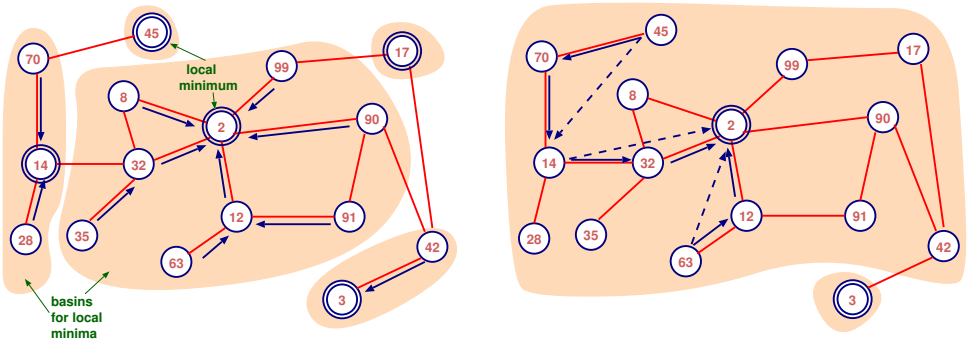
YAPPERS [6] is a P2P system that, like LMS, assumes a "given topology" model and combines structured and unstructured designs. In YAPPERS, nodes publish an item by storing it at a node of the 2-hop neighborhood, which is structured as a small DHT. Search is achieved by flooding all and only the nodes in the network that *might* have the item, and requires nodes to keep state for an extended local neighborhood (within 5 hops). The authors do not present a formal analysis.

Bloom filters [2, 12], compact digests of sets of elements, have previously been considered for structured topologies [16] as a way to provide fast lookup. In this paper, we present an application of Bloom filters on a completely unstructured topology. Theoretical properties of random walks have been exploited by [7] to improve search in unstructured networks.

3. LMS PROTOCOL DESCRIPTION

We assume a system of principals (*nodes* or *peers*) structured as an overlay network on top of a communications infrastructure such as the Internet. The topology of this overlay network is dictated by external requirements, and its maintenance is beyond the scope of our protocol. We assume that nodes can communicate with one another if and only if they are neighbors in the overlay topology.[1]

[1]This assumption can be weakened and communications



(a) 1-hop Neighborhoods

(b) 2-hop Neighborhoods

Figure 1: Local minima, basins and deterministic forwarding. Nodes are labelled with their distances from the key. Concentric circles denote local minima, and shaded regions their basins. Arrows indicate the path from a node towards its local minimum. (a) All forwarding arrows for $h = 1$. (b) Forwarding arrows for two nodes with $h = 2$. A dashed arrow indicates the target towards which a node forwards a probe when different than the next hop.

Nodes in the system have unique identifiers generated uniformly at random from an *ID space* of all bit strings of some length λ. λ must be chosen large enough to guarantee uniqueness with high probability (for example, 160 bits). How this identifier is constructed is in general application-specific.[2]

We define a node v's *neighborhood* as the set of nodes at most h hops away from v in the overlay. For each of these nodes, v knows its unique identifier, how many hops away it is, and v's next hop towards it. We do not present a protocol for propagating this information, but it is easily implemented by repeated exchanges of a node's known $h-1$-hop neighborhood. Note that nodes have no knowledge of the topology beyond this neighborhood.

3.1 Protocol overview

LMS enables nodes to publish data items (i.e. files, documents) and to retrieve data items published by others. Items published by the system are given *key identifiers* (or simply *keys*) in the same ID space as the nodes, for example by hashing the name or contents of the item. The protocol then attempts to store an item at nodes that are close to its key in the (circular) ID space. In terms of the distance between a node and key, we do not find the global minimum (as in a distributed hash table), but rather a *local* minimum.

Publishing an item involves storing *replicas* of the item at a number of randomly selected local minima; retrieving an item involves querying randomly selected local minima until one is found that holds a replica. Crucially, the distribution of these random selections is typically not the uniform distribution; it is a function of the underlying topology, and is naturally generated by the distributed algorithms that we employ. Intuitively, the more replicas are placed, the easier it will be to find one of them. Random minimum selection is accomplished by performing a random walk through the overlay before actively looking for a local minimum.

made possible over larger distances in the overlay. This does not substantially alter the protocol, other than to allow optimizations that reduce the number of messages exchanged.

[2]For instance, the SHA-1 hash of a node's public key.

3.2 The basic protocol

For clarity of the exposition, we first present a slightly simplified version of the LMS protocol. In this version, placing and locating items are essentially identical. We will then refine this to account for the differences between the two operations and to add optimizations.

To select a random local minimum, a node generates a *probe* message, which has the general form $\langle$`probe`, *initiator, key, walk_length, path*$\rangle$. A probe moves through the network with a *random walk* followed by a *deterministic walk*. The *walk_length* parameter is initialized to some positive value. A node that receives the probe (including the initiator) first examines this parameter. If it is greater than zero, the probe is in the random walk phase, and the node forwards it to a randomly selected neighbor[3] and decrements the value. If *walk_length* is zero, the probe is forwarded according to the deterministic walk, which is described below. In either case, a node appends itself to *path* before forwarding the probe so that a local minimum can direct its response back to the initiator.[4]

Deterministic walk. Deterministic forwarding is done using a greedy algorithm. A node v receiving a probe computes the distance between *key* and every node in its neighborhood including itself. We define the distance between a key x and a node w as $d(x, w) = \min\{x - w \bmod 2^\lambda, w - x \bmod 2^\lambda\}$.

Let v' be the node that minimizes $d(key, \cdot)$ over the neighborhood of v. If $v = v'$, then v is the local minimum; it then stores or returns a replica, depending on the type of probe. Otherwise, v determines the next-hop node towards v' (from its local knowledge of the topology) and forwards the probe

[3]The distribution for this selection is uniform in our case, but nonuniform choice is also possible. For instance, if the topology is dictated by trust relations, more highly trusted nodes might be given greater weight in selection. Another possibility, introduced in [3], is to weight the distribution according to resource availability.

[4]If messages can be exchanged by nodes that are not direct neighbors, then *path* is not needed, since the local minimum can communicate directly with the initiator.

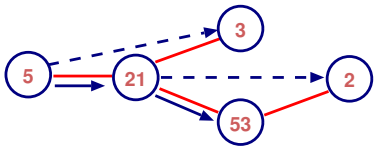


Figure 2: An illustration of "unexpected" message forwarding. Node 5 sees 3 in its 2-hop neighborhood, and forwards the probe to 21, the next hop towards 3. Node 21 sees a better match in 2, and so forwards the probe to 53 rather than 3.

to this node. Recall that we assume v cannot communicate directly with v' if it is more than one hop away.

For a local minimum of a key x, v_x, we define the set of nodes that deterministically forward to this minimum as its *basin of attraction* (or *basin*). Note that while it is possible for a local minimum to "attract" nodes from outside of its neighborhood, it is also possible for the basin to be only the minimum itself. This is illustrated in Fig. 1. It is also possible that a message "intended" for a node v' will be redirected towards a better minimum v'' at an intermediate hop, as shown in Fig. 2.

3.3 Protocol details

Replica placement. We now distinguish between probes for placing replicas and those for locating replicas. A `REPLICA-PLACE` probe is initiated by the owner of an item, and includes an additional field *item*. When a local minimum receives a `REPLICA-PLACE` probe, it checks if it already holds a replica, and if not whether it has the resources to store one. If the minimum is able to store a replica of the item, it informs the initiator.

When a local minimum cannot store a new replica, it performs *duplication avoidance*: doubling the initial random walk length and restarting the probe's random walk. Because the probe is starting from a new location, it is less likely to again reach a duplicate minimum than if it were started with the same random walk length from the initiator.

We must limit the number of times that duplication avoidance is invoked, because the owner of an item might attempt to place more replicas than there are local minima for the item's key. This leads to `REPLICA-PLACE` probes that circulate through the network indefinitely, with progressively longer random walks. Consequently, we include a parameter initialized with the maximum number of failures permitted. Duplication avoidance decrements this parameter, discarding the probe when it becomes negative. The final form of a `REPLICA-PLACE` probe is then ⟨`REPLICA-PLACE`, *initiator, key, walk_length, item, initial_walk_length, failure_count, path*⟩.

Replica lookup. A `SEARCH-PROBE` performs a lookup operation on an item key. A local minimum that receives a `SEARCH-PROBE` checks whether it holds a replica of the item and returns either the replica or notice of failure to the initiator. The form of a `SEARCH-PROBE` is simply ⟨`SEARCH-PROBE`, *initiator, key, walk_length, path*⟩.

Note that LMS is a probabilistic algorithm. The probability of a single `SEARCH-PROBE` locating a replica is fairly low, so a node will have to initiate a number of probes when looking for an item. This can be done in serial or in parallel. The former increases the expected time until the node receives a successful response, while the latter increases the load on the system.

Variant: Bloom filters. The efficiency of item lookup can be improved by employing Bloom filters. If a replica holder propagates a digest of the items it replicates out some distance h_B through the network, it becomes simple for a node participating in a lookup to check whether it is likely that a particular node within h_B hops can satisfy the lookup request. This variant has a cost, which is the periodic transmission of h_B Bloom filters to each direct neighbor. Each of these filters can be large. For example, for $h_B = 2$, in a graph where each node has degree 4 and holds 100 replicas, the size of each filter is about 1.3kbytes. We describe this variant in greater detail in the full version.

3.4 The adaptive protocol

Having only local information, a node has no *a priori* way to know how many replicas of its items it must place in order for them to be easily found by other nodes. Increasing the number of replicas reduces the average number of search probes needed to find a replica, and vice-versa. Search performance thus provides a feedback mechanism to determine if an item has been replicated sufficiently.

The *adaptive protocol* is the component of LMS that determines and dynamically adjusts the number of replicas to be placed. It is run periodically for each item by its owner and ensures that a sufficient number r of replicas are available at any time. Let s be the average number of probes needed to find the item at a given time. The adaptive protocol ensures that $s \approx f(r)$, where f is an arbitrary function, chosen by the owner.[5] For simplicity of the exposition we present the protocol for the case $f(r) = r$ (see Section 4). The overhead of this protocol is minimal, aside from any additional replica placements, as it leverages existing actions.

A search from a random node in the system takes on average s probes to find a replica of an item I owned by node v. Node v periodically learns an estimate of s, as explained below; from the current number of reachable replicas r, it computes the number of replicas r' that it adaptively determines are needed as $r' = \lceil \alpha r + (1-\alpha)s \rceil$. Here α is a hysteresis parameter between 0 and 1 (we use 0.9) that controls how sensitive the algorithm is to fluctuations in s. We define $\delta = r' - r$ as the needed change in number of replicas, and a non-zero value results in either placing or deleting replicas.

Since searches are a normal and frequent part of the system's operation, it is a simple matter for a node v_i to keep track of the number of probes s_i it needed to send before finding a replica of I.[6] If s_i is included in a `SEARCH-PROBE` message, the replica holder can store it and forward a batch of probe results to v periodically. These results provide a representative sample from which v can estimate the average s. Note that nodes on the threshold of being able to find replicas of I will tend to drive r up, allowing new nodes to find replicas and improving the sampling over the network.

[5]For example, this could be used to take the popularity of an item into account by storing many more replicas so that most searches require very few probes.

[6]For an unpopular item, the owner of an item might select random nodes and request that they perform searches for the item to make up for the lack of search feedback.

4. ANALYSIS OF LMS

We now give a rigorous analysis of the protocol; we make no assumptions about the topology layer, other than the necessary condition that the topology, which is a graph G, is connected. The three main results derived are: (a) tight bounds on the probability that a search in LMS fails; (b) bounds on the expected walk-length (i.e. the number of nodes visited by a search probe, Section 3.2) conditional on successful search; and (c) robustness under failures. We also derive the asymptotic performance of LMS. In the full version, we briefly analyze a variant of the protocol that includes Bloom filters.

Probability of successful search. Consider an item x. Let r denote the number of replicas of x that the owner of x places, and let s be the number of search probes that another node conducts to search for x. We derive an upper bound on the probability that the search fails. If G is regular and "symmetric" (e.g., if it is a random regular graph), we expect the r replicas of a key x to be placed uniformly at random at the k local minima for x; similarly for the s locations at which the s searches end. In such a case, the probability of "failure" (unsuccessful search) is essentially of the form $(1 - \frac{r}{k})^s \leq \exp(-\frac{rs}{k})$, where $\exp(a)$ denotes e^a. However, we desire protocols that work for time-varying and arbitrary topologies G where the distribution may be quite non-uniform. Our first main result is that the failure probability is always bounded by a function of the form $\exp(-\Omega(rs/k))$, in the case of interest where r and s are bounded by $O(k)$. (Indeed, if r or s is $\Omega(k)$, we get a trivial linear-time algorithm.)

THEOREM 4.1. *For any connected topology G, the following bounds on the failure probability hold:*
(i) $\exp(-(s^2/k) \cdot (1 - s/k))$*, if* $s = r$*;*
(ii) $\exp(-\Omega(s \cdot \min\{r/k, 1\}))$*, if* $s \leq r$*; and*
(iii) the bound of (ii) with s and r interchanged, if $r < s$*.*

Note: The bound of (i) becomes trivial if $s \geq k$; in such cases where $s = r = \Omega(k)$, we can use (ii) to get a bound of $\exp(-\Omega(s))$.

We present the main ideas of the proof here, and defer further details to the full version.

PROOF. Let n denote the number of nodes in G. If we conduct a random walk on G for a "large enough" number $T = T(G)$ of steps, then the distribution of the last (i.e., Tth) vertex visited, approaches a unique *stationary distribution SD, no matter where we started.* (Technically, we need to stay at each vertex with probability 1/2, and move with probability 1/2, in case G is bipartite. Also, SD places probability $d(v)/(2m)$ on each node v, where $d(v)$ is the degree —number of neighbors— of v, and m is the total number of links.) For most natural models of randomly chosen/evolving graphs (such as random regular graphs), $T = O(\log n)$ suffices; in the pathological worst case, T being a suitable constant times n works for any graph. We choose T appropriately, and take our parameter `INITIAL-WALK-LENGTH` to be at least T; in our simulations, choosing `INITIAL-WALK-LENGTH` to be 3 and doubling its value each time a duplicate local minimum is found is sufficient to obtain very good results.

The above-seen property of the stationary distribution SD, makes our setting concrete as follows. Fix a key x. Let r denote the number of replicas of x that we place, and s be the number of walks that we conduct to search for x. Our goal is to show that even when r and s are only moderately large, a search for x will succeed with high probability, no matter what G is. We proceed as follows. Let D be the probability distribution of choosing a vertex w (which is a local minimum for x) as follows: choose a vertex v using distribution SD, and then deterministically go to a local minimum w starting at v, as described in Section 3. Then, replica-placement is as follows: choose a multiset R of r local minima by sampling r times independently from D, remove duplicates from R, and place the replicas at the locations in R. Search for x is as follows: choose a multiset S of s local minima by sampling s times independently from D, and search is successful iff S intersects R.

We arrive at the following question: let $K = \{1, \ldots, k\}$ be the set of k local minima for x, and suppose we choose multisets R and S as in the previous paragraph. When can we show that $\Pr[R \cap S \neq \emptyset]$ is high? The heuristic argument a few paragraphs above shows that if D is the uniform distribution on K, then, the probability of unsuccessful search is of the form $\exp(-\frac{rs}{k})$; this Theorem shows that this upper bound indeed holds for any distribution D if $r, s \leq k$. It seems reasonable to conjecture that D being the uniform distribution is the worst case, since otherwise elements of K that have high probability could appear in both R and S with increased likelihood. This intuition turns out be false: consider the case where $k = 2$, $s \ll r$, and D places probabilities p and $1 - p$ on the two elements of K. The failure probability here is $p^r(1-p)^s + p^s(1-p)^r$. If D is the uniform distribution (i.e., $p = 1/2$), then the failure probability is $2^{1-(r+s)}$; however, the failure probability can be made much larger — about $(s/(r+s))^s \cdot \exp(-rs/(r+s))$ — by setting $p = s/(r+s)$. Similar counterexamples can be constructed for all k. In general, the case where r and s are quite different, needs care.

We first note a stumbling block: given that one element of S did not lie in R, the conditional probability of this happening for another element of S can go up! Thus we have an undesirable positive correlation among these events: in particular, if y is the probability that an arbitrary single element of S does not lie in R, then the probability of unsuccessful search is *at least as large as* y^s. We take a different approach, wherein the correlations actually help us. Recall that $K = \{1, 2, \ldots, k\}$, and let X_i be the event that i lies in both R and S. As usual, $\overline{X_i}$ denotes the complement of X_i; letting p_i be the probability that distribution D places on i and setting $q_i = 1 - p_i$, we have $\Pr[\overline{X_i}] = (q_i^r + q_i^s - q_i^{r+s})$.

The probability of unsuccessful search is $\Pr[\bigwedge_{i=1}^{k} \overline{X_i}]$. Our first key idea is that the events $\overline{X_i}$, $i = 1, 2, \ldots, k$, are negatively correlated! Intuitively, this seems clear: given that none of $X_1, X_2, \ldots, X_{i-1}$ held, this informally "leaves more slots free" for X_i to occur. We can prove this basically showing that the event "$\bigwedge_{j=1}^{i-1} \overline{X_j}$" is positively correlated with the event X_i. This negative correlation leads to the bound

$$\Pr[\bigwedge_{i=1}^{k} \overline{X_i}] \leq \prod_{i=1}^{k} \Pr[\overline{X_i}] \quad (1)$$

$$= \prod_{i=1}^{k} (q_i^r + q_i^s - q_i^{r+s}). \quad (2)$$

Part (i) of the theorem can now be proved by using the concavity of the function $z \mapsto \ln(f(z))$ over the domain $z \in (0,1)$, where $f(z) = 2z^r - z^{2r}$; we defer the details to the full version. We now consider part (ii) of the theorem, where $s \leq r$. As shown in the example with $k = 2$ a few paragraphs above, this needs more care. In particular, the analog of the function f above, can have the property that $\ln f$ is neither totally convex nor totally concave in the domain $z \in (0,1)$. The basic ideas here are as follows. Recall the probabilities p_i from above. Partition K into three sets:

$$\begin{aligned} C_1 &= \{i : p_i \leq 1/r\}; \\ C_2 &= \{i : 1/r < p_i \leq 1/s\}, \text{ and} \\ C_3 &= \{i : p_i > 1/s\}. \end{aligned}$$

The product in (2) now splits into three products, which we will analyze separately and then combine.

For $j = 1, 2, 3$, let $v_j = \sum_{i \in C_j} p_i$ and note that the sum $v_1 + v_2 + v_3 = 1$. Let us first consider any $i \in C_1$. Since $0 \leq p_i \leq 1/r$, it is easy to show here that

$$q_i^r + q_i^s - q_i^{r+s} = 1 - \Theta(rsp_i^2).$$

Next, basic convexity arguments show that for any constant $\alpha > 0$, and any given values for $|C_1|$ and v_1, the quantity $\prod_{i \in C_1}(1 - \alpha rsp_i^2)$ is maximized (as a function of the variables p_i) when $p_i = v_1/|C_1|$ for each $i \in C_1$. Further simplification yields

$$\prod_{i \in C_1} (q_i^r + q_i^s - q_i^{r+s}) \leq \exp(-\Omega(rsv_1^2/|C_1|)). \tag{3}$$

Next consider C_2. In this case,

$$q_i^r + q_i^s - q_i^{r+s} = 1 - \Theta(sp_i),$$

and a simple argument yields

$$\prod_{i \in C_2} (q_i^r + q_i^s - q_i^{r+s}) \leq \exp(-\Omega(s \cdot v_2)). \tag{4}$$

The set C_3 is where the function $q_i^r + q_i^s - q_i^{r+s}$ exhibits complex behavior. Fortunately, we can deal with it as follows. For any $i \in C_3$, $q_i^r + q_i^s - q_i^{r+s}$ is at most

$$q_i^r + q_i^s \leq 2 \cdot q_i^s \leq 2 \cdot \exp(-sp_i) \leq \exp(-\Omega(sp_i)),$$

where the final inequality follows from the fact that $p_i \geq 1/s$. Therefore,

$$\prod_{i \in C_3} (q_i^r + q_i^s - q_i^{r+s}) \leq \exp(-\Omega(s \cdot v_3)). \tag{5}$$

Putting (3), (4) and (5) together with (2), we get that for some constant $\gamma > 0$, $\Pr[\bigwedge_{i=1}^k \overline{X_i}]$ is upper-bounded by

$$\exp(-\gamma \cdot s \cdot (rv_1^2/|C_1| + v_2 + v_3)),$$

which equals $\exp(-\gamma \cdot s \cdot (rv_1^2/|C_1| + 1 - v_1))$; this, in turn, is at most $\exp(-\gamma \cdot s \cdot (rv_1^2/k + 1 - v_1))$. Elementary calculus that determines the maximum of this last expression as a function of v_1 (where v_1 takes values in $[0,1]$), yields the bound of part (ii) of the theorem.

Finally, part (iii) of the theorem follows by symmetry. □

	Search cost	State
general case	$O(\frac{n}{rd_h} \log n)$	$O(d_h)$
$r = d_h = \Theta(n^{1/3})$	$O(n^{1/3} \log n)$	$O(n^{1/3})$

Table 1: Asymptotic performance of LMS. n is the number of nodes, d_h is the minimum size of a h-hop neighborhood and r is the number of replicas.

Expected walk-length. A second quantity of interest is the number N of nodes visited by any single search probe. This, multiplied by the number s of probes, yields the total cost of a search query (similarly, we get a multiplier of r in the replica-placement). We can write $N = T + l$, where T is the length of the random walk and l is the length of the deterministic walk. As we discussed above, $T = O(\log n)$ will work for practical networks. As far as l is concerned, we present the following result which shows that $l = O(\log n)$ with high probability:

THEOREM 4.2. *For any graph G of n nodes and for any positive constant c, the number l of steps of any deterministic walk to a local minimum is at most $(c+1)\log n$ with high probability (at least $1 - 2/n^c$).*

The proof of this is deferred to the full version.

Asymptotic performance of LMS. Part (ii) of Theorem 4.1 allows us to estimate the performance of LMS. In particular, for $r, s \leq k$, the probability that a search fails is of the form $\exp(-\Omega(rs/k))$. Therefore, for any given k, we can choose $rs = \Theta(k)$ to make the failure probability an arbitrary small constant. For example, if $r = s = \lceil 2\sqrt{k} \rceil$, the probability of unsuccessful search is essentially at most $e^{-4} \sim 0.018$. (In fact, this upper bound only holds for relatively regular graphs; as G gets more irregular, the failure probability becomes smaller.)

Note that the number of local minima is, *in expectation*, $k = O(n/d_h)$, where d_h is the minimum h-hop neighborhood size in the graph. This means that, in any class of graphs where $d_h = d_h(n)$, if LMS places $r = r(n)$ replicas, then a search requires $s = O(\frac{n}{rd_h})$ probes. This translates into a search cost of $O(\frac{n}{rd_h} \log n)$ by virtue of Theorem 4.2. As far as the *routing state* kept by a node, this is proportional to the size of its h-hop neighborhood. Since this can be much larger than d_h, we assume that a node has a cap $\Delta = O(d_h)$ on the number of nodes in its neighborhood it is willing to keep state for.

Table 1 summarizes the asymptotic performance of the protocol. The table also shows, for concreteness, the interesting special case, where $d_h = \Omega(n^{\frac{1}{3}})$ (i.e., the given topology has h-hop neighborhoods that are not too small) and where we choose r such that r and s are of the same order of magnitude.

Fault-tolerance. We now investigate the performance of LMS when some of the replicas are lost. We consider a very strong failure model that makes no assumptions about the timings of the failures: failures can even occur at the instant after the s searches have been completed. In particular, we have the following. Let Y be the random variable denoting the set $R \cap S$, assuming no faults thus far; i.e., if no faults have occurred thus far, the search has found $|Y|$ replicas of x.

In an adversarial model where the adversary can arbitrarily delete up to t replicas of x, the adversary can wait until the searches have just terminated, and then delete up to t replicas from Y. Thus, the search succeeds iff $|Y| \geq t+1$. Similarly, in a "random failure" model where each replica survives independently with probability p, each element of Y can get deleted with probability $1-p$. In such failure models, we can show the following: if the expected value $E[|Y|]$ is at least as large as a necessary lower bound, our search will succeed with high probability. In particular, in the adversarial model, we require $E[|Y|] \geq t + c_1\sqrt{t}$ for a constant t, where c_1 is some constant, for a good probability of success; this is nearly best-possible in the sense that there is another constant c_2 such that if $E[|Y|] \leq t + c_2\sqrt{t}$, then the success-probability can be very low. Similarly, in the random-failure model, we have the near-optimal result that $E[|Y|] \geq c_3/p$ suffices. In the specific case of near-regular networks, these requirements say that: (i) in the adversarial model, an overhead of only $O(\sqrt{t})$ in the number of replicas and in search time suffices, as compared to the no-fault case; and (ii) in the random-faults model, an overhead of $O(1/\sqrt{p})$ suffices. All these results are obtained by using the negative-correlation result (1) with some large-deviation bounds [13]. We present just a short proof-sketch here. We are interested in giving an expression to the expectation of the size of Y and in bounding the lower-tail of $|Y|$, i.e., we aim to show that $|Y|$ is not much below its mean, with high probability. Clearly, we have $|Y| = \sum_{i=1}^{k} X_i$, where the X_i are as in (1).

As far as the expectation of $|Y|$ is concerned, linearity of expectation gives:

$$\mathbf{E}[|Y|] = \sum_{i=1}^{k} \Pr[X_i] = \sum_{i=1}^{k} (1-q_i^r)(1-q_i^s) \qquad (6)$$

where the q_i are as in the proof of Theorem 4.1. Equation (1) extends to any set of indices: for any $C \subseteq \{1, 2, \ldots, k\}$,

$$\Pr[\bigwedge_{i \in C} \overline{X_i}] \leq \prod_{i \in C} \Pr[\overline{X_i}].$$

The results of [13] can be used to show, in conjunction with this negative-correlation property, that we can apply the Chernoff bound to the lower-tail of $|Y|$ (although the X_i are not independent). This strong property leads to our fault-tolerance results; the details are deferred to the full version.

5. EXPERIMENTAL RESULTS

In this section we present an experimental study of LMS using a simulation (described in the full version) both to demonstrate large-scale behavior of the algorithm and to compare its performance on various topologies and with different extensions to the base protocol. In addition, we present results from a complete implementation of the base protocol.

5.1 Lookup performance

The results of the first set of experiments are shown in Table 2. The Gnutella graph represents a crawl of the actual deployed system [17, 18], while the others are generated. One randomly generated item is replicated in each trial, and the number of replicas is determined by an iterative procedure so that the average number of lookup probes needed to find one replica is approximately equal to the number of replicas. The "Visited" column shows the cost of the search either using the basic LMS protocol (the "LM" column) or the Bloom filter variant with $h_B = 2$ (the "LM+BF" column, see Section 3.3). The costs listed are the total number of nodes visited by the lookup probes until a replica is located (including all unsuccessful probes, executed serially).

Graph Type	Size	Avg. deg.	# Repl.	Visited LM	Visited LM+BF
power-law	10K	4.11	3	17.7	4.4
random	10K	4.11	22	131.1	21.8
Gnutella	61K	4.7	16	83.9	15.7
random	61K	4.7	45	282.8	43.8
random	100K	17	14	55.9	14.0
random	100K	12	19	87.1	19.0
random	100K	7	34	185.4	34.0

Table 2: Lookup performance for different graphs. The average number of lookup probes is approximately equal to the number of replicas. Measured quantities are averaged over 10,000 trials for each of 60 generated graphs, excluding the Gnutella graph.

The 10K graphs were chosen for comparison with the results of [10]. The most successful protocol in [10], *check*, visits approximately 150 nodes (slightly more than our 130), but requires over 90 replicas in contrast to our 22. As the graphs grow larger, LMS continues to perform reasonably. On the Gnutella graph, in particular, LMS requires relatively few replicas and lookup probes; we expect this graph to be typical of many applications with unstructured topologies. Note that the Bloom-filter-based lookup performs considerably better than the standard LMS lookup, but this comes at the cost of maintaining and propagating Bloom filters.

In this experiment, we assume homogeneous nodes we do not take into account the effects of node congestion. Further our model does not allow a node to add neighbors. Therefore we do not discuss a comparison with Gia here (see Section 2).

Random graphs show the worst-case performance of LMS. In order to demonstrate the lower-bound behavior of the protocol, we restrict our experiments to these graphs hereafter.

5.2 Number of replicas vs. lookup overhead

In this section, we further consider details of LMS performance. We introduce the notion of an *iso-success curve*. An *iso-success* curve, for a given graph G and a given probability p, is the set of all pairs (r, s), such that if the owner of an item places r replicas and a node v uses *up to* s search probes to search the item, then the probability that v finds the item is p.

Iso-success curves precisely capture the tradeoff between number of replicas and lookup overhead. In Fig. 3, we present the iso-success curves for a single random graph of size 100K nodes (average degree 17). The figure shows the *worst-case* number of probes required for each number of replicas to achieve the given level of success probability (without using Bloom filters). For example, a little more than 20 probes were required with 10 replicas for 70% probability of finding an item. The curve was obtained by measuring the distribution of the number of probes for each

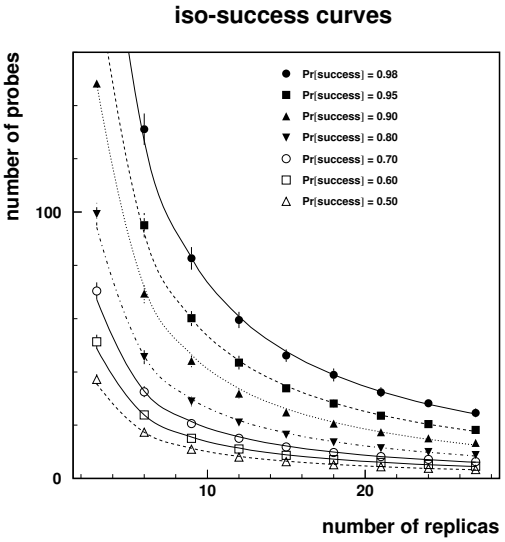


Figure 3: Number of probes as a function of the number of replicas. Random graph with 10^5 nodes, of average degree 17. The numbers labeling the curves are the particular success probabilities. The curves show fits to the functional form $f(r) = \frac{a}{r} + b$.

number of replicas, using 10,000 trials.

For each success probability, the average number of search probes for each number of replicas was fit to the function $f(r) = \frac{a}{r} + b$, using the standard deviation of the numbers of probes as the uncertainty in their average. The fits demonstrate that $rs = Constant$ for a fixed probability is a very good approximation (b is small and negative in all fits). A global fit to all of the data reveals that approximately half of the local minima dominate the network, greatly reducing the expected number of replicas needed for reliable lookup ($k/2$ rather than k in the results of Section 4).

Along with the tradeoff measure, the iso-success curves also provide us with a snapshot of the distribution of the expected number of probes required for lookups on a given graph. For example, with 15 replicas, 50% of the lookups require less than 8 probes (in the worst case), and only 5% of the lookups required more than 40 probes. This information is useful in implementation since it provides a strategy for fixing the number of probes that should be launched in parallel in order to reduce lookup latency.

5.3 Failure analysis

LMS is extremely robust, and in this section, we analyze its resilience under the failure model described in Section 4. Specifically, we consider random failures in which a given fraction of nodes in the networks fail. In these results, we consider how many replicas should be provisioned to handle specific failures (and validate our analysis). In these experiments, the adaptation is via static provisioning, i.e. the application/system designer places extra replicas. In later simulation (Section 5.4) and implementation results (Section 5.5), we consider how the system can adapt at run-time as it becomes aware of new failures.

In Table 3, we consider a random graph with 100,000 nodes (average degree 7). We consider that each replica, after being placed, can independently fail (the node discards the data) with the probability specified in the first column. In each case, we present an average over 10,000 runs of the number of replicas that needed to be placed *before the failures* such that the probability of a successful search *after the failures* is at least 99%. We present the specific placement that equalizes the number of search probes and the number of replicas.

Failure Probability	# Replicas	Average Visited	Analytic Bound
0	36	188	36
0.1	38	200	37.9
0.2	41	213	40.2
0.3	45	231	43.0
0.4	48	262	46.5
0.5	53	289	50.9

Table 3: Performance under random failures: random graph with 10^5 nodes, avg. degree of 7. The analytic bound predicts the number of replicas which in this case is equal to the number of probes.

If f is the failure probability, the number of replicas that equalizes the expected number of probes is $r(f) = \frac{r(0)}{\sqrt{1-f}}$. This analytic prediction is the last column of the table. It is clear that the analysis is extremely accurate in this case, and that LMS scales extremely well with failures. The most important point to note here is that the number of replicas only has to scale with the square root of the fraction of failures, which is an extremely positive outcome.

5.4 Performance under high churn

We now examine the behavior of LMS in a network with a high rate of nodes leaving and joining. The topology is a random graph of 10K nodes of average degree 7. A departing node is immediately replaced by a new node with the same number of neighbors; this maintains both the size of the graph and the degree distribution. Bloom filters are not used in this experiment. The neighborhood distance h is 2.

At the start of the simulation, a node u creates one replica of its item. Every minute (in simulation time) a random node searches for the item (over 99% of all searches succeed). The simulator also chooses q nodes at random (excluding u) to remove, where $q = 10000/T$ for an average node lifetime of T minutes. Every three minutes, the item's owner u selects a node by initiating a random walk (of length 6) and has that node perform a search for the item. Based on the result of the search, u applies the adaptive protocol (Section 3.4); placing or removing replicas as needed. The number of replicas r needed is based on a *cost ratio*, an input parameter defined as r/s, where s is the expected number of probes to find the item[7]. The adaptive protocol then attempts to equalize the replication cost and the total cost of all (expected) searches.

The results of the experiment are shown in Table 4, which averages results over seven initial random graphs. The search cost is collected from the one-minute lookups. The adaptive overhead includes the cost of the search initiated by u and the cost of placing additional replicas. The search cost and

[7]In the notation of Section 3.4, this means running the adaptive protocol with $f(r) = r/(\text{cost ratio})$.

Cost Ratio	Node Lifetime	Search Cost	Adaptive Overhead	Average Replicas
2	15	44.2	69.8	16.4
	30	35.9	63.4	20.3
	60	37.2	53.2	18.6
5	15	29.6	116.6	27.4
	30	35.7	70.7	23.8
	60	32	62.1	25.8

Table 4: Cost of searching, publishing and maintaining the overlay in basic LMS, in presence of high node churn. Lifetimes are measured in minutes.

number of replicas are fairly stable over all scenarios. The adaptive overhead is also relatively stable, with the exception of 15-minute lifetimes for a cost ratio of 5. Note that we do not include the overhead of maintaining the network, as it is external to the LMS protocol.

5.5 Implementation

We have implemented the complete LMS protocol (without Bloom filters). Multiple nodes are run on a single host, enabling us to test fairly large networks. The topology is maintained by a separate topology server that creates edges between nodes at random while enforcing a minimum degree for each node. All nodes are fail-stop; as nodes leave the network their neighbors obtain replacements as needed to maintain the minimum degree. The available bandwidth for each host and the capacity of the topology server define the limits of the network size that we were able to construct. The deployment was limited to locally available hosts.

Two experiments were performed, one with 512 nodes and one with 1024, both focusing on the behavior of the adaptive protocol, and both with a minimum node degree of 3 (average 3.9 ± 1.1) and $h = 2$. For the smaller network, we investigate a system in which searches and adaptation are frequent. The purpose of this experiment is to demonstrate that our protocols are functional in a real deployment under high (for the limited resources available) load. The behavior of the adaptive protocol is shown in Figure 4. Each node replicates one item[8] and performs a search for a random item every 10 seconds (staggered by random offsets at start-up), so that a new search is started roughly every 20 milliseconds. The adaptive protocol is run every 20 seconds (again staggered), and is based on the results of these searches. The average number of replicas tends to converge on a value dependent on the specific topology. The standard-deviation error band around the average number of replicas is dominated by the variation in the number of search probes, which we expect to be close to 1, as observed.

LMS probes increase in size as they propagate, but almost all are 400 bytes or smaller (excluding item size), and all are less than 1000 bytes. A successful lookup sends an "adaptive hint" back to the item's owner, the average size of which is 28.5 ± 0.5 bytes; most of this is the key identifier and can be batched by the replica holder or otherwise amortized.

[8]Note that the number of items per node is not important when considering network load. What matters is the number of probes circulating through the network, or essentially the number of searches initiated per second. Our experiments are fundamentally limited by having eleven I/O-intensive processes on a single host, forcing us to restrict the rate at which searches are initiated.

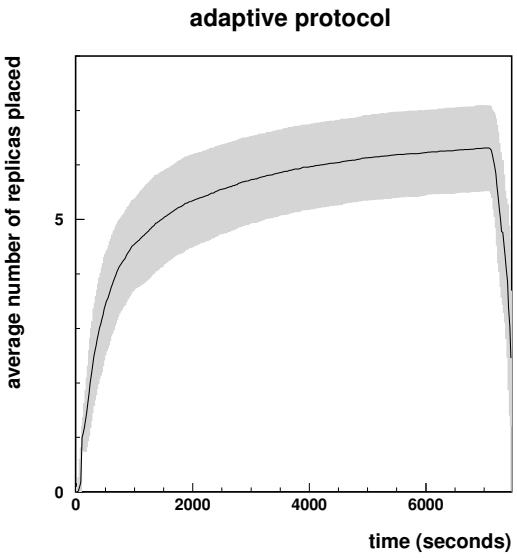


Figure 4: Effect of the adaptive protocol. The x axis shows elapsed time, with 20s between subsequent runs of the adaptive protocol. The y axis shows the average number of replicas placed for all items in the system. The shaded region indicates the one standard deviation region. The steep drop-off at the right is due to the system shutting down.

Replicas are soft state, so the owner of an item must send a refresh message (66.9 ± 0.8 bytes on average) to replica holders, which is done every 20 seconds in this experiment. Replicas that are not refreshed are garbage-collected by their holder. Neighborhood propagation consumes considerable bandwidth, but we have constructed a much more efficient protocol that only propagates changes.

The experiment with 1024 nodes was used to verify that the adaptive protocol can cope with the sudden loss of half the network. Only one item is replicated, and its owner initiates periodic searches for it. Halfway through the experiment 512 of the nodes depart. We do not present specific results of this, as providing no special insight into the system, but mention that it performed very well under extremely adverse conditions.

6. CONCLUSIONS

In this paper, we designed LMS, an efficient unstructured P2P protocol that provides a DHT-like publish and lookup functionality. Unlike a DHT [21, 19, 15], LMS is designed for a model in which the topology (i.e. the graph where a vertex denotes a peer and an edge denotes that two peers know of each other and can communicate directly) is determined by an external entity, therefore peers are not allowed to choose their neighbors. We demonstrated through analysis and simulation that LMS is an efficient protocol with stronger performance guarantees than other unstructured protocols that work in the same model. We also presented a prototype implementation, which shows the practicality of the design.

LMS is of practical interest for distributed applications relying on trust relations between nodes. These trust rela-

tions define a graph which, when traversed along its links, provides a known level of assurance for operations. Specifically, we developed a decentralized public-key infrastructure based on a web-of-trust. Traversing links in the trust graph implicitly generates certificate chains assuring a node searching for a public key of that key's correctness. More details on this application will be available in a later publication.

7. ACKNOWLEDGEMENTS

This material is based upon work supported in part by: NSF grant 0208005, ITR Award CNS-0426683, NSF grant CCR-0310499, NSF award ANI0092806 and DoD contract MDA90402C0428.

We thank the referees for their valuable comments.

Bobby Bhattacharjee and Aravind Srinivasan are also affiliated with the Institute for Advanced Computer Studies, University of Maryland.

8. REFERENCES

[1] I. Abraham and D. Malkhi. Probabilistic quorums for dynamic systems. In *International Symposium on Distributed Computing (DISC)*, pages 60–74, 2003.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[3] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM Press, 2003.

[4] J. Chu, K. Labonte, and B. N. Levine. Availability and locality measurements of peer-to-peer file systems. In *Proc. ITCom: Scalability and Traffic Control in IP Networks II Conferences*, volume 4868, July 2002.

[5] I. Clarke, S. G. Miller, T. W. Hong, O. S. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1), Jan./Feb. 2002.

[6] P. Ganesan, Q. Sun, and H. Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitrary topology. In *22nd Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.

[7] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *IEEE Infocom*, 2004.

[8] `http://www.gnutella.com`.

[9] S. Jiang, L. Guo, and X. Zhang. LightFlood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. In *International Conference on Parallel Processing*, 2003.

[10] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM Press, 2002.

[11] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192. ACM Press, 2002.

[12] M. Mitzenmacher. Compressed Bloom filters. In *20th Annual ACM Symposium on Principles of Distributed Computing (PODC '01)*, pages 144–150, Newport, RI USA, August 26–29 2001. ACM Press.

[13] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.

[14] V. Ramasubramanian and E. G. Sirer. Beehive: Exploiting power law query distributions for O(1) lookup performance in peer to peer overlays. In *Proceedings of NSDI*, 2004.

[15] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*, pages 161–172, San Diego, CA USA, August 27–31 2001. ACM.

[16] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002), Proceedings*, volume 3, pages 1248–1257, New York, NY USA, June 23–27 2002. IEEE.

[17] `people.cs.uchicago.edu/~matei/GnutellaGraphs/`

[18] M. Ripeanu and I. T. Foster. Mapping the Gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors, *Peer-to-Peer Systems, First International Workshop (IPTPS 2002)*, volume 2429 of *Lecture Notes in Computer Science*, pages 85–93, Cambridge, MA USA, March 7–8 2002. Springer.

[19] A. I. Rowstron and P. Druschel. Pastry: Scalable, distributed object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350, Heidelberg, Germany, 2001. Springer.

[20] S. Saroiu, K. P. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Syst.*, 9(2):170–184, August 2003.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*, pages 149–160, San Diego, CA USA, August 27–31 2001. ACM.

[22] B. Zhao, K. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley Technical Report, 2001.