

Efficient Lookup on Unstructured Topologies

Ruggero Morselli

Bobby Bhattacharjee

Michael A. Marsh

Aravind Srinivasan

Abstract

We present LMS, a protocol for efficient lookup on unstructured networks. Our protocol uses a virtual namespace *without imposing specific topologies*. It is vastly 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 DHT.

We present analytic bounds for the worst-case performance of our protocol. Through detailed simulations, we show that the actual performance on realistic topologies is significantly better. We also show in both simulation and a complete implementation (which includes over one thousand 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 negligible, and its performance approaches that of DHTs on networks of similar size.

We discuss several applications that are particularly suited to LMS as a replication platform. In particular, we give a model of a publishing service in a trust network and we show experimental results that confirm the advantages in this case of LMS over a DHT.

1 Introduction

We present a protocol for efficient lookups on unstructured topologies. These topologies are those most often employed by popular peer-to-peer applications where deployed systems may contain millions of peers. Unfortunately, the underlying lack of structure in these topologies usually means that even simple primitives such as key lookup are potentially resource intensive. This problem has generated much research in two dimensions: (1) development of more sophisticated lookup protocols (than flooding) on unstructured networks, and (2) development of fundamentally new lookup strategies, by introducing structure, with bounded worst case performance.

The most promising approaches for improving search on unstructured networks are based on variants of random walks [12, 9, 4]. Further, existing compression techniques such as Bloom Filters [20] can also be used in this setting to improve search quality. All these systems support attribute search (i.e., retrieving some or all items satisfying a certain predicate, for example containing a certain key-

word), not simply lookup (retrieving the item with a given identifier).

Research in the latter dimension consists of distributed hash tables (DHTs), which impose a specific topology, and can then perform extremely efficient lookups on this topology. DHTs introduce a new primitive—namespace virtualization—that is not used in unstructured networks. Virtualizing the namespace means that both node IDs and item IDs are chosen, uniformly at random, from a large set. Items are “mapped” to nodes with “similar” IDs, and the underlying topology is constructed to permit efficient routing (typically logarithmic in the number of nodes [22, 26, 24, 13]) on this virtual namespace. Namespace virtualization is not used in an unstructured network since the lookup protocol designer is not at liberty to choose the topology.

In this paper, we present a new protocol LMS (Local Minima Search), that couples the ideas of random walks and namespace virtualization to provide provably efficient lookup on unstructured topologies. Specifically, LMS has two fundamental properties:

- P1** It can provide efficient lookup under a *given-topology assumption*. Such an assumption means that an entity external to the protocol layer provides the protocol with an *underlying topology* and that there is an intrinsic high cost for two nodes to communicate directly, unless they are neighbors in the underlying topology. Trust networks are an example of a setting where the given-topology assumption holds.
- P2** It incorporates a distributed replica management protocol that is robust against a multitude of adversarial behaviors, and can dynamically adjust to topology changes. This protocol attempts to optimize lookup efficiency according to a heuristic that can be specified per-item and per-node.

Unless **P1** or **P2** are specifically needed, LMS has slightly inferior performance than a DHT. In particular, LMS routes asymptotically in $O(n^{1/(h+2)})$ steps, where h is a small constant, typically 2 or 3.

In LMS, the owner of each object places replicas of the object on several nodes. Like in a DHT, in LMS, replicas are placed 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 n is a local minimum for an object if and only if the ID of n is the closest to the item's ID in n 's neighborhood. 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.

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 network¹. In Section 4, we derive expressions for the number of necessary replicas and probes for specific probabilities of success. More importantly, we derive a technique that allows us to provide lower bounds on the performance of LMS, i.e. we show that these probabilities hold *regardless of the topology of the underlying network*.

It might seem that LMS provides the worst of both worlds: namespace virtualization does not allow sophisticated searches that can be implemented over unstructured networks, yet LMS does not achieve the lookup efficiency of a DHT. However, the use of LMS for item placement and lookup does not preclude flooding-based searches, and hence LMS provides strictly greater functionality to an unstructured network. In addition, the use of local minima in the virtualized namespace provides high-assurance random distribution of item replicas, so that a flooding search can expect to reach a replica of interest in a relatively small bounded distance. Further, index-based searches that can be implemented over DHTs can also be implemented over LMS. A goal of this paper is to demonstrate that for some applications, the inherent resilience of LMS amply compensates for its slightly lower efficiency (compared to a DHT); for these scenarios, we believe LMS will be the underlying platform of choice for building wide-area applications.

Contributions The contributions of this paper are as follows:

- We introduce the base LMS protocol, and derive expected and worst-case bounds on performance. A particularly novel feature of our analysis is that the lower bounds hold regardless of the underlying topology. We analyze the performance of LMS over realistic topologies via a set of comprehensive simulations and over a testbed with 1024 peers.
- We present three applications that can be built on the top of LMS and take advantage of its unique properties. We describe in detail one of these applications: a publishing service in an environment where some

nodes are non-cooperative, but a local notion of trust is available. We describe a model of the system and we show how LMS is more suitable than a DHT in such setting.

2 Related work

Unstructured networks have received a lot of attention due to large file sharing systems (such as Kazaa, Morphueus, and Gnutella) that have been built over them. Of these, Gnutella [6] is probably the most-studied unstructured peer-to-peer network; a lot of work has been done to understand Gnutella dynamics, for example [21] and [23]. The original Gnutella search protocol was based on naive flooding. In [10], the author discusses an improved flooding algorithm that reduces the number of messages per search, while still reaching all the nodes in the network. More sophisticated search techniques based on random walks are described in [12]. We directly compare LMS to the best protocols described in [12] in Section 5. In [4] the authors propose Gia, a system that improves on Gnutella by several techniques.

Structured peer-to-peer networks (such as Distributed Hash Tables) provide better performance than their unstructured counterparts. Chord [24], Pastry [22], Tapestry [26], CAN [18] and Viceroy [13] are some examples. Related to them is Beehive [17], a replication framework that can be built on top of a DHT. LMS uses a virtual namespace, like DHTs do, but on an unstructured topology. This has previously been done in Freenet [5] for a different reason.

In this paper, we employ Bloom filters [2] as a primitive. A Bloom filter is a digest of a set with which set membership for a certain element can be tested with a controllable false positive probability. Compressed Bloom filters, introduced in [15] improves on the size of these data structures. Even more important, for our paper, is the idea of *attenuated Bloom filters* [20]. In [20], the authors explore how to use this technique in a structured network, such as Tapestry, to provide fast lookup. In our paper, we present an application in a completely unstructured environment.

Finally, an important result in random walks that we apply for the Bloom filter analysis is due to [8]; the results say that, if the graph has sufficient expansion properties, then *for any set S* , the probability that a random walk of length l intersects S decreases exponentially with l . Theoretical properties of random walks have also been exploited by [9] to improve search in unstructured networks.

¹The probability of failure can be made arbitrarily small.

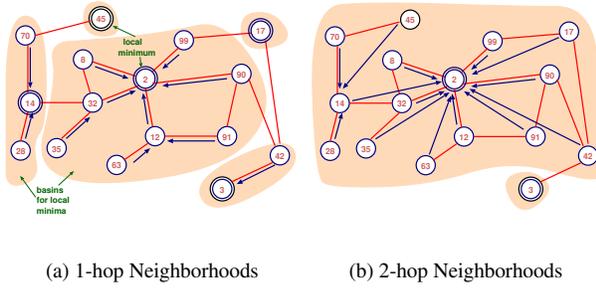


Figure 1: Local minima, basins and deterministic routing. Nodes are labelled with their distances from the key. Local minima are denoted by concentric circles, and their basins are shown as shaded regions. The path from each node towards its local minimum is shown by the arrows.

3 LMS Protocol Description

We assume a system of principals (*nodes* or *peers*) that can communicate with each other. Each node has a network address (for example an IP address and TCP port) and an ID number generated uniformly at random from an *ID space* —the set of all bit strings of some length λ , where λ is large enough to guarantee uniqueness, with high probability (for example $\lambda = 160$). The node ID is in general application-specific. We will refer to a node and its network address interchangeably, such as in messages or state data.

Topology Layer Since LMS is designed for unstructured networks, we assume that there exists a separate *topology layer*, independent of our protocol, which provides a node v with a list of *neighbors*. Each neighbor is another node whose address is known to v . Each node can communicate with any other node in the system as long as it knows the other node’s network address, but only nodes specified by the topology layer are neighbors.

We define the *neighborhood* of a node v as the set of all nodes at most h hops from v in the network. The value h is a system parameter called the *lookaround distance*, and is a small integer constant. A node will learn the identities and addresses of all nodes in its neighborhood.

Each item in the system is given an item ID (or item key), which is a random value from the ID space (such as the output of a hash function). The *distance* between two values x_1 and x_2 in the ID space is defined as $\min\{x_1 - x_2 \bmod 2^\lambda, x_2 - x_1 \bmod 2^\lambda\}$. We say that a node v is a *local minimum* for an item key x if the distance between x and the ID of v is less than the distance between x and the ID of any other node in v ’s neighborhood, as shown in Figure 1. For a large system there will be many local minima for a given key x .

Our protocol stores and searches for items by finding subsets of the local minima for a key x . Storage involves placing replicas of an item at the minima found. If the set of minima returned by a search intersects the set of minima at which replicas were placed, then the search is successful.

3.1 Routing to a Local Minimum

Consider a node v in possession of a message that must be routed to a local minimum for a key x . For v and x there will be a unique node in v ’s h -hop neighborhood minimizing the distance to x (equidistant nodes can be resolved by imposing an ordering based on their positions in v ’s neighborhood list). From node v ’s perspective, this shortest-distance node v_1 is a possible local minimum, so v forwards the message to v_1 . This process continues (v_1 forwarding to v_2 , etc.) until some node v' determines that it is the shortest-distance node to x in its neighborhood. This node v' is a local minimum for x . We define the set of nodes that will route a message with key x to v' according to this process as v' ’s *basin* for the key x , as shown in Figure 1.

3.2 Random Local Minimum Selection

Purely deterministic routing will always lead a node to the same local minimum for a given key (assuming a static topology), making it unlikely that a search will lead to the local minimum at which a replica is stored. By adding a *random walk* before the deterministic routing, we make potentially many more local minima accessible to a node, and the result of any one search or storage message is a *random* local minimum.

If a node is processing a message in the random walk phase, it forwards the message to a randomly selected neighbor. The number of random walk steps is determined by the sender, and when completed the message is forwarded according to the deterministic routing procedure of the previous section. Both replica placement and searching employ the same random-walk-followed-by-deterministic-routing procedure to find random local minima.

3.3 Replica Placement

The owner v of an item uses the random local minimum selection described above to place replicas. For each replica it needs to place, v sends out a `REPLICA-PLACE` message, which includes v ’s address and the key x of the item. The `WALK-LENGTH` and `INITIAL-WALK-LENGTH` fields control the random walk, and are both initially set to v ’s default random walk length. Upon receiving a `REPLICA-PLACE` message, a

node checks the current value of `WALK-LENGTH`. If it is greater than zero, the node decrements `WALK-LENGTH` and forwards the message to a randomly selected neighbor. Otherwise, the node performs deterministic routing to a local minimum v' for x .

If v' does not currently hold a replica, it contacts v to request a copy of the item. Otherwise, v' doubles the value of `INITIAL-WALK-LENGTH`, sets `WALK-LENGTH` to this value, and restarts the random walk phase for the message. We refer to this as *duplication avoidance*. Doubling the length of the random walk increases the probability that the message will end up outside of v' 's basin for x . An additional field, `MAX-FAILURE-COUNT`, is decremented each time duplication avoidance is invoked, and if zero the message is discarded and a failure notification returned to v .

Replicas are soft state, and when initially placed are tagged with an expiry time. The owner of an item periodically sends a `REFRESH` message to each node holding a replica of the item, which increases the time to expiry. If a replica expires, the storing node deletes it. A node receiving a `REFRESH` message for an item it does not hold (whether due to expiry or never having been stored) responds with a `REFRESH-NACK` message, allowing the owner to keep track of which replicas are no longer present so that additional replicas may be placed if needed. Changes in the topology might cause a node storing a replica no longer to be a local minimum for the item, in which case it also responds with a `REFRESH-NACK` message and deletes the replica.

3.4 Replica Lookup

Replica lookup begins the same way as replica storage, with the selection of a random local minimum. Duplication avoidance is unnecessary for replica lookups, so a `SEARCH-PROBE` message used to perform a lookup does not include the `INITIAL-WALK-LENGTH` and `MAX-FAILURE-COUNT` fields, but is otherwise identical to a `REPLICA-PLACE` message. All that is relevant is whether the selected local minimum v' holds a replica of the item, and this is conveyed in a response from v' to the searching node v . If any node visited by the `SEARCH-PROBE` message holds a replica, it can terminate the search and respond directly to v .

A single probe will often not suffice for locating an item, so a searching node v might have to send multiple `SEARCH-PROBE` messages. Occasionally a probe will reach a local minimum already searched for x . In this case v will double the initial value of `WALK-LENGTH` for subsequent probes until no further duplications occur, at which point `WALK-LENGTH` is reset to its default value. Each node v has a configurable maximum number of failures; when v has received this number of unsuccessful

probes for x it declares that the search has failed.

Probes might be sent out in parallel to decrease the average wait for a successful probe. This places a greater burden on the system, however, since more nodes will participate in the search than necessary, on average. Each node has a parameter specifying the maximum number of probes it can send out in parallel.

3.5 Determining Number of Replicas

A node has very limited knowledge of the network topology, restricted to its h -hop neighborhood. Consequently it has no *a priori* way to determine the appropriate number of replicas to place in order to maximize the efficiency of the system. While some minimum number of replicas might be placed for fault tolerance, these may be insufficient for many nodes to find the replicated item efficiently.

By increasing the number of replicas of an item stored, we can reduce the average number of search probes required to locate any of the replicas. In general, the desired number of replicas might be any function of the average number of searches,² but for simplicity of the exposition the protocol we present attempts to equalize the number of replicas and the average number of search probes needed (Section 4). The overhead of this protocol is minimal, aside from any additionally needed replica placements, as it leverages actions already performed by nodes in the system.

A node v owns an item I . A search from a random node in the system takes on average s probes to find one of the replicas. Node v periodically learns an estimate of s , as explained below, and, from the current number of reachable replicas r , 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 system parameter between 0 and 1 (we use 0.9). A larger value of α provides greater hysteresis, so that r changes more slowly and is less sensitive to fluctuations in the measured value s . We define $\delta = r' - r$ as the needed change in number of replicas. If δ is positive, δ new replicas must be placed. If δ is negative, $-\delta$ replicas are allowed to expire by ceasing `REFRESH` messages to the corresponding nodes.

The Adaptive Protocol Determining the average number of probes s requires non-local information from search probes. Since searches are a normal and frequent part of the system's information, 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 .³ When it retrieves

²For 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.

³For 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 item, v_i includes s_i in the message. If replica nodes store the actual item, they collect these probe results and periodically forward them to the item’s owner (such as in a response to a REFRESH message).

If replica nodes only store pointers to the owner of the item, retrieval requests go directly to the owner. In either case the size of the probe information in a retrieval request adds a small amount of overhead to the system. The former case adds the additional overhead of a small REFRESH acknowledgment, which occurs much less frequently than individual searches.

Learning the Neighborhood The topology layer only supplies a node with its immediate neighbors, so nodes must propagate neighbor information further in order to determine h -hop neighborhoods. This is done incrementally, with nodes first learning their 2-hop neighbors and then 3-hop and so forth, since each iteration results in more information that a node can propagate to its neighbors. Neighborhood propagation messages can incur high overhead, with IDs alone requiring 20 bytes per h -hop neighbor. For an average of d neighbors per node, this means more than $20 \cdot d^{h-1}$ bytes in overhead per node. We can reduce the cost of these messages by using abbreviated IDs (fewer bytes per node) or only propagating the latest changes in a neighborhood (fewer nodes per message).

3.6 Extension: Bloom Filters

Bloom filters can provide compact digests of the keys of items replicated by nodes. By forwarding these Bloom filters along with neighborhood updates we can improve performance, though with additional communications overhead. Note that it is possible to use an *attenuated* Bloom filter [20] to combine incoming filters and transmit only a single filter for each distance. Thus, from each neighbor v' a node receives a single filter for the items replicated at v , a single filter for all of v ’s one hop neighbors, a single filter for v ’s two hop neighbors, and so on. It is possible to further reduce the filter overhead by using arithmetic encoding to compress the filters (as described in [15]).

Use of the filters within LMS is relatively straightforward: upon receiving a SEARCH-PROBE message, a node v can check all of the Bloom filters it holds for an h -hop neighbor storing a replica of the item specified by the search. If there is a match, v can short-circuit the usual search protocol and forward the SEARCH-PROBE message directly to the matching neighbor. In general, it is not necessary to keep (attenuated) Bloom filters for the entire neighborhood for which a node keeps ID information.

the lack of search feedback.

Often it is sufficient to keep Bloom filters for a smaller neighborhood (1 or 2 hops) to get significant benefit without incurring the (computation and bandwidth) overhead of constructing and disseminating large filters from multiple hops away. In the special case that the Bloom filter and identifier neighborhood are identical, the search phase of the protocol can dispense with the directed search component of LMS, i.e. in this case, it is sufficient to perform a random walk during the search. We present an analysis of this special case, along with a general analysis of Bloom filters within LMS in Section 4.1.

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. We derive sufficient conditions for searches to succeed with high probability. 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 unsuccessful search is essentially $(1 - \frac{r}{k})^s \leq e^{-\frac{rs}{k}}$, where e is the base of the natural logarithm. 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:

Theorem 4.1 *For any connected topology G , the probability of unsuccessful search is at most $e^{-(1-o(1))rs/k}$, where the “ $o(1)$ ” term tends to zero as k gets large. In other words, the probability of failure is essentially of the form $e^{-rs/k}$.*

We next give a proof sketch for this theorem and explain some of the subtleties involved; we then extend our results to fault-prone networks and analyze the attenuated Bloom-filter technique.

Let n denote the number of nodes in G . If we conduct a random walk on G for “large enough” number $T = T(G)$ of steps, then the distribution of the last (i.e., T th) vertex visited, approaches a unique “stationary distribution” SD , no matter where we started. (This distribution 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 3.1. 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 .

Thus we arrive at the following question: let 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 in the first paragraph of this section shows that if D is the uniform distribution on K , then, the probability of unsuccessful search is at most $e^{-\frac{rs}{k}}$. In Appendix A we formally prove that this upper bound indeed holds for any distribution D .

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.) We prefer to make such a choice for r and s both for the desire to have search and placement roughly equal in complexity and similar, and also because a larger value for r or s will lead to duplication with high probability.

This shows the asymptotic performance of LMS. If the average degree of the graph is d , then $k = O(n/d^h)$. If $d = O(n^{\frac{1}{h+2}})$ (i.e. the given topology has an average degree that is not too small) then we get $r = s = O(n^{\frac{1}{h+2}})$ also. In particular, for $h = 2$, this means that the degree, number of replicas and cost of a search scale like $n^{1/4}$. This is a very good result; for networks of size less than 100,000, $n^{1/4}$ is at least as good as $\log n$.

A second quantity of interest is the number V of nodes visited by any single search probe. This, multiplied by the number s of probes, yields the total cost of a search query. We can write $V = T + l$, where T is the length of the random walk and l is the length of the deterministic routing. As we discussed above, $T = O(\log n)$ will work for practical networks. As far as l is concerned, in the Appendix B we prove the following result:

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 $1 - 2/n^c$).*

Next, what about failures? 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 (2) with some large-deviation bounds [16].

4.1 Analysis of Bloom filter extension

We now analyze the attenuated Bloom-filter technique for unstructured search. We assume that the Bloom filters are propagated within a h' -hop neighborhood of each replica.

Suppose that there are r replicas placed for item i in the graph. The placement could either be uniformly at random or using local minima — the only restriction is that the h' hop neighborhood of the replicas do not overlap too much. Let S be the union of the h' -hop neighborhoods of the nodes where the replicas are placed, i.e. each node in S either holds a replica, or holds a Bloom filter that matches item i . On average, the number of nodes in S is around $r \cdot d^{h'}$ (again assuming that the neighborhoods are almost distinct). There are also two ways of performing the search, when Bloom filters are used: we can do the

LMS search as described in Section 3, that is to say a random walk, followed by Deterministic Routing or we can just send a random walk.

Note that a search finds the item i iff it visits a node in the set S . Let’s now consider the probability of hitting a node in the set S during the random walk: for a graph with sufficient expansion, the probability that a random walk of length l visits a node in the set S is given by results due to [8] and [11]. Specifically, $\Pr[\text{search probe fails}] = \Pr[\text{walk does not intersect } S]$ is $e^{-\Omega(l \frac{|S|}{n})}$. Note that this is a powerful result: it essentially states that there is an independent probability of $e^{-\Omega(\frac{|S|}{n})} = e^{-\Omega(\frac{rdh'}{n})}$ that a node in S is not visited *for every step of the random walk*.

It is possible to show that, if the LMS lookaround distance h (Section 3) is equal to the Bloom filter depth h' , then performing the search with just a random walk of length $l + 1$ is as efficient as performing a random walk of length l followed by Deterministic Routing (which in general takes more than one step). The reason for this is that, after l steps of the random walk, if no element in S has been hit yet, performing an extra random step will lead to success with probability at least $1 - e^{-\Omega(\frac{rdh'}{n})}$. If *instead*, we performed Directed Routing towards the local minimum, the probability that we succeed would be equal to the probability that there is a replica at *that* local minimum, i.e. $r/k = r \cdot d^h/n$, which is essentially the same.

Finally, we note that if the neighborhood is propagated to a larger radius than the Bloom filters, then the LMS search augmented with Bloom filter checks is more efficient than a pure random walk.

Bloom filter size estimation. Assume that we are employing attenuated Bloom filters [20] of depth h' , that each node has I items degree d . It is easy to show that, for any $p > 0$, in order to achieve a probability of false positive of at most p/d , the length of the Bloom filter m (number of bits) should satisfy:

$$m = (-\log \frac{p}{d})(\log e) I d^{h'-1} \quad (1)$$

This ensures that the probability that a node finds a false positive in *some* neighbor’s Bloom filter is at most p .

For example, in a graph of average degree $d = 4$ with $I = 100$ items per node, if we want to achieve a false positive probability of at most $p = 10^{-5}$, we need Bloom filters with $m = 10739$ bits.

5 Experimental results

In this section we present a study of LMS. We perform extensive experiments using a simulation both to demon-

strate 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 Simulation Methodology

The message-level protocol simulator (written in Perl, about 1600 lines of code) chooses a node, uniformly at random, and simulates replica placement performed by that node. A search is performed from another similarly chosen node, which sends out search probes one at a time (Section 3.4) and records how many probes are necessary to find the item. This placement-search simulation is repeated for 10,000 trials with the same graph and key.

We present results from three kinds of graphs: *random*, *power-law*, and *Gnutella*. The random graphs have uniform edge probability between any two nodes. (In order to generate very large random sparse graphs, we generate an even larger sparse graph and take the giant component). We consider a variety of random graphs with sizes ranging from 10,000 to 100,000 nodes with different average degrees. In a power law graph of n nodes, node i ($i = 1, \dots, n$) has degree $d_i = \omega/i^\alpha$ for some positive constants ω and α . The properties of these types of graphs have recently been extensively studied in the context of the Internet [14] and of peer-to-peer networks [21]. Our power-law graphs are generated using the `inet` (version 3.0) topology generator [25], which is intended to model AS-level topology of the Internet. Specifically, we consider 10,000 node `inet` graphs. For this topology size, `inet` yields graphs with average degree 4.11. The third type of graph we use represents a snapshot generated by crawling the Gnutella peer-to-peer network. The Gnutella topology has 61,274 nodes with average degree 4.70, and also exhibits some power-law type properties [21]. Nominally, for all topologies, we assume that each peer keeps identity information for 2-hop neighbors.

For all simulation results, we note the following performance metrics: (1) success probability of the search; (2) average number of probes needed to find the item; (3) average number of nodes visited during the search. For specific simulations, we also record other information as needed (for example the distribution of the number of probes needed to locate the items).

5.2 Lookup Performance

In this basic experiment, we simulate the LMS protocol over different random graphs, power-law graphs, and the crawled Gnutella topology. As mentioned above, each row (except the Gnutella graph) is an average of 60 different graphs of the specified type, and for each instance

Graph Type	Size, avg. deg.	# Repl.	Visited LM	Visited LM+BF
power-law	10K, 4.11	6	4.8	4.3
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 1: Lookup performance for different graphs. The average number of probes is equal to the number of replicas.

we conduct 10,000 trials. For each trial, a node chosen uniformly at random places replicas for a random key. For this experiment, we set the maximum number of trials high enough such that the probability of success (for searches by nodes chosen uniformly at random as well) is at least 0.99 (See Table 1). In this result, we show the average case performance for a single replica placement: in particular, for each kind of graph, we select the number of replicas for which *the average number of probes is approximately equal to the number of replicas*. This is an interesting point in the overhead-performance tradeoff, since the number of replicas can be specified independent of network size. The cost metric is the total number of nodes visited during a search (adding up over all probes). In the Table, the “Visited (LM)” column represents the cost using only the local-minima search, while the “Visited (LM+BF)” column shows the number of nodes visited using the same number of replicas when 2-hop attenuated Bloom filters are also used to locate replicas.

There are several points of note in this basic result. First with very few replicas, we get excellent performance, compared to other unstructured protocols. Obviously, DHTs inevitably perform much better than LMS; however we are interested in a setting where a DHT is not feasible and we need the P2P protocol to offer a fast key lookup service. In particular, we chose the 10K node graphs (first two rows of the table) to be comparable to the graphs used in [12]. In this case, our results (number of visited nodes and replicas) are also comparable to the results presented in [12]. For both graphs, the best protocol in [12] is called *check* — an improved random walk protocol. The *check* protocol visits approximately 150 nodes, using 92-98 replicas on 10K node graphs. For random graphs, using only the minima search, we can achieve slightly better performance (about 130 visited nodes – see Table 1) *with only 22 replicas* (it would visit only 27 nodes if we placed 90 replicas); in the Power Law graph, we can achieve excellent performance with only 6 repli-

cas (4.8 nodes visited). Bloom filters improve the worst case performance (seen in random graphs) almost by an order of magnitude, while maintaining the same number of replicas. Both search techniques outperform naive unstructured searches such as flooding (which would incur $O(n)$ overhead) by orders of magnitude. As a comparison, a DHT like Chord achieves $\frac{1}{2} \log n$ nodes visited, which in this case translates to 7 hops.

Second, LMS is a viable protocol for lookup on large graphs. For reasonably well connected graphs (100K nodes) with degree 17 (17 is $\lceil \log_2 10^5 \rceil$), LMS needs on the order of $\log n$ replicas, and with Bloom filters, can locate items visiting only $\log n$ nodes⁴. The results that employ the Bloom filter extension are almost comparable to DHT performance on similar graphs. Obviously, DHTs only need to place a single replica, and do not need to maintain Bloom filters, but the result is still remarkable, since LMS offers this level of performance without choosing the topology. We note that for fault-tolerance, all DHT protocols advocate placing a small number of replicas.

As our results show, graphs with power law degree distributions are, in fact, a favorable case for LMS. The reason for this is that the few high degree nodes behave as “shortcuts” in the graph. In the rest of the paper, we only consider random graph topologies since LMS performance on power law graphs is strictly better.

5.3 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. 2, we present the iso-success curves for a single random graph of size 100,000 (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 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

⁴The $\log n$ number of replicas and lookup performance holds for graphs of this size, but not asymptotically. The asymptotic bound is $O(n^{1/(2+h)})$ for h hop neighborhoods.

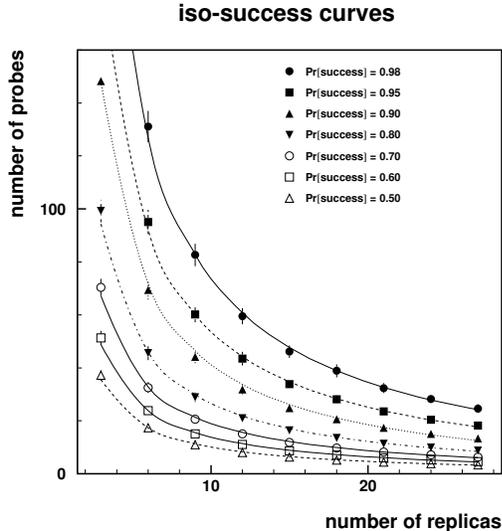


Figure 2: Number of probes as a function of the number of replicas. Random graph with 10^5 nodes, of average degree 17. The numbers labeling the curves are the particular success probabilities. The curves show fits to the functional form $f(r) = \frac{a}{r} + b$.

of the numbers of probes as the uncertainty in their average. The fits demonstrate that $rs = \text{Constant}$ for a fixed probability is a very good approximation (b is small and negative in all fits). A 2-dimensional fit to number of probes as a function of number of replicas and probability was also performed, using the functional form $F(r, P) = \frac{A \log(1-P)}{r} + B$; the fit confidence level is greater than 99%. The analytic bound in Section 4 predicts $A \approx k$, the number of local minima, in the worst case. Our observed result is very close to $A = k/2$, indicating that for random graphs roughly half of the local minima dominate the graph in terms of combined basin size. The constant B is small and negative (around -1), indicating that our probability of failure is slightly below this improved level of $e^{-2rs/k}$.

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.

Place- ment	Search Alg.	#Bloom Hops	# N'bor Hops	Avg. # Visited
rand.	rand.	2	2	55.9
LM	rand/LM	2	2	49.1
LM	LM+BF	2	3	14.9

Table 2: Evaluation of Bloom filters performance; graph with 10^5 nodes, average degree 7. All cases had 34 replicas.

5.4 Effect of Bloom filters

Our results consistently show that Bloom filters significantly reduce the overhead of LMS, and they should be used by applications whenever feasible. Our experiments show coupling Bloom filters with local minima placement improves performance beyond pure random placement. In Table 2, we consider a random graph with 100,000 nodes (average degree 7), and average the number of visited nodes for 1000 keys. The placement, and search algorithms are varied. In each case, 34 replicas are placed. (Recall that with 34 replicas on this graph, the pure LMS protocol requires 34 probes to achieve 0.99 probability of success (Table 1).) In the first row of Table 2, we note the number of nodes visited when replicas are placed uniformly at random, and searched using random walks. In the second row, we place replicas using local minima, and search using random walks (recall that this is the special case when the random walk search is equivalent to LMS search because the Bloom filter digests cover the same nodes as the neighborhood digests). From the table, we see that local minima placement improves performance by about 10% in this case. The intuition behind this improvement is simple: placing at local minima is more likely to put replicas into nodes with larger neighborhoods.

Finally, in the last row we note how performance improves if the neighborhood is propagated further than Bloom filters. This is a reasonable design choice since neighborhoods can be extremely well compressed (e.g. in this case, we need only around 13 bits per neighbor to compactly represent the three hop neighborhood such that the probability of adding an extra hop to a probe because of a mislabeled neighbor is less than 2^{-6}). In this case, using the base LMS algorithm augmented with Bloom filter checking reduces the average number of visited hops to only 14.9 (a factor of 3.7 improvement over using Bloom filters without local minima).

5.5 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

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.

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 5.6) and implementation results (Section 5.7), 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.

If f is the failure probability, the number of replicas $r(f)$ 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.

5.6 Performance under high churn

We now show that LMS can maintain reasonably high performance in presence of a very dynamic network in which nodes frequently leave and join, incurring in a limited overhead.

In this simulation, a random node u is chosen; u publishes an item and periodically runs the adaptive protocol (Section 3.5). At each time unit, a random node searches for the item and the simulator records the cost of the

Cost Ratio	Node Lifetime	Search cost	Adapt Overhead	Avg. Replicas	Prob. Succ.
2	15	44.2	69.8	16.4	0.995
	30	35.9	63.4	20.3	0.996
	60	37.2	53.2	18.6	0.996
5	15	29.6	116.6	27.4	0.995
	30	35.7	70.7	23.8	0.993
	60	32	62.1	25.8	0.994

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

search. At each time unit, q random nodes are removed from the graph, then q new nodes join the network. We simulate periodic neighborhood updates and the behavior of nodes the neighborhoods of which are out of date due to changes in the graph.

The original graph is a random graph with 10,000 nodes and average degree 7. Every time a node with degree d is deleted, a new node is added to the graph with d edges to random neighbors. This maintains the same degree distribution at any time during the simulation. We consider three different scenarios where the average time spent in the network by a node is 15, 30 and 60 minutes respectively, we assume that that neighborhood updates and adaptive protocol iterations happen every 3 minutes and the simulation time is 300 minutes. We assume $h = 2$, no Bloom filters.

Suppose node u implements the adaptive protocol, and places r replicas. u then chooses a random node in the network (using a walk of length 6), asks that node to search for the item, and uses the number of probes needed in that search as a sample for s . The adaptive protocol attempts to achieve a specific ratio for s/r (denoted by “cost ratio” in the results).

Table 4 shows the results of the experiment, which are averages of runs over 7 different random graphs. The search cost is the average number of nodes visited during a search operation. The next column is the average number of messages that the adaptive protocol causes to be sent in the network every update period. For example, for cost ratio 5, and 15 minutes average node lifetime, for each exported item, about 117 messages get sent every 3 minutes *over the whole network*. The last two columns in the results show the average number of replicas and the probability of success of the searches.

Note that in addition to the overhead due to the adaptation, every neighborhood updates causes, on average, each node to send 7 (avg. degree) messages for neighborhood updates. Each of these messages, on average contain information about 6 neighbors. In the worst case, when the entire neighborhood changes, this causes around 1700 bytes of overhead every 3 minutes (assuming 40 bytes per neighbor). However, the expected number of changes in a

two hop neighborhood in 3 minutes (assuming 15 minute lifetimes) is 16.8: 8.4 joins and 8.4 leaves. This information can be propagated using around 350 bytes (40 bytes per joining node, and 12 bits per departing node). From these results, we conclude that LMS locates items in presence of high churn with negligible overhead; we believe this is a strong result, and LMS can be deployed in highly dynamic networks.

5.7 Implementation

We have implemented the complete LMS protocol. filters), The implementation was written in C++ and employs SHA-1 hashes of node identities to obtain 160-bit identifiers. Messages are passed using ASN.1 encoding through self-serializing objects.

Multiple nodes are run on a single host, enabling us to test fairly large networks. The topology layer is implemented by having a separate topology server that creates edges between nodes at random while enforcing a minimum degree for each node. 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.

Nodes communicate through TCP/IP, maintaining connections as long as possible once established. This makes nodes fail-stop, obviating the need to periodically poll neighbors to determine if they are still alive. As nodes leave, their neighbors obtain replacements as needed to maintain the minimum degree.

We deploy the implementation on a testbed of 47 machines, each a 650MHz Pentium III running Linux. The topology server runs on a separate machine. Nodes maintain a minimum degree of 3, with an average degree of 3.9 ± 1.1 . We use two-hop neighborhoods for all tests.

In Table 5 we examine the overhead that nodes incur. For these tests we use a 1024-node network, and each node owns 64 items, for a total of 64K keys replicated throughout the system. Communications with the topology server (e.g., joins and leaves) are not included, since the topology is defined externally to our protocol, but the associated cost of propagating neighborhood changes is included. Indeed, the greatest overhead comes from the periodic updating of neighborhood information. The overhead for these updates scales as d^2 for two-hop neighborhoods, one factor of d from the size of the messages and the other factor from the number of messages that must be sent. Roughly 40 bytes of information are needed for each neighbor described, 20 bytes of which are the neighbor’s identifier and the rest includes information such as host name and port number, plus slight overhead from the ASN.1 encoding. Note that while our implementation incurs this full cost every time neighborhood infor-

mation is propagated, most of this cost can be eliminated by propagating changes only. The size of the adaptive hint messages is dominated by the 20-byte key. An item retrieval request, which we do not include, would have to include this key, so the actual overhead of reporting the number of probes required to find a replica is minimal—a small integer can be represented in ASN.1 encoding in as little as three bytes. Refresh messages are larger at nearly 70 bytes, due primarily to inclusion of the item owner’s information. This allows a node to send a negative acknowledgment to the owner if a replica has already been deleted. This overhead could be reduced by instead sending an “expired” message from the replica holder to the owner when a replica is deleted, and might prove more efficient, since such expirations are likely to be less frequent than refreshes, which must be done periodically. We expect refresh and expiry periods to be fairly long when compared with searches, however—on the same order as the neighborhood updates, so refresh overhead is not a great concern.

Figure 3 shows the effects of the adaptive protocol in a network of 512 nodes. Each node in this test owns one item to allow for more frequent updates without saturating the network. Each node periodically performs a search for a randomly selected key from the list of those stored, which is known *a priori* by all nodes. The horizontal axis shows elapsed time, while the vertical axis shows the number of replicas needed according to the protocol averaged over all 512 items. Two curves are shown, one for a single initial replica placement and the other for three initial replicas. The adaptive protocol takes longer to impact the one-replica case, since initially searches are unlikely to find the key, and item owners are not known unless a search is successful. With three initial replicas, an early successful search is much more likely, allowing the number of replicas to adapt much sooner. In both cases the number of replicas stabilizes to the same value, which is dictated by the topology.

We investigate the effect of node failures in Figure 4. For this test we use a 1024-node network with one item. The number of replicas stabilizes fairly quickly. Halfway through the test we remove half of the nodes from the network over the course of one minute. The resulting sparseness of the network causes the adaptive protocol to rapidly increase the number of replicas beyond the previous plateau, after which it stabilizes for the remainder of the test. The drop-off at the end is due to nodes leaving the network at the end of the test.

6 Applications of LMS

In this section, we describe three application built over LMS. These applications specifically benefit from the

Message type	Fixed cost (bytes)	Cost/neighbor (bytes)	Incurred
Neighborhood Propagation	11.3 ± 2.5	42.0 ± 0.4	periodically, per new neighbor
Replica Refresh	66.9 ± 0.8	—	periodically, per item owned
Adaptive Hints	28.5 ± 0.5	—	on searches

Table 5: Breakdown of overhead costs (in bytes) for the implementation. Replica refresh and neighborhood propagation costs are periodic, and adaptive hint costs are part of searching for items.

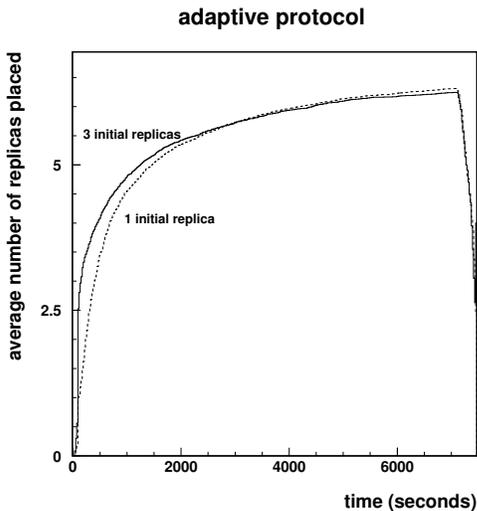


Figure 3: Effect of the adaptive protocol. The x axis shows elapsed time, with 20s between subsequent runs of the adaptive protocol. The y axis shows the average number of replicas placed for all items in the system.

properties (**P1** and **P2**) of LMS outlined in the Introduction. Due to lack of space, we will describe only the first application in detail and only briefly mention the other two.

6.1 Trust-Network Topologies

Networks defined by trust relationships are becoming increasingly important. In such a network, each node u has some notion of trust for a small number of other nodes, called a *direct trust value*. The direct trust value that u has for another node v reflects the likelihood with which u expects v to correctly discharge its application-specific obligations. If u has a high direct trust value for v (according to some metric) and v has a high direct trust value for u , then a link exists between u and v in the trust network.

The peer-to-peer application we consider in this scenario is one in which nodes publish and retrieve items with well-known identifiers. Some fraction of the nodes might be compromised by an attacker and deviate from the protocols, affecting both the quality of service and the correctness of the system.

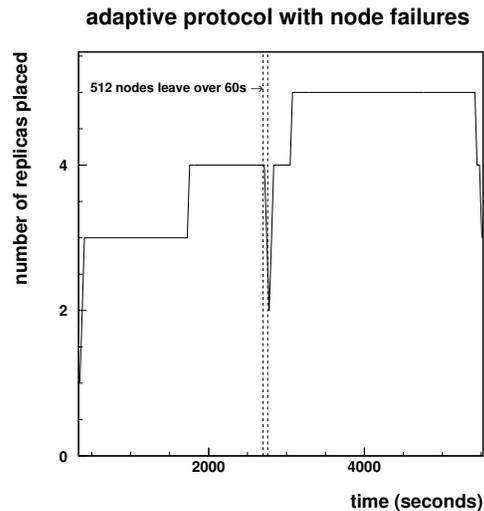


Figure 4: Effect of node failures on the adaptive protocol. The x -axis shows time, with 30s between subsequent runs of the adaptive protocol. The y -axis shows the number of replicas placed. Half the nodes leave the system at 2700 seconds. The drop-off on the far right is from system shutdown.

In this section, we compare the performance of Pastry [22] and LMS for such an application, and show that while Pastry is capable of adapting to the underlying topology, the ability of LMS to entirely conform to the specified topology yields better performance. We also sketch a modification of Pastry that uses an LMS idea and can achieve excellent performance in this setting. However significant work is required to make this modification practical.

Studying behavior of those protocols in such hostile environment is a very complex question that we do not address here in its entirety. Instead we focus on a simplified model and we leave further investigation for further work. It is our intuition, however, that our results are valid in a much more general setting. Specifically, we make the following assumptions:

1. Each node has a public-private key pair for a digital signature scheme.
2. The content of an item is digitally signed by its

owner.

- Each item unequivocally belongs to an owner. More precisely, given the key of an item it is possible to unequivocally determine the public key of the owner of the item. This can be obtained, for example, by defining the form of an item key as “owner-public-key/item-descriptive-string”. This is the same concept as *signed-subspace keys* in Freenet [5].

The use of digital signatures precludes an attacker generating false data on behalf of any node which it does not control. Rather than considering arbitrary messages generated by compromised nodes, we limit our experiments to the case where compromised nodes might refuse to process messages. More sophisticated attack models are outside the scope of this paper.

The simulation In each iteration of the simulation a random node publishes an item with a random ID and then another random node performs a random search for the same item; the simulator records whether the search returns the item or fails. The network topology is generated as an undirected random graph with 1000 nodes with average degree 6. For simplicity, we assign a direct trust value of 0.5 to each edge of the graph.

For each pair of nodes (u, v) , the simulator computes the *indirect trust value* of u into v , which is computed as a function of the direct trust values on the edges of the graph. We considered more than one definition of indirect trust value and we show results for the one that, we believe, better represents the concept of trust in a social network [1]. It is known by now that the interconnection patterns in a wide variety of engineered and social/biological networks do not directly conform to well-studied random models[1]. This motivates us to study general topologies – those that are not structured a priori in some particular way. Also, since such networks typically exhibit high local clustering[1], we have come up with a novel method of evaluating trust. Our metric is more refined than simple metrics that only look at shortest paths, and takes into account the fact that a large number of high-fidelity paths between two nodes, usually connotes a trustful relationship between these nodes. Specifically, we define the indirect trust value of u into v as the probability that u and v remain connected in a random experiment where each edge (u', v') of the graph is retained with probability equal to the direct trust of the edge and deleted otherwise.

The simulator then flips a biased coin and marks the pair “good” with probability equal to the indirect trust and “bad” otherwise. If the pair is bad, then v will systematically ignore any messages generated by u ; if the pair is good, then v will process such messages correctly.

Protocol	Replicas	Success Pr.	Visited
Pastry	1	0.436	2.02
Modified Pastry	6	0.923	4.7
	8	0.956	4.9
	10	0.972	5
	12	0.98	5.14
LMS	6	0.928	35.1
	8	0.944	26.9
	10	0.95	23.2
	12	0.955	20.2

Table 6: Probability of success and number of nodes visited by Pastry and LMS on a trust network.

We built the Pastry simulator (approx. 1650 lines of code) on top of the `FreePastry` [7] library. The proximity of two nodes is defined to be the indirect trust value, as defined above. We also adapted the LMS simulator for this experiment. Similarly to the previous experiments, h was set to 2.

Table 6 shows the results of the simulation. Each experiment is the average of runs over 7 different random graphs with 10,000 iterations on each graph.

Note that Pastry can achieve a probability of success of only 43.6% under the attack of untrusted nodes. LMS can instead achieve variable levels of success probability, depending on the setting of the number of replicas and the maximum number of probes. In particular, we see how LMS can be parametrized to achieve a much higher success probability with a reasonably small number of visited nodes.

While it is possible to configure Pastry to create more than one replica, the base protocol would create these replicas in the leaf set of the root node of the item [22]. Messages routed from a node u to different replicas will therefore follow approximately the same path, so a “bad” node v along the path to the root node will likely be on the path to all of the replicas. Instead, we consider a *Modified Pastry* protocol⁵, that creates multiple replicas for the same item, assigns to each replica an *independent random id* and routes them independently. This can be done, for example, by making the id of the i -th replica of an object with identifier x to be the hash of x concatenated with i . When a node searches for x , it will search sequentially for all the replicas of x until a replica is found. We note that such a protocol performs very well in this setting (see Table 6). However, more work is needed to construct a practical protocol which can determine how many replicas to place during publishing and convey the number of replicas available during searching.

⁵Note that our modified secure Pastry protocol is significantly more efficient than the secure routing protocol [3], which was developed under a slightly different model, and requires trusted key generation

We conclude that the property **P1** of LMS to match its overlay topology with an underlying topology, makes it an attractive substrate for this kind of application. Finally, it is possible to apply some of the LMS ideas to a DHT protocol, but this requires further refinement.

6.2 Other applications over LMS

A number of other applications can also benefit from the properties of LMS, and we have experimented with a few.

Paranoia is a censorship-resistant publishing system, and uses data replication to prevent deletion. The appeal of LMS for this application is that it employs no distributed state, so attacks against nodes do not cause massive updates, which could otherwise lead to a denial of service. In addition, the lack of predefined structure in LMS allows nodes to join the network according to their own policies and metrics, such as only connecting to other nodes in which they have some level of trust or maintaining a certain minimum degree. The base LMS protocol is not suitable for censorship-resistant publishing; slight modifications have to be made to accommodate anonymity. Item owners and retrievers are masked by probabilistically launching additional searches from nodes along the lookup path, providing deniability more than true anonymity (similar to [19]). Since the adaptive protocol opportunistically uses information collected during routine searches, but the actual owner is not known in *Paranoia*, the owner is forced to initiate its own searches in order to collect samples for replica adaptation. This is similar to the case in normal LMS for adapting the number of replicas for an unpopular item (Section 3.5). Here the property **P1** of LMS is particularly valuable, since no requirement is made on a node's neighbor set.

SDS is a peer-to-peer distributed striped file storage system. It uses erasure codes rather than block replication to more efficiently and reliably store files. The drawback is that partial files cannot be retrieved, since all stripes of a file must be used to recover any piece of the file. *SDS* is designed to work on top of either LMS or a DHT. The primary advantage of using LMS instead of a DHT in *SDS* is in load balancing (property **P2**), since LMS distributes stripe replicas randomly among the possible local minima, while DHTs require some additional mechanism such as multiple hashings of a file name.

7 Conclusions and Future Work

In this paper, we have introduced a new technique (LMS) for searching in unstructured networks. By using a virtualized namespace, LMS is able to distribute items broadly through the network, achieving good load balancing. For known-key lookups, LMS provides a great improvement

over existing algorithms that operate on unstructured networks without interfering with the effectiveness of other algorithms run on the same networks. While distributed hash tables provide more efficient key lookup, they place strong constraints on the peer-to-peer networks on which they are constructed. The resilience of LMS is also considerably better than the base protocols for common DHTs, with much work needed to improve DHT resilience to the level possessed by the base LMS protocol.

Our analysis derived tight bounds for worst-case performance of LMS. These bounds, while not nearly as efficient as the performance typical of DHTs, demonstrate that LMS can run with acceptable efficiency on any network, regardless of topology. In many cases, the performance of LMS is considerably better than the worst-case bounds, and is in some circumstances comparable to that of a DHT. We also derived strong lower bounds on performance for the specific case where the number of probes in a search for an item is equal to the number of replicas of that item.

To emphasize the advantages of LMS, we presented three applications that benefit greatly from the properties that LMS possesses. We investigate one of these, that of a network topology defined by mutual trust relations, in detail. This network is characterized by nodes dropping messages sent by untrusted nodes, and even at a highly adversarial level of distrust LMS was able to perform with a probability of success not very different from that in a benign network. Pastry, by comparison, performed relatively poorly in this environment, and only equaled the performance of LMS when given advantages not currently possible in an actual deployment.

References

- [1] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 27 May 2000.
- [2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [3] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.
- [4] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott 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.
- [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] Open Source Community. Gnutella, 2001. See <http://gnutella.wego.com>.
- [7] Peter Druschel et al. FreePastry: A modular, open source implementation of the pastry p2p routing and location substrate. <http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry/>, July 2003.
- [8] David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27(4):1203–1220, 1998.
- [9] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks. In *IEEE Infocom*, 2004.
- [10] Song Jiang, Lei Guo, and Xiaodong Zhang. LightFlood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. In *International Conference on Parallel Processing*, 2003.
- [11] N. Kahale. Large deviation bounds for Markov chains. Technical Report 94-39, DIMACS, 1994.
- [12] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott 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.
- [13] Dahlia Malkhi, Moni Naor, and David 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.
- [14] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in Internet topologies. *ACM SIGCOMM Computer Communication Review*, 30(2):18–28, 2000.
- [15] M. Mitzenmacher. Compressed bloom filters. In *20th Annual ACM Symposium on Principles of Distributed Computing*, pages 144–150, 2001.
- [16] 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.
- [17] Venugopalan Ramasubramanian and Emin Gun Sirer. Beehive: Exploiting power law query distributions for O(1) lookup performance in peer to peer overlays. In *Proceedings of NSDI*, 2004.
- [18] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, 2001.
- [19] Michael Reiter and Aviel Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.
- [20] Sean C. Rhea and John Kubiatowicz. Probabilistic location and routing. In *Proceedings of INFOCOM 2002*, 2002.
- [21] M. Ripeanu and I. Foster. Mapping the Gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.
- [22] Antony Rowstron and Peter 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)*, 2001.
- [23] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Syst.*, 9(2):170–184, 2003.
- [24] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.
- [25] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator. Technical Report UM-CSE-TR-456-02, University of Michigan, 2002.
- [26] 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.

A Proof of Theorem 4.1

We now show that the bound of $e^{-\frac{rs}{k}}$ holds when $r = s$, no matter what distribution D is; there are similar results for other choices of r and s . 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. Number the local minima for x as $1, 2, \dots, k$, and let X_i be the event that i lies in both R and S . As usual, \bar{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 $\Pr[\bar{X}_i] = (q_i^r + q_i^s - q_i^{r+s})$.

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

$$\begin{aligned} \Pr\left[\bigwedge_{i=1}^k \bar{X}_i\right] &\leq \prod_{i=1}^k \Pr[\bar{X}_i] \\ &= \prod_{i=1}^k (q_i^r + q_i^s - q_i^{r+s}). \end{aligned} \quad (2)$$

Now, assuming $r = s$ and $f(z) = 2z^r - z^{2r}$, we have

$$\ln \Pr\left[\bigwedge_{i=1}^k \bar{X}_i\right] \leq \sum_{i=1}^k \ln(f(q_i)). \quad (3)$$

Note that $\sum_i q_i = k - \sum_i p_i = k - 1$. Now, by considering the second derivative, it can be shown that in the domain $z \in [0, 1]$, the function $z \mapsto \ln(f(z))$ is *concave*. Thus, subject to the constraint $\sum_i q_i = k - 1$, the value $\sum_{i=1}^n \ln(f(q_i))$ is maximized when all the q_i are equal (i.e., equal to $1 - 1/k$). This, in combination with (3) and some further calculation, leads to the bound stated in Theorem 4.1.

B Proof of Theorem 4.2

Fix an arbitrary object x . We aim to show that the “deterministic routing to a local minimum for x ” take at most $O(\log n)$ steps⁶ in expectation, and also with high probability.

Recall that we currently hash to 160-bit IDs, and also recall our notion of distance between two hash values. Since the ID-space has size large enough (2^{160}), the situation is essentially equivalent to the following. Let C be a circle of unit circumference. We then hash each entity v (whether it is an object or a node) to a random point $h(v)$ on the circumference of C . The distance between two points p and q that lie on C , denoted $D(p, q)$, is the length of the shorter of the two arcs that connect them; thus, $D(p, q) \leq 1/2$. Note that this is the same notion of distance that we currently employ; thus, our notion of local minima etc. carries over here exactly. (That is, we always aim to find a neighbor who has the smallest distance $D(\cdot, \cdot)$ to the value $h(x)$.) The utility of mapping on to the circle is that the analysis becomes smoother (e.g., via integrals).

Suppose we start at a node u_0 , and are routing to a local minimum for x . We assume without loss of generality that the hash value $h(x)$ of x , is the lowest point P on C . Whenever we say “distance”, we will mean the distance function $D(\cdot, \cdot)$. We now introduce some random variables. Let u_1, u_2, \dots be the successive nodes visited; if u_i is the local minimum found, then u_{i+1}, u_{i+2} etc. equal u_i . Let $X_i = D(h(u_i), P)$ denote the distance “between u_i and x ”; note that the sequence X_0, X_1, \dots decreases until we hit a local minimum. Our key idea is to show that this sequence decreases fast enough. For $t \geq 0$, let A_t be the indicator random variable for the event that the walk has **not** yet stopped after t steps (i.e., after visiting u_0, u_1, \dots, u_t).

Our key lemma is the following:

Lemma B.1 *For any $t \geq 1$ and any $z \in [0, 1/2]$, $\mathbf{E}[X_{t+1}A_t \mid X_t A_{t-1} = z] \leq z/2$.*

We will prove Lemma B.1 below; let us now see why the lemma yields the desired $O(\log n)$ bound. Note that $\mathbf{E}[X_1 A_0] \leq \mathbf{E}[X_1] \leq 1/2$; thus, Lemma B.1 and an induction on t yield that $\mathbf{E}[X_t A_{t-1}] \leq 2^{-t}$. In particular, letting $T = (c + 1) \log n$ where c is some suitable constant, we get

$$\mathbf{E}[X_T A_{T-1}] \leq n^{-(c+1)}. \quad (4)$$

Now, suppose $u_T = u$ and that $X_T A_{T-1} = z$. Node u has at most n unexplored neighbors; for each of these neighbors v , $\Pr[h(v) < z] = 2z$. (The factor of two comes from the fact that $h(v)$ can fall on either side of point P on the circle.) Thus, the

⁶If the maximum degree is Δ , I think we can further strengthen this to $O(\log \Delta + \log \log n)$ –Aravind

probability that u is **not** a local minimum is at most $2nz$; more formally,

$$\forall z, \Pr[A_T = 1 \mid X_T A_{T-1} = z] \leq 2nz.$$

So,

$$\Pr[A_T = 1] \leq 2n \cdot \mathbf{E}[X_T A_{T-1}] \leq 2/n^c$$

by (4), which is negligible if, say, $c \geq 2$. Thus, the routing takes at most $O(\log n)$ steps in expectation, and also with high probability.

We now prove Lemma B.1:

Proof: First of all, we can assume that $z \neq 0$; since $X_{t+1} \leq X_t$ and $A_t \leq A_{t-1}$, the lemma is trivial if $z = 0$. Therefore, we have $A_{t-1} = 1$ and $X_t = z$; i.e., the walk has not stopped after visiting node u_{t-1} , and also $D(h(u_t), P) = z > 0$. Let Y be the random variable denoting the number of “unexplored” neighbors of u_t ; i.e., the number of neighbors of u_t that do not belong to the set $\{u_0, u_1, \dots, u_{t-1}\}$. If $Y = 0$, then u_t is a local minimum, and hence $X_{t+1} A_t = 0$. We will now prove:

$$\forall d \geq 1, \mathbf{E}[X_{t+1} A_t \mid ((X_t A_{t-1} = z) \wedge (Y = d))] \leq z/2. \quad (5)$$

If we can do so, we will be done, since if the lemma holds conditional on all positive values of d , it also holds unconditionally. For notational simplicity, we will from now on refer to the l.h.s. of (5) as Φ .

Fix some $d \geq 1$; in all arguments below, we are conditioning on the event “ $(X_t A_{t-1} = z) \wedge (Y = d)$ ”. Let v_1, v_2, \dots, v_d denote the d unexplored neighbors of u_t . If $h(v_i) > h(u_t)$ for all i , then u_t is a local minimum, and hence $X_{t+1} A_t = 0$. Therefore, conditioning on the value $y = \min_i d(h(v_i), P) \leq z$, and also considering the d possible values of i that achieve this minimum, we get

$$\Phi = 2d \cdot \int_{y=0}^z y(1-2y)^{d-1} dy.$$

Again, the factor of two up-front comes from the fact that the “minimizing neighbor” v_i can fall on either side of point P on the circle. A simple computation yields

$$\Phi = \frac{1 - (1 - 2z)^d \cdot (1 + 2zd)}{2(d + 1)}. \quad (6)$$

We need to show that the r.h.s. of (6) is at most $z/2$. Changing variables to $y := 2z$ and rearranging, we need to show that

$$f(y) \doteq (d + 1)y/2 + (1 - y)^d(1 + yd) \geq 1,$$

where $0 \leq y \leq 1$ and $d \geq 1$ is an integer. This inequality is easily seen to hold if $d = 1$, so we may assume $d \geq 2$.

It can be verified that

$$\begin{aligned} f'(y) &= (d + 1)/2 + d(1 - y)^d - d(1 + yd)(1 - y)^{d-1}; \\ f'(1/d) &= (d + 1) \cdot (1/2 - (1 - 1/d)^{d-1}); \\ f''(y) &= d(1 - y)^{d-2}(d + 1) \cdot (dy - 1). \end{aligned} \quad (7) \quad (8)$$

We see from (8) that f' has a unique minimum (in our domain $0 \leq y \leq 1$) at $y = 1/d$. Also, for integer $d \geq 2$, the function $(1 - 1/d)^{d-1}$ has value $1/2$ when $d = 2$, and decreases as d takes on higher integral values. Thus, (7) shows that $f'(1/d) \geq 0$. Since this minimum value is non-negative, it follows that $f'(y) \geq 0$ for $0 \leq y \leq 1$. So, since $f(0) = 1$, we get $f(y) \geq 1$ for all $y \in [0, 1]$, as required. ■