2004

Efficient Randomized Search Algorithms in Unstructured Peer-to-Peer Networks

Ronaldo A. Ferreira

Murali Krisna Ramanathan

Ananth Y. Grama
Purdue University, ayg@cs.purdue.edu

Suresh Jagannathan
Purdue University, suresh@cs.purdue.edu

Report Number:
04-022

EFFICIENT RANDOMIZED SEARCH ALGORITHMS IN UNSTRUCTURED PEER-TO-PEER NETWORKS

Ronaldo A. Ferreira
Murali Krishna Ramanathan
Ananth Grama
Suresh Jagannathan

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #04-022
July 2004

Efficient Randomized Search Algorithms in Unstructured Peer-to-Peer Networks

Ronaldo A. Ferreira Murali Krishna Ramanathan Ananth Grama Suresh Jagannathan

Department of Computer Sciences – Purdue University
West Lafayette, IN, 47907, USA
{rf, rmk, ayg, suresh}@cs.purdue.edu

Abstract—**Searching for objects in unstructured peer-to-peer (P2P) networks is an important problem, and one that has received recent attention. In this paper, we present a simple, elegant, yet highly effective technique for object location (including rare objects). Our scheme installs object references at a known number of randomly selected peers. A query to the object is routed to a predetermined number of random peers, selected independently of the installation procedure. The high probability of a non-empty intersection between these two sets forms the basis for our search mechanism. We prove analytically, and demonstrate experimentally, that our scheme provides high probabilistic guarantees of success, while incurring minimal overhead.**

Effective realization of the approach builds on a number of recent results on generating random walks, and efficiently estimating network size for unstructured networks. The presence of failures (departures) in the network pose additional challenges. Finally, effective strategies for installing references to replicas are critical for optimizing the tradeoff between installation overhead and search time. We address these issues in an analytical framework and validate our results on a variety of real and synthetic topologies. Our results generalize to related problems of estimating and controlling object replication, and eliminating duplicates in large-scale unstructured networks.

Index Terms—**System design, Simulations, Mathematical optimization, Statistics**

I. Introduction

Search is a fundamental service in peer-to-peer (P2P) networks, one that has received considerable research attention [1], [2], [3], [4], [5]. In contrast to structured networks, search in unstructured networks is considerably more challenging because of the lack of global routing guarantees provided by the overlay. In spite of this apparent disadvantage, unstructured P2P networks have several desirable properties not easily achieved by their structured counterparts – they support inherent heterogeneity of peers, are highly resilient to peer failures, and incur low overhead at peer arrivals and departures. These characteristics make unstructured networks attractive to users who cannot commit their resources for sustained periods of time. The attractiveness of these features is reflected in the popularity of such networks in practice. Gnutella networks, like Limewire [6], for example, have peak populations in the order of millions of users.

In typical unstructured P2P networks, such as Gnutella [7], a peer searches by flooding a (hop) limited neighborhood. This method, though simple, does not provide any guarantee that an object existing in the network will be found. Moreover, flooding does not scale well in terms of message overhead, since each query may generate a significant amount of traffic. Several recent studies have addressed these completeness and scalability issues. In [8], Cohen *et al.* improve the efficiency of search in unstructured P2P networks by replication. They evaluate two different replication strategies, uniform and proportional, and conclude that an optimal replication strategy lies between the two schemes. Their replication scheme is based on access frequencies of objects. The focus of our work is on locating any object (independent of its access frequency) in an efficient manner. In this sense it complements the work of Cohen and Shenker, and other replication and caching mechanisms. In [1], Lv *et al.* show that search using random walk can reduce network traffic by up to two orders of magnitude when compared with flooding-based techniques. Yang and Garcia-Molina [9] also present several strategies for reducing the overhead of search in unstructured P2P networks. In one approach, called iterative deepening, a node searches the network by querying all its direct neighbors (flooding); if the query is not resolved, the depth of the flooding is incremented. This technique is related to the expand-ring algorithm proposed in [1]. These schemes aim to provide search success guarantees, while limiting overall messaging overhead.

The objective of providing search guarantees, while simultaneously minimizing messaging overhead, is important, and has been solved elegantly for structured

P2P networks. In these environments [3], [4], [5], nodes participate in more involved protocols for joining and leaving the network; these protocols guarantee that a well defined topology is always preserved. A single primitive is provided to search for an object: given the object name (or hash code), return the IP addresses of nodes that currently have a copy of the object. Using a a distributed hash table (DHT) abstraction these techniques typically provide an upper bound of $O(\log n)$ on overlay hop-count for queries, where n is the total number of nodes in the network. This upper bound is achieved using $O(\log n)$ routing information per node. Unfortunately, as mentioned above, even though DHT systems provide deterministic location services, they may incur significant overheads for maintaining the network, and cannot exploit node heterogeneity. For this reason, they are less suited to environments with frequent node arrivals and departures.

In this paper, we present a simple, elegant, and highly effective technique for object location based on a variant of the birthday paradox. Our scheme, installs object references at a known number ($O(\gamma\sqrt{n})$) of randomly selected peers for small γ. By choosing γ to be $\sqrt{\ln n}$, we show that any search will succeed with high probability (w.h.p.[1]).

A query to the object is routed to $O(\gamma\sqrt{n})$ random peers, selected independently of the installation procedure. The high probability of a non-empty intersection between these two sets forms the basis for our search mechanism. Specifically, we prove analytically, and demonstrate experimentally, that our scheme provides high probabilistic guarantees of success, while incurring minimal overhead, even for objects with very low popularity (replication). Our experiments also reveal that even for very small γ, the likelihood of a query failing is negligible.

While the intuition underlying our search protocol is easily stated, effective realization of the approach builds on a number of recent results on generating random walks, and efficiently estimating network size for unstructured networks. Gkantsidis *et al.* [2] show that it is possible to choose k peers randomly in an unstructured network by first performing a random walk of length $O(\log n)$[2] and then proceeding with the random walk for k more hops. The last k peers encountered in the random walk represent a uniform sample of the network. A number of researchers have addressed the problem of estimating network size [10], [11], [12]. In [10], for example, Horowitz *et al.* present an estimation scheme that allows a peer to estimate the size of its network based only on local information with low overhead by maintaining a logical ring. Mayank *et al.* [11] propose an estimation scheme based on the birthday paradox [13][page 45]. The paradox is that for $\sqrt{n}$ independent samples from a population of size n, the probability that two samples have the same value is at least $1/2$. A peer estimates the network size by sending a message on a random walk and using the hop count when the message returns to the peer. It is shown that it takes approximately $\sqrt{n}$ hops for a message to return to its sender. Psaltoulis *et al.* [12] propose a network size estimate algorithm for large and dynamic networks by using sampling techniques.

The presence of node failures or departures (churning) in the network pose additional challenges to defining a robust search protocol. To account for this, we derive analytic expressions for augmented reference installations as well as refresh procedures. Effective strategies for installing references to replicas are critical for optimizing the tradeoff between installation overhead and search time. If reference pointers are installed by each replica, the associated overhead is likely to be high for popular items while yielding limited benefit in terms of search for less popular objects. Conversely, if none of the replicas install reference pointers, peers will not be able to find nearby copies of replicated items. To address this, we present a novel approach to controlled reference installation, which we demonstrate strikes a desirable balance between reference installation and search overheads. We address these issues using an analytical framework, and validate our results on a variety of real (Gnutella) and synthetic topologies. In [14], Gummadi *et al.* perform a detailed measurement study of p2p networks. We use the results on failure characteristics of the peers to evaluate our search mechanism, when peers fail or depart in the system.

The techniques presented in this paper generalize to a number of related problems. These include, estimating and controlling the replication factor of objects, and duplicate elimination of object copies in large unstructured networks. Each of these problems can be accomplished with limited global knowledge and with a message complexity of $O(\sqrt{n})$.

A. Technical Contributions

We summarize the main contributions of this paper as follows:

[1]Throughout this paper, w.h.p. (with high probability) denotes probability $1 - \frac{1}{n^{O(1)}}$.

[2]The exact length of the random walk involves the second eigenvalue (λ_2) of the network graph. It is shown in [2] that good global connectivity translates into constant λ_2, which is the case for P2P networks.

1) A novel search protocol to locate any object (including rare objects) in unstructured P2P networks that provides high probabilistic guarantees, while incurring minimal messaging overhead.
2) Analytic and experimental evaluation of augmented reference installation and refresh protocols to account for node failures.
3) Techniques for controlled replication, that balance overheads of replication and search.
4) Extensions of the algorithm to related problems in estimating replication factor of objects, and duplicate elimination.

II. Placement and Search Protocols

As mentioned, the proposed query mechanism relies on installing a known number of references to each object randomly across the network. The corresponding search algorithm queries an independently chosen random set of peers. The intersection of these two sets results in a successful search. We describe each of these steps in detail.

A. Installing References to Objects

To share its content with other peers in the network, a peer must install references to each of its objects at other peers. This is similar to the process of publishing content in structured peer-to-peer networks. The main difference is that in our approach, a peer p can publish an object O by installing references to O at *any random set* Γ_p of peers selected independently of O's identifier (or hash). There are three kinds of overheads involved due to this mechanism.

- Memory overhead: Since we install references (and not replicas of actual objects, since we are concerned with locating the object), the memory overhead is not significant.
- Processing overhead: If every peer in the network installs references at $O(\sqrt{n})$ ($\Gamma_p = O(\sqrt{n})$) other peers for an object, on an average $O(\sqrt{n})$ references are installed at any peer. When a search message is received, the peer can perform the search in $O(\log n)$ time and when an installation message is received, it can insert the reference in $O(\log n)$ time.
- Communication overhead: The overhead of messaging and keeping information consistent is the most important one. It includes overhead for installing references and searching for objects. Trivially, if we do not install any references, search overhead is $O(n)$ and if we install $O(n)$ references, search is local. In both cases, the communication overhead is $O(n)$ to locate any object in the network. Here, we show an asymptotically superior scheme which involves $O(\sqrt{n})$ overhead for installing references, and $O(\sqrt{n})$ overhead for locating objects.

An object can be published using any form of identification, for example, a file name or meta information about the object. This is in contrast to structured peer-to-peer networks, where the object and the node identifiers where it resides are tightly coupled. To provide guarantees that content published by a peer p can be found by any other peer in the network, three fundamental questions need to be answered:

1) Where should the nodes in Γ_p be located in the network?
2) What is the size of the set Γ_p?
3) When a peer q attempts to locate an object, how many peers must q contact?

The answer to the first question, in our framework, is extremely simple – we select Γ_p in such a way that any node in the network has an equal probability of belonging to the set. This selection criteria is important for two reasons – it provides a measure of fault tolerance, i.e., in the event of a node failure, the node can be easily replaced. Secondly, as we shall see, in an unstructured network, this facilitates search, as there is no consistent global routing infrastructure. Since conventional P2P networks are expected to scale to millions of peers and nodes only connect to a small set of other nodes, creating a uniformly sampled set in the network is not a straightforward task. To accomplish this, we rely on the recent work of Gkantsidis *et al.* [2], who show that it is possible to construct a uniform sample of size k in an unstructured network by performing a random walk of length $\Omega(\log n)$ and then proceeding with the random walk for k more hops. The last k peers encountered in the walk represent a uniform random sample.

The second and third questions above can be answered using the birthday paradox as described above. This result has been used in various distributed algorithms [15], [11]. For example, to define a probabilistic quorum, quorum members are chosen uniformly at random from all members in the network, and the quorums have size $\gamma\sqrt{n}$. In [15], Malkhi *et al.* show that two quorums chosen in this manner intersect with probability at least $1-e^{-\gamma^2}$. However, this result is presented in the context of distributed systems in which nodes have complete knowledge of all other nodes in the system, and in which the population of nodes is stable.

Using the results above, the protocol used by a peer to publish its content follows naturally. To guarantee search success w.h.p., we set $\gamma = \ln n$. In this case,

Join of p:

- Connect to K peers already in the network, where K is a constant chosen independently by p.
- Estimate the network size and assign it to n.
- Create a message M as follows:
  - M.TTL = $\log n + \sqrt{n}$
  - M.SSIZE = $\sqrt{n}$
  - M.type = SEARCH
  - M.sender = $(IPaddr_p, PortNo_p)$
  - M.data = Metadata(O)
- Send M to one of its K neighbors selected uniformly at random.
- If M is answered by a peer at distance l, with probability $\frac{l}{\sqrt{n}}$ execute the following:
  - Create a message M′ as follows:
    - M′.TTL = $\log n + \gamma\sqrt{n}$
    - M′.SSIZE = $\gamma\sqrt{n}$
    - M′.type = INSTALL
    - M′.sender = $(IPaddr_p, PortNo_p)$
    - M′.data = Metadata(O)
  - Send M′ to one of its K neighbors selected uniformly at random.

Fig. 1. Join algorithm.

we achieve an analytic probability of success to be $1 - \frac{1}{n^{\theta(1)}}$. In our experiments, we demonstrate that for typical overlay topologies, even setting γ to a small constant (e.g., 2) results in successful search for all queries, including queries for unpopular objects. Upon joining the network, a peer p performs a random walk of length $\gamma\sqrt{n} + \log n$. In the random walk message, p includes information about its content and asks the peers along the random walk to install references to its content. Peers that are reached by the random walk send back to p their identification (IP addresses). Node p uses these peers as its set Γ_p. Figure 1 illustrates the algorithm a peer executes on joining the network. In the algorithm, M.SSIZE specifies the size of the set Γ_p and is used by peers along the random walk to determine if the content ($Content_p$) present in the message must be installed or not. The first $\log n$ peers in the random walk do not install the content, they simply forward the message.

Theorem 1: If f is the percent of nodes leaving the network in a given time period, $f * |\Gamma_p|$ new references need to installed after the time period.

Proof: Since peers in the network leave frequently, a peer p must ensure that approximately $\gamma\sqrt{n}$ peers ($1 \leq \gamma < \ln n$) in its set Γ_p are still present in the network to guarantee the high probability of successful searches. As peers in Γ_p are selected uniformly at random from the network, a node can be easily replaced by selecting uniformly, at random, a new node from the network. If a percentage f of nodes leave the network, it is expected that approximately f percent of the nodes from Γ_p also leave the network. Therefore, p must periodically select $f * |\Gamma_p|$ new peers from the network to replace nodes in Γ_p that might have left the network. ■

In [14], Gummadi *et al.* show that in the Gnutella network 50% of the nodes leave the system every hour. If we assume the same departure model, a peer p sharing an object must add $\frac{\gamma\sqrt{n}}{2}$ peers hourly to Γ_p.

B. Controlled Installation of Object References

The question of whether the replica of an object installs a different set of reference pointers is an important one. If none of the replicas install reference pointers, peers would be routed to possibly distant copies of objects, even though replicas might exist on nearby nodes. Conversely, if every replica inserts reference pointers, popular objects may attempt to install reference pointers on all peers. Neither of these extremes is desirable. To avoid this situation, we use a probabilistic algorithm to determine if a node should install reference pointers to its objects. When a peer p joins the network, it sends a query for an object using a random walk of length $\sqrt{n}$. If the query is unsuccessful, then p installs the pointers with probability one. If the query is successful and the responding peer q is at a distance l from p (where distance is the number of hops the random walk traversed), then p installs the pointers with probability $\frac{l}{\sqrt{n}}$. We show that this algorithm significantly reduces the number of reference pointers to popular objects without significantly impacting the performance of the search protocol.

C. Object Search

A peer p searches an object in the network by performing a random walk. The random walk message contains the query information and a time-to-live (TTL) field set to $\gamma\sqrt{n}+\log n$. Nodes along the random walk process the query by searching their local content and the reference pointers installed by other peers. If a peer can respond to a query, it sends the response directly to the querying peer and stops the random walk, i.e., it does not forward the random walk message. If a peer cannot respond to a query, it decrements the TTL field by one and, if the resulting value is greater than zero, forwards the message. If the TTL value reaches zero, the message is not forwarded. Figure 2 presents the algorithm that needs

Search: To search for an object O, peer p executes the steps below:

- Create a message M as follows:
  - M.TTL = $\log n + \gamma\sqrt{n}$
  - M.type = SEARCH
  - M.sender = $(IPaddr_p, PortNo_p)$
  - M.data = Metadata(O)
- Send M to one of its K neighbors selected uniformly at random.

Fig. 2. Search algorithm.

to be executed by peer p to initiate a search. Figure 3 shows the algorithm executed by peers along the random walk when a message is received.

D. Analysis of Search Success

Theorem 2: Any object in the network can be found w.h.p in $O(\sqrt{n \ln n})$ hops.

Proof: Let O be the object installed at a peer p. The failure of a search on p's content can be analyzed by defining an indicator random variable X_p in the following manner:

$$X_p = \begin{cases} 1, \text{if } \Gamma_p \cap \Upsilon_q = \emptyset \forall q : q \neq p \\ 0, \text{otherwise} \end{cases}$$

where Γ_p is as defined before, and Υ_q is the set of peers present on the random walk initiated by any peer q issuing a query.

The probability that X_p is equal to one, i.e., Γ_p does not intersect with any Υ_q set, $Pr[X_p = 1] = \left(1 - \frac{\gamma\sqrt{n}}{n}\right)^{\gamma\sqrt{n}}$. The expected value of X_p, $E[X_p] \approx e^{-\gamma^2}$. When $\gamma = \sqrt{\ln n}$, object O cannot be located with probability $\frac{1}{n}$.

Hence, any object in the network can be found w.h.p in $O(\sqrt{n \ln n})$ hops. ∎

This analysis shows that search for even rare objects succeeds w.h.p. The probability of a query succeeding in our algorithm when the TTL of the random walk for search is fixed at m is $1 - e^{-\frac{m^2}{n}}$. For the same TTL, the probability of the query succeeding in a pure random walk algorithm, described in [2], is $1 - e^{-\frac{m}{n}}$. The implications of the difference in these bounds can be better explained with an example. Consider a scenario where there are 10^6 peers in the network and a single copy of an object is present on a single peer. To search the object with the TTL set to 10,000 using a pure random walk [2], the probability that the object is found is approximately 0.01. However, using our algorithm (with $\gamma = 2.5$), pointers to the object are installed at 2,500 peers. The probability that the object is found using our approach with TTL set to 2,500 is 0.99. Therefore, with half of the messages (including the messages for installation of pointers), we guarantee almost deterministic detection of the object.

Process Message: On receiving a message M, peer p executes the steps below:

- If M.type = INSTALL
  - If M.TTL < M.SSIZE
    - Install and index the pointers in M.data locally.
  - Sets M.TTL = M.TTL - 1.
  - If M.TTL > 0
    - Send M to one of its K neighbors selected uniformly at random.
  - Otherwise, do not forward M.
- If M.type = SEARCH,
  - Search the local index for M.data.
  - If p has a pointer to M.data, it creates a new message M′ as follows:
    - M′.type = RESPONSE
    - M′.sender = $(IPaddr_p, PortNo_p)$
    - M′.data = $(IPaddr_q, PortNo_q)$, q is the peer that has a copy of the object.
    - Send M′ to M.sender.
  - If the query cannot be answered by p,
    - Set M.TTL = M.TTL - 1.
    - Send M to one of its K neighbors selected uniformly at random.
- If M.type = RESPONSE, it contacts the node address present in M.data for the object O.

Fig. 3. Algorithm for processing messages.

III. Experimental Evaluation

In this section, we study the performance of our search technique and compare its performance with a pure random walk algorithm, where reference pointers to objects are not installed. In [2], Gkantsidis *et al.* show that a pure random walk algorithm[3] for searching yields better performance than search by flooding. Therefore, we present detailed results of comparisons to the random

[3]We refer to this algorithm as pure random walk in the rest of the paper.

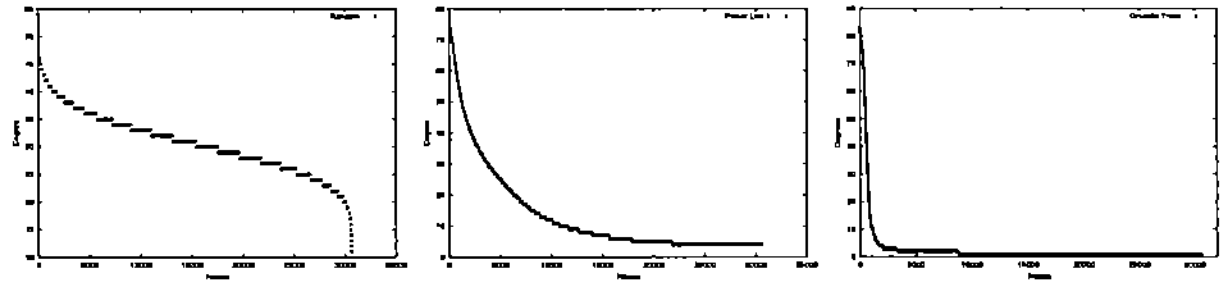

Fig. 4. Degree distribution for the three different topologies used in our experiments.

walk algorithm only. Comparisons to flooding-based schemes show that our technique is significantly superior. Our simulation setup closely resembles the setup in [2] to facilitate accurate comparisons between the algorithms.

We simulate the algorithms over three different topologies. The number of nodes in each topology is fixed to match the number of nodes in a trace of a real Gnutella network.

1) *Real topology trace*: This is a partial view of the Gnutella network that is available at [16]. The number of nodes in this particular view is equal to 30,607. We use this value as reference for the other topologies in order to compare the impact of the topologies on the results. An illustration of this topology is also available at [16].
2) *Random graph*: A normal random graph with 30,607 nodes generated using the Georgia Tech. (GT-ITM) topology generator [17].
3) *Power-law graph*: a random graph where the degrees of the nodes follow a power-law distribution. Power-law graphs have been extensively used to model network topologies. It is believed [14] that the topologies of peer-to-peer networks can be modeled using power-law graphs. We generate a power-law graph with 30,607 nodes, where the maximum degree is chosen to match the maximum degree of a node in the Gnutella trace. The node degrees are chosen from a power-law distribution, and once the degrees of the nodes are chosen, the nodes are connected randomly.

The motivation for selecting three different topologies is to demonstrate that the improvements offered by our algorithm generalize to various commonly accepted network models. The node degree distributions for all the topologies are shown in Figure 4.

In our experiments, we assume that one object is the target of all searches and that the object is replicated at a fraction α of the peers. The fraction α is varied from 0.0033% to 0.12% for most of the experiments. When $\alpha = 0.0033\%$, there is only one copy of the object in the entire network. We also consider the case when an object is extremely popular and is replicated at 50% of the nodes. These values are chosen to simulate objects of differing popularity, from very rare objects to extremely popular ones.

As is well known, P2P networks are characterized by a very dynamic population in which nodes join and leave the network frequently. The dynamic nature of the network population is believed to be an important parameter in these systems. In a system that involves publication of reference pointers, it becomes even more important to examine the impact of varying conditions. The different scenarios used to account for node dynamics are as follows:

1) *Static*: All the peers in the system are present with no departures and change in connections.
2) *Dynamic*: All the peers in the system are present throughout the simulation. However, we periodically select two edges at random (selected uniformly) and exchange their end points. This characteristic is closely related to the operation of a real P2P system, where nodes frequently choose to reconnect to other nodes. The same scenario is also used in [2].
3) *Failures without updates*: In this scenario, when a peer leaves the network, it is replaced by a new peer. The neighbors of the new peer are chosen randomly from the peers in the network and are not related to the peer leaving the network. We assume that the object owners do not leave the network during the simulation, but peers holding pointers to the objects can leave. In this specific setup, the object owner does not update the pointers to new peers in the network. Pointers to the objects are installed only at the beginning of the experiment.
4) *Failures with updates*: This is the same as the previous scenario, except that the owner of an object periodically (after τ units of time) installs

its pointers in a set of $\frac{f\gamma\sqrt{n}}{100}$ peers picked uniformly at random from the network, where f is the percentage of peers leaving the network in a period equal to τ. In [14], Saroiu et al. show that 50% of the peers leave the network in a period of one hour. In this case, $f = 50$ and $\tau = 60$.

We first investigate the average number of hops necessary to resolve a query. In this scenario, all nodes in the network issue one query for the object. We assume that the queries are issued after the object owners have already installed pointers to the objects. We simulate random walks of unbounded length, the random walk stops only when the object or a pointer to it is found.

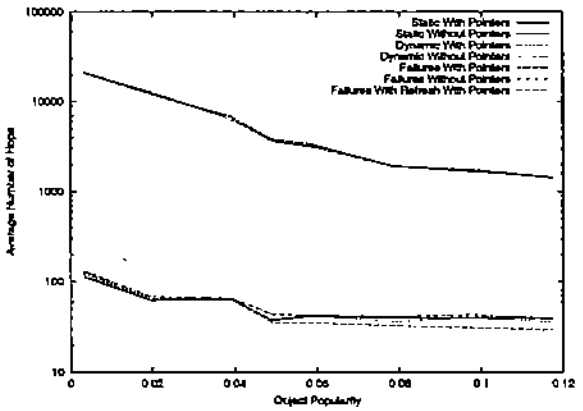

Fig. 5. Average number of hops in the Gnutella topology.

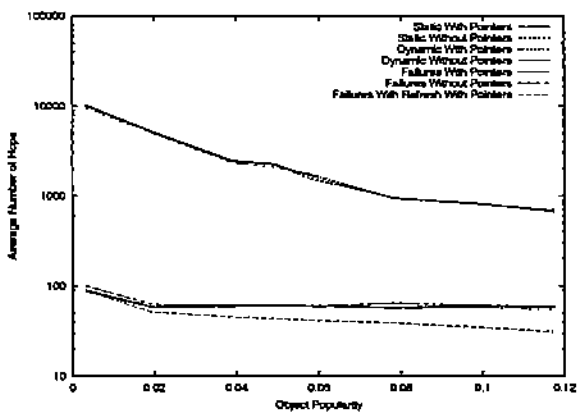

Fig. 6. Average number of hops in the power-law topology.

Figures 5, 6, and 7 present the average number of hops to successfully resolve a query as a function of object popularity for different topologies and compares our approach to the pure random walk. The average number of hops is on a log scale. As can be observed for the static case, when the object is present at a single peer, the average number of hops to successfully find the object is

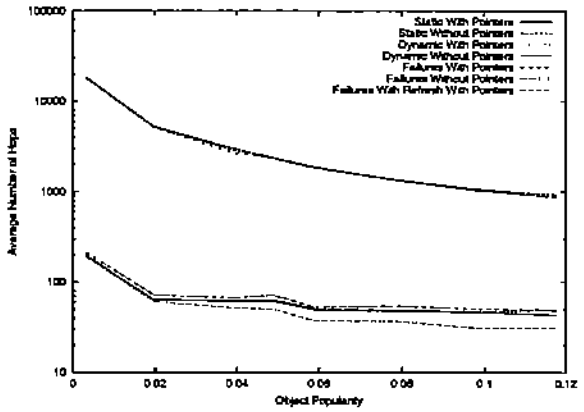

Fig. 7. Average number of hops in the random-graph topology.

approximately 100 while the pure random walk method needs approximately 20,000. This is an improvement of over two orders of magnitude. The number of hops for installation of pointers in our method is not accounted for. Even including the installation of pointers for every search (the worst case assumption), the average number of hops will still be less than 300 (we use $\gamma = 1$), a 600 fold improvement.

The dynamic scenario is similar to static one as both the methods involved are random walks and are independent of edge connections (on an average). When peers fail without refresh, it can be observed that the average number of hops has a slight increase. However, when peers fail with refresh, the average number of hops decreases slightly due to periodic refreshing of pointers onto new nodes. Since the owners of the object do not fail in the network and only the others (including peers which installed pointers to the object) fail, pure random walk is not affected by failures without refresh. When the object popularity increases to 0.12%, the performance improvement is still more than one order of magnitude. We can also observe in the figures that the average number of hops is much smaller than $\sqrt{n \ln n}$ (in this case, 562 approximately), the suggested theoretical maximum length of the random walk for our protocol.

In the next experiment, we investigate the percentage of queries that fail when the TTL of the random walk is bounded. In this scenario, all nodes in the network issue one query for the object. We assume that the queries are issued after the object owners have already installed pointers to the objects. The random walk will end whenever a response to a query is found or it has reached its TTL.

Figure 8 shows the percentage of queries failing for different topologies as a function of object popularity. We experiment with two values of γ ($\gamma = 1$ and $\gamma = 2$).

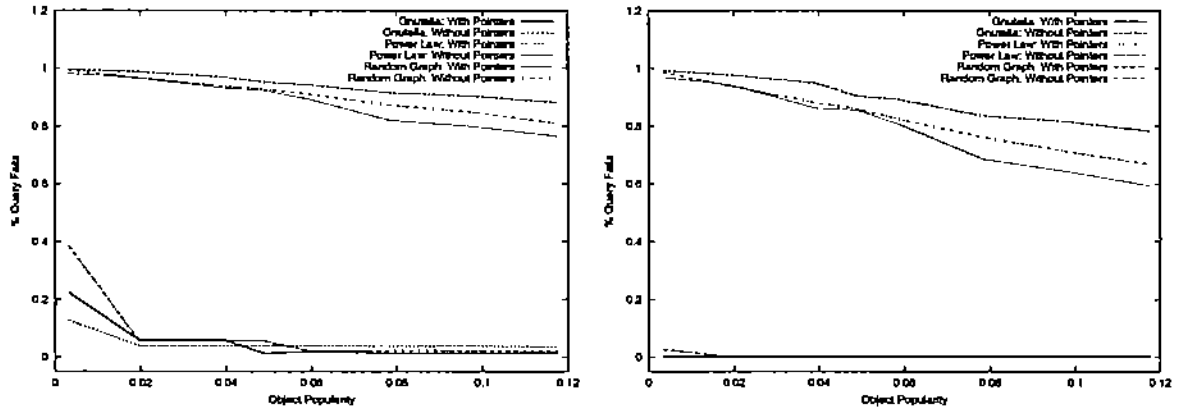

Fig. 8. Percentage of failures of a query as a function of object popularity. TTL = $\gamma\sqrt{n}$. left $\gamma = 1$ and right $\gamma = 2$.

In our approach, when there is a single object present in the network and $\gamma = 1$, the influence of topology is quite significant. The random graph topology has the lowest failure rate at 15% with the highest being 39% in a power law topology. We believe that this is a direct consequence of the power law nature of the graph. The intuitive reasoning is that the random walk needs to take more hops from a high degree peer to move from one locality to another (the locality being defined as the concentration of peers around a high degree peer). Also, this corroborates our analysis that the probability of failure of a search is approximately equal to $\frac{1}{e^{\gamma^2}}$. Here $\gamma = 1$ and the probability of failure is approximately 0.37. With increase in object popularity, the failure percentage of our method approaches zero rapidly. Pure random walk almost always fails when there is a single object in the network and with increase in object popularity, around 20% of queries are successful. In our approach almost all the queries are successful when $\gamma = 2$, while the pure random gives a maximum of 20% success rate when the object popularity is increased to the highest value used in our experiment.

The number of messages per peer, in our approach as well as the pure random walk approach, is a function of the degree of a peer in the network. A higher number of messages are processed by peers with high degree. In comparison with the pure random walk approach, the number of messages per peer in our method is at least 20 times and sometimes 900 times less than the messages in pure random walk. Figure 9 shows the number of messages per peer for different topologies when the network is static. The number of messages is represented on a logarithmic scale.

Another parameter that we investigate in our experiments is the percentage of peers owning an object that install pointers to an object in the network. In this scenario, the popularity of an object is varied from 0.0033% to 50%. The experiment is conducted for different topologies under static conditions. There are no searches carried out after the installation phase.

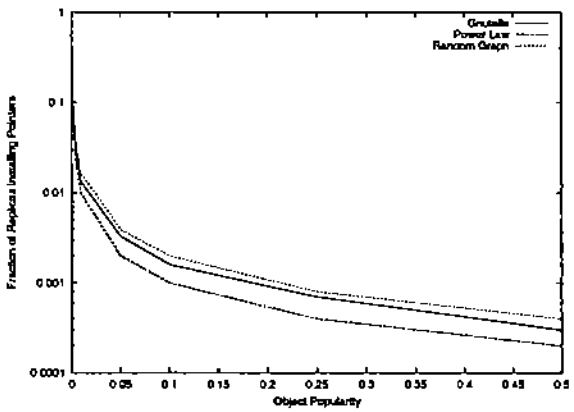

Fig. 10. Percent of object owners installing pointers.

Figure 10 shows, on a logarithmic scale, the percentage of object owners installing pointers to an object for different topologies. When almost 50% of the peers own a copy of the object, only a very small percentage (0.0002) of the peers owning the object initiate a random walk to install pointers to the object. This is result is due to our algorithm for controlling installation of pointers. In this algorithm, a peer installs pointers to its object with probability proportional to the length of the random walk for querying the object.

In the next experiment, we study the changes in the average number of hops with failing peers. In this scenario, after the completion of the installation phase, we remove a fraction β ($0.1\% \leq \beta \leq 50\%$) of peers from the network. Peers that are owners of the object are never removed. The same number of peers that are removed

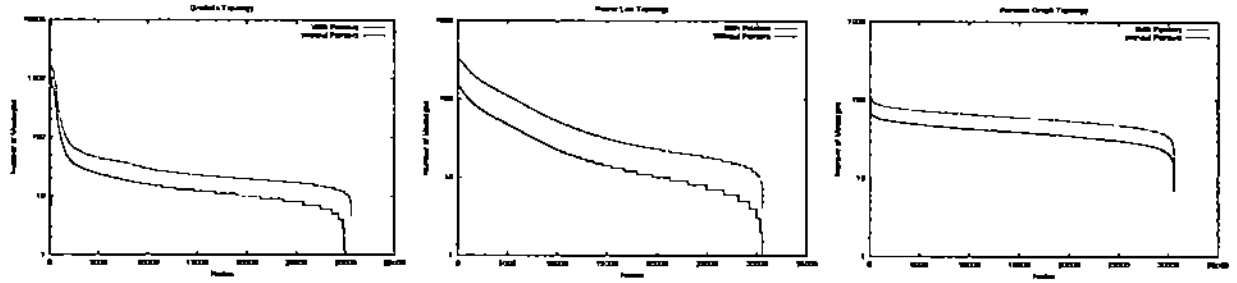

Fig. 9. Number of messages per peer.

are immediately reintroduced to keep the network size constant. The new peers, however, connect to different peers than the ones leaving the newtork, and do not posses any pointers to objets. The TTL in this case is not bounded; a random walk stops only when the object is found.

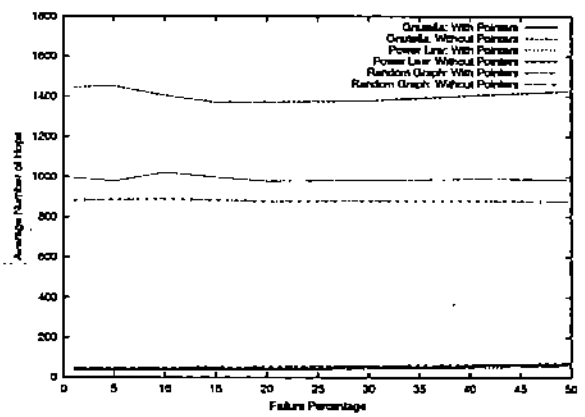

Fig. 11. Average number of hops as a function of failure rate of peers, when object popularity=0.1%.

Figure 11 shows the average number of hops needed for a query to be successful as function of peer failures for different topologies. The object popularity is set at 0.1%. With increase in the failure percentage, there is little increase in the average number of hops for a successful query using our approach. Pure random walk, as expected, requires more hops for a successful query.

The conclusions of the experiments are:

- Installing pointers to an object at $\sqrt{n}$ peers selected uniformly at random from the network results in probability of success close to one for queries initiated from any node in the network.
- The cost associated with a search is drastically reduced (by orders of magnitude) using our approach and the number of messages processed per each peer is reduced as well.
- The cost of the installation phase is negligible when compared to the cost of finding an object using the pure random walk method.
- The approach is robust under realistic failure models.
- Controlled replication helps in limiting the number of peers having a replicated copy of an object installing a pointer to it.

IV. Application to Related Problems in Object Replication

The general framework presented in this paper can be applied to a number of related problems. These include estimation of the number of replicas of an object, controlling replication for caching mechanisms, and duplicate elimination. We motivate the importance of these problems using a simple example. Consider an archival P2P system in which each peer archives its content probabilistically based on the number of peers that also own the same file in the network. If a peer is not a unique owner of the file, then archiving the file at each peer independently results in storage redundancy (albeit to the benefit of improved end-user latency). Furthermore, it is quite common in a P2P system for multiple peers to own a copy of the same file. Hence, it becomes important to estimate the number of copies present in the network and eliminate redundant copies.

In general, our method can be extended to any operation that requires estimating the number of nodes K interested in performing a specific operation with the following restrictions:

1) The identity of all the nodes involved is not known.
2) Broadcasting over the entire network is not a plausible solution.
3) Approximate estimation is sufficient.

Each peer interested in performing the estimate sends an *estimate* message to a set of $\gamma\sqrt{n}$ peers chosen uniformly at random from the network. This can be performed by each peer doing an independent random walk on the network. After this, each peer performs

another random walk by sending messages to $\gamma\sqrt{n}$ peers and requesting them to send back the number of *estimate* messages they received. Based on this, each peer p estimates the value as follows. If Y_i is the number of *estimate* messages received by peer i, then peer p estimates K as $\frac{\sum_{i=1}^{\gamma\sqrt{n}} Y_i}{\gamma^2}$.

Theorem 3: If peer p estimates K as $\frac{\sum_{i=1}^{\gamma\sqrt{n}} Y_i}{\gamma^2}$, then the standard deviation of the estimate is given by $\frac{\sqrt{K}}{\gamma}$.

Proof: The estimate problem can be reduced to a balls and bins problem. Let K be the number of peers interested in the estimate. When throwing m balls randomly into n bins, the probability p_r that a bin has r balls is approximately the Poisson distribution with mean $\frac{m}{n}$. In our problem, the number of balls thrown are $K * \gamma\sqrt{n}$, and the number of bins are n. The average number of balls in each bin is $\frac{K*\gamma\sqrt{n}}{n}$. Let Y_i be a random variable that denotes the number of *estimate* messages received by a peer i, then K is estimated as follows:

$$\frac{K * \gamma\sqrt{n}}{n} = \frac{\sum_{i=1}^{\gamma\sqrt{n}} Y_i}{\gamma\sqrt{n}} \Rightarrow K = \frac{\sum_{i=1}^{\gamma\sqrt{n}} Y_i}{\gamma^2}$$

The variance σ^2 for the estimated value of K is:

$$Var\left[\frac{\sum_{i=1}^{\gamma\sqrt{n}} Y_i}{\gamma^2}\right] = \frac{\sum_{i=1}^{\gamma\sqrt{n}} Var[Y_i]}{\gamma^4} = \frac{\gamma\sqrt{n}\gamma\sqrt{n}K}{\gamma^4 n} = \frac{K}{\gamma^2}$$

Hence, the standard deviation is given by $\frac{\sqrt{K}}{\gamma}$. ■

V. Related Work

In [8], Cohen *et al.* improve the efficiency of search in unstructured P2P networks by replication. They evaluate two different replication strategies, uniform and proportional, and show that both the strategies have the same average performance on successful queries and an optimal replication strategy lies between the two schemes. The replication strategies assume that access frequencies of the objects are known, and the replication of an object should be based on its popularity. In our approach, we do not assume knowledge of object access frequency, and use replication of pointers to speed up queries. Moreover, our technique does not distinguish between frequently and infrequently accessed objects. An important consequence of our approach is that we can provide probabilistic guarantees on locating even rare objects in the network. In this sense it complements the work of Cohen and Shenker, and other replication and caching mechanisms.

Lv *et al.* [1] show, using simulations, that random walk is a good technique for searching unstructured P2P networks. They also present a detailed study of search and replication in unstructured networks, including the square root replication policy developed by Cohen *et al.* [8], and their impact on searches for popular objects. They report that a random walk approach has better performance compared to the standard approach of searching by flooding. Their result complements our work. We show that the random walk approach with installed references reduces the communication overhead (messages for installing references and searching for objects) by an order of magnitude when compared to pure random walk approach.

In [2], Gkantsidis *et al.* perform an extensive study of random walks in P2P networks. The authors explore the performance of random walks for searching and uniform sampling. For searching, the authors show that random walks perform better than flooding when the length of the random walk is the same as the number of peers covered by flooding with bounded TTL. Another important result in the paper is that it is possible to simulate selection of a uniform sample of elements from the entire P2P by performing a random walk of required length. We use this result in our protocol.

In [15] Malkhi *et al.* show, based on the birthday paradox, that any two quorums of size $\Omega(\sqrt{n})$, where quorum members are picked uniformly at random, intersect with high probability. Probabilistic quorums were proposed in the context of distributed systems where nodes have complete knowledge about all other nodes in the system, and the population of nodes is stable. In P2P networks, it is not feasible for nodes to have complete knowledge of the network. Moreover, peers leave the network frequently. Our protocol is also influenced by the birthday paradox, but we do not make any assumptions of the knowledge about other peers in the network, and we show that our protocol can operate in a dynamic scenario.

Adamic *et al.* [18] propose algorithms for search in power-law networks. Searches are also performed using random walks, but the random walk is biased toward high-degree nodes. When a node has to forward a query message, it forwards the message to its neighbor of highest degree. The paper shows that this approach can improve significantly the search for popular objects. One shortcoming of this approach is that the high-degree nodes in the network are responsible for processing the query load of a significant fraction of the network.

Yang et al. [9] study several search strategies for unstructured P2P networks. All the proposals (iterative deepening, directed BFS, and local indices) focus on searching only, and not on placement. As a result, none of the proposals can guarantee that rare objects can be found. Crespo et al. [19] place routing indices (hints) at each peer, which allows peers to forward queries to

neighbors that are likely to have answers. The random walk search in [20] also uses hits to improve search. These approaches do not provide guarantees on the success of a search. In contrast, there are no hints in our system (the pointers are answers to queries), we select neighbors randomly, and provide strong guarantees for search.

As mentioned earlier in this paper, structured P2P networks [3], [4], [5] provide strong guarantees for search in P2P networks by imposing a well defined topology on the network. Peers and objects are assigned hash-based identifiers and objects are assigned to peers based on these identifiers. Objects in the network are found by performing efficient routing protocols that lead to the peers responsible for storing pointers to the objects. In addition to the overheads of maintaining a structured overlay, these methods cannot handle complex queries efficiently.

VI. Conclusion

In this paper, we present the design of an efficient search protocol for unstructured P2P networks that provides probabilistic guarantees on the success of a search. To the best of our knowledge, ours is the first result that provides (probabilistic) guarantees of detecting even rare objects in unstructured P2P networks with reasonable bounds on associated overhead. Supported by elaborate experiments, we show that our approach is an improvement, by over an order of magnitude, compared to the pure random walk method. We also present and analyze an application of our scheme to the problem of estimating the number of peers interested in performing a specific operation, such as duplicate elimination and controlled replication.

References

[1] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ACM ICS'02 Conference*, New York, NY, USA, June 2002.

[2] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *Proceedings of IEEE INFOCOM, 2004*, Hong Kong, March 2004.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, San Diego, CA, August 2001, pp. 149–160.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, San Diego, CA, August 2001, pp. 247–254.

[5] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-0101141, UC Berkeley, Computer Science Division, April 2001.

[6] Limewire., "http://www.limewire.com/," .

[7] Gnutella., "http://gnutella.wego.com/," .

[8] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *ACM SIGCOMM'02 Conference*, 2002.

[9] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks," in *IEEE International Conference on Distributed Systems (ICDCS)*, Vienna, Austria, July 2002.

[10] K. Horowitz and D. Malkhi, "Estimating network size from local information," in *The Information Processing Letters journal 88(5)*, December 2003. pp. 237–243.

[11] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, "Estimating Aggregates on a Peer-to-Peer Network.," Technical Report, Computer Science Department, Stanford University, 2003.

[12] D. Psaltoulis, D. Kostoulas, I. Gupta, K. Birman, and A. Demers, "Practical Algorithms for Size Estimation in Large and Dynamic groups.," http://www.cs.cornell.edu/Info/Projects/Spinglass/index.html.

[13] R. Motwani and P. Raghavan, ," in *Randomized Algorithms*. Cambridge University Press, 1995.

[14] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[15] D. Malkhi, M. Reiter, and R. Wright, "Probabilistic quorum systems," in *Proceedings of the 16th Annual ACM Symposium on the Principles of Distributed Computing (PODC 97)*, Santa Barbara, CA, August 1997, pp. 267–273.

[16] Snapshots of the Gnutella network. Limewire.org, "http://crawler.limewire.org/data.html," .

[17] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *Proceedings of IEEE INFOCOM 1996*, March 1996.

[18] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in Power Law Networks," *Physical Review E 64*, pp. 46135.1–46143.8, 2001.

[19] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proceedings of the The 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, July, 2002.

[20] Freenet., "http://freenet.sourceforge.net/," .