# Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks

Ronaldo A. Ferreira   Murali Krishna Ramanathan   Asad Awan   Ananth Grama   Suresh Jagannathan
Department of Computer Sciences – Purdue University
West Lafayette, IN, USA
{rf, rmk, awan, ayg, suresh}@cs.purdue.edu

## Abstract

*Search is a fundamental service in peer-to-peer (P2P) networks. However, despite numerous research efforts, efficient algorithms for guaranteed location of shared content in unstructured P2P networks are yet to be devised. In this paper, we present a simple but highly effective protocol for object location that gives probabilistic guarantees of finding even rare objects independently of the network topology. The protocol relies on randomized techniques for replication of objects (or their references) and for query propagation. We prove analytically, and demonstrate experimentally, that our scheme provides high probabilistic guarantees of success, while incurring minimal overhead. We quantify the performance of our scheme in terms of network messages, probability of success, and response time. We also evaluate the robustness of our protocol in the presence of node failures (departures). Using simulation, we show that our scheme performs no worse than the best known access-frequency based protocols, without compromising access to rare objects.*

## 1. Introduction

Search is a fundamental service in peer-to-peer (P2P) networks, one that has received considerable research attention [10, 5, 15, 13]. 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. More importantly, they are simple to implement and incur virtually no overhead in topology maintenance. Consequently, many real-world large-scale peer-to-peer networks are unstructured. Gnutella networks, like Limewire [8], for example, have peak populations in the order of millions of users.

In typical unstructured P2P networks, such as Gnutella [6], a peer searches by flooding a (hop) limited neighborhood. This simple method does not provide any guarantee that an object that exists in the network can 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 [3, 10]. Cohen *et al.* [3] studies how replication can be used to improve search in unstructured P2P networks. Objects are replicated based on their access frequencies. The paper evaluates different replication strategies and concludes that the optimal replication strategy (the one that minimizes the average search size) is proportional to the square root of the access frequencies. Lv *et al.* [10], in a followup work, suggests heuristics to approximate the optimal replication strategy.

Our work concentrates on the problem of 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 [10]. The objective of providing search guarantees, while simultaneously minimizing messaging overhead, is important, and has been solved elegantly for structured P2P networks [15, 13], but it has not been fully explored in unstructured networks. The lack of such guarantees in unstructured networks has prevented the development of new applications that can exploit the benefits of this environment. For instance, completely distributed Grid environments could benefit from search mechanisms that guarantee retrieval of any available resource in the network. Similarly, in massively distributed file systems such mechanisms can be incorporated to preserve guaranteed file access semantics.

In this paper, we present a simple but highly effective technique for object location based on a variant of the birthday paradox. Our scheme installs object references at a set with $(O(\gamma\sqrt{n}))$ of randomly selected peers, where $n$ is the number of peers in the network and $\gamma$ is a system parameter. A query to the object is routed to another set with $O(\gamma\sqrt{n})$ 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. 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 $\gamma$, the likelihood of a query failing is negligible. By choosing $\gamma$ to be $\sqrt{\ln n}$, we show that any search succeeds with high probability (w.h.p.[1]).

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 problem, we present a novel approach to controlled reference installation, which we demonstrate strikes a desirable balance between reference installation and search overheads. Experimental results show that our strategy yields slightly better average search size than the best heuristic presented in [10], and in addition provides strong guarantees for locating rare objects. We address these issues using an analytical framework and validate our results on a variety of real (Gnutella) and synthetic topologies.

## 2. Placement and Search Protocols

To share its content with other peers in the network, a peer must install references to each of its objects at other peers. This installation procedure is similar to the process of publishing content in structured peer-to-peer networks. However, in our approach a peer $p$ can publish an object $O$ by installing references to $O$ at *any random set* $\Gamma_p$ of peers selected independently of $O$'s identifier (or hash). Moreover, an object can be published using any form of identification, for example, a file name or meta information about the object.

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 $\Gamma_p$ be located in the network? 2) What is the size of the set $\Gamma_p$? 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 simple – nodes in the set $\Gamma_p$ are selected uniformly from the network, i.e., any node in the network has an equal probability of belonging to the set. This selection criterion is important because it provides a measure of fault tolerance, i.e., in the event of a node failure, the failing node can be easily replaced. Since conventional P2P networks are expected to scale to millions of peers and peers only connect to a small set of other peers, creating a uniformly sampled set in the network is not a straightforward task. Gkantsidis *et al.* [5] shows 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 [2]. The last $k$ peers encountered in the walk represent a uniform random sample. This result, however, applies to network topologies that can be modeled as expander graphs.

For more generic networks, like the ones encountered in real-world networks, an auxiliary algorithm must be executed to determine probabilistic weights for different branches of the random walk. The theory of uniform sampling in non-uniform graphs has been extensively studied in a number of works [7, 1]. For the sake of completeness, in Section 2.3 we briefly discuss the Metropolis-Hastings (MH) algorithm that we use in our simulations.

The second and third questions above can be answered using the birthday paradox [12]. The birthday paradox has been used in [11] to define probabilistic quorums and in [2] to estimate aggregates in distributed systems. In a probabilistic quorum, quorum members are chosen uniformly at random from all members in the network, and the quorums have size $\gamma\sqrt{n}$. Malkhi *et al.* [11] shows that two quorums chosen in this manner intersect with probability at least $1-e^{-\gamma^2}$. Probabilistic quorums were 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.

```
procedure Publish($p$):
  ▷ Create a message M as follows:
      – M.TTL = $c \times \log n + \gamma\sqrt{n}$
      – M.SSIZE = $\gamma\sqrt{n}$
      – M.type = INSTALL
      – M.sender = $(IPaddr_p, PortNo_p)$
      – M.data = Metadata($O$)
  ▷ Send M′ to a neighbor $j$ selected with probability $Pr_{pj}$.
```

**Figure 1. Algorithm for publishing content.**

Using the results above, the protocol employed by a peer to publish its content follows naturally. A peer $p$ publishes its content by performing a random walk of length $c \times \log n + \gamma\sqrt{n}$, where $c$ is a constant. In the random walk message, $p$ includes information about an object $O$ (or a set

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

[2]The exact length of the random walk involves the second eigenvalue ($\lambda_2$) of the network graph [5].

of objects) and requests the peers along the random walk to install references to it. The first $c \times \log n$ peers just forward the random walk message, the remaining $(\gamma\sqrt{n})$ peers install locally the references to $p$'s content and send back to $p$ their identifications (IP addresses). Node $p$ uses these peers as its set $\Gamma_p$. Figure 1 illustrates the publish algorithm and Figure 2 shows the algorithm executed by peers along the random walk when a message is received. In the algorithms, M.SSIZE specifies the size of the set $\Gamma_p$ and is used by peers along the random walk to determine if the content (*Metadada*$(O)$) present in the message must be installed or not. $Pr_{pj}$ is the probability of a node $j$ being selected as the next hop by node $p$. In Section 2.3, we show how these probabilities are computed.

```
procedure ProcessMessage($p$):
  ▷ 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 a neighbor $j$ selected with probability $Pr_{pj}$.
          ○ 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 a neighbor $j$ selected with probability $Pr_{pj}$.
      – If M.type = RESPONSE, $p$ retrieves object $O$ from the node whose address is in M.data.
```

**Figure 2. Algorithm for processing messages.**

## 2.1. Controlled Installation of Object References

The question of whether the replica of an object installs 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 $c \times \log n + \gamma\sqrt{n}$. 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 $\frac{l}{c \times \log n + \gamma\sqrt{n}}$. We show in Section 3 that this algorithm significantly reduces the number of reference pointers to popular objects without significantly reducing the probability of finding rare objects.

```
procedure Join($p$):
  ▷ 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 = $c \times \log n + \gamma\sqrt{n}$
      – M.type = SEARCH
      – M.sender = $(IPaddr_p, PortNo_p)$
      – M.data = Metadata($O$)
  ▷ Send M to a neighbor $j$ selected with probability $Pr_{pj}$.
  ▷ If M is answered by a peer at distance $l$, with probability $\frac{l}{c \times \log n + \gamma\sqrt{n}}$ execute the procedure Publish.
```

**Figure 3. Join algorithm.**

## 2.2. Object Search

A peer $p$ searches for 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 $c \times \log n + \gamma\sqrt{n}$. 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 4 presents the algorithm executed by a peer $p$ to initiate a search.

```
procedure Search($p$):
  ▷ To search for an object $O$, peer $p$ executes the steps below:
      – Create a message M as follows:
          ○ M.TTL = $c \times \log n + \gamma\sqrt{n}$
          ○ M.type = SEARCH
          ○ M.sender = $(IPaddr_p, PortNo_p)$
          ○ M.data = Metadata($O$)
      – Send M to a neighbor $j$ selected with probability $Pr_{pj}$.
```

**Figure 4. Search algorithm.**

## 2.3. Uniform Sampling

In a simple random walk, all neighbors of a node $p$ have the same probability of being selected as the next hop, i.e., $Pr_{pj} = 1/K$. It is well known [12] that a simple random walk in an undirected graph of a minimum length proportional to $\log n$ converges to a stationary distribution $\pi^T$. The probability of a node $i$ being selected as a sample of the distribution $\pi^T$ is equal to $d_i/2m$, where $d_i$ is the degree of node $i$ and $m$ is the number of edges of the graph. In real-world network, the degrees of the nodes vary significantly. Therefore, a simple random walk selects nodes with different probabilities, with low degree nodes having lower probabilities of being selected. We present an algorithm that modifies the transition probabilities between nodes to produce a probability transition matrix with stationary uniform distribution. Thus, a random walk on a network with node transition probabilities defined using this algorithm results in a uniform sample. The algorithm is traditionally presented in the context of Markov chains, but its adaptation to a distributed network is straightforward [1].

**Metropolis-Hastings**

The algorithm is an adaptation for uniform sampling of the classical Metropolis-Hastings algorithm [7, 1]. In the distributed algorithm, each node $i$ sends a message stating its degree information, $d_i$, to each of its neighbors $j \in \Psi(i)$, where $\Psi(i)$ is the set of nodes directly connected to $i$. Once the information of each of the neighbors is received, the transition probabilities are set up as follows:

$$Pr_{ij}^{mh} = \begin{cases} 1/max(d_i, d_j) & \text{if } i \neq j \text{ and } j \in \Psi(i) \\ 1 - \sum_{j \in \Psi(i)} (Pr_{ij}^{mh}) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

The result of this algorithm in the entire network is the implicit construction of a double stochastic matrix. A random walk of a given minimum length using the probabilities above results in uniform samples. Due to space limitations, we refer the reader to [7, 1] for more details on the algorithm and its associated theory.

## 2.4. Failures

An important aspect of current P2P networks is the departure or failure of nodes. To account for this, we must install additional references. The following theorem determines the number of references.

**Theorem 1** *If $f$ denotes the fraction of nodes leaving the network in a given time period, $f * |\Gamma_p|$ new references need to be installed after the time period to maintain search success rate.*

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

## 2.5. Analysis of Search Success

The success of the search algorithm used in conjunction with the publish algorithm is quantified by the following theorem.

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

*Proof:* Let $O$ be any object owned by 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 $\Gamma_p$ is as defined before, and $\Upsilon_q$ 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., $\Gamma_p$ does not intersect with $\Upsilon_q$, $Pr[X_p = 1] = \left(1 - \frac{\gamma\sqrt{n}}{n}\right)^{\gamma\sqrt{n}}$. The expected value of $X_p$ ($E[X_p]$) is approximately $e^{-\gamma^2}$. When $\gamma = \sqrt{\ln n}$, object $O$ cannot be located with probability $\frac{1}{n}$. Therefore, any object in the network can be found w.h.p in $O(\sqrt{n \ln n})$ hops. □

## 3. Experimental Evaluation

In this section, we study the performance of our search technique and compare its performance with different algorithms presented in the literature. We first investigate the case where there are multiple objects in the network and nodes have fixed cache sizes. In order to certify that our controlled replication strategy minimizes the average search size, we compare our scheme with the best heuristic presented and evaluated in [10]. We use the same parameters presented in [10] to avoid complicated tunings of the two algorithms. Later in this section, we investigate the performance of our scheme under different parameters and much larger networks.

Lv *et al.* [10] proposes a heuristic that replicates objects proportional to the search size. Each query keeps track of

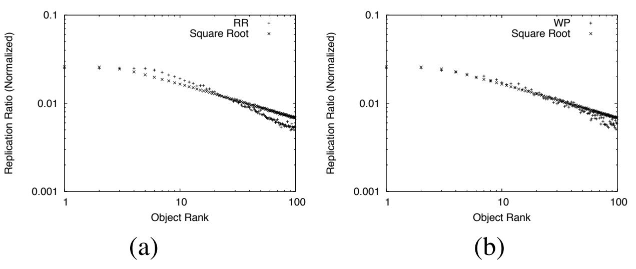

**Figure 5. Distribution of replication ratios of the two different strategies. Objects are ranked by their access frequencies, with low rankings indicating highly accessed objects.**

the search size and when the query is finished the object is replicated on as many sites as the number of probed nodes. The replicas are placed in nodes selected uniformly from the network. This heuristic is justified by observing that the number of replicas of an object $i$ ($r_i$) can be described by the differential equation $\dot{r}_i = q_i c \frac{n}{r_i}$, where $\dot{r}_i$ is the time derivative of $r_i$, $q_i$ is the query frequency of $i$, and $c$ is a constant.

Furthermore, $r_i = \lambda\sqrt{q_i}$ is a fixed point of the equation involving the logarithm of the object ratios. This simple calculation suggests that replicating proportional to the number of sites probed would yield square-root replication, which has been shown to be optimal [3].

We reproduce the set of experiments presented in [10] to compare our strategy, which we refer as WP – With Pointers, with the best heuristic (RR - Random Replication) presented in [10]. We simulate two different topologies: random graph with 9,836 nodes and power-law with 9,230 nodes. These numbers and the remaining of the parameters described next are exactly the same as the ones presented in [10]. The simulation starts by uniformly placing $m = 100$ distinct objects into the network. Then queries to objects are generated according to a Zipf-like distribution with parameter $\alpha = 1.2$. Each query is initiated from a node that does not have a copy of the object being queried, querying nodes are selected uniformly from the network. Each node has a small cache that can store at most 40 objects. The cache replacement policy is Random Deletion. That is, when a cache eviction is necessary, an object is selected randomly from all objects present in the cache. As discussed in [10], Random Deletion is one of the policies that can help in achieving Square-Root replication. Policies based on access frequencies, like LRU, do not have this property.

Figure 5 shows the replication ratios produced by the two heuristics in the power-law graph, the random graph yields similar results. Figure 5-(a) shows the results for the RR heuristic and Figure 5-(b) shows the results for our heuristic. The reference points (Square Root) represent the distribution that is the square root of the query distribution. As we can see, our strategy better approximates the optimal strategy. The performance gain, however, is minimal. The average search size in our scheme is only two percent better than in the RR heuristic. Our main point is to show that our scheme is as good as the best replication heuristic currently known while having the extra benefit of giving strong guarantees of locating rare objects. This last point will be made clear in the next experiments. The striking difference between the two strategies can be seen at the tail of the distribution. Our strategy places a few more copies of low ranked objects (about 12% more copies for the 40% less popular objects) while placing copies no more than the necessary for high ranked objects.

The previous experiment shows that our scheme is competitive with the best known heuristic that replicates objects based on access frequencies. We now turn our attention to scalability and accuracy issues of our scheme. In the next set of experiments, we assume that one object is replicated at a fraction $\alpha$ of the peers. The fraction $\alpha$ is varied from 0.0033% (a single object is present in the network) to 0.12%. These values are chosen to simulate objects of differing popularity, from very rare objects to extremely popular ones. For this particular set of experiments, the frequencies of accesses of the objects are not taken into consideration. Pointers to objects are possibly installed only by the owners at the beginning of the simulation and queries to the single object are performed from all other nodes. We show our results using as comparison a simple random walk scheme without replication. The main goal of these experiments is to show that rare objects can be found using searches with bounded TTL. We also investigate the impacts of simple random walks in our scheme and random walks using the transition probabilities computed by the Metropolis-Hastings algorithm.

Our simulation setup closely resembles the setup in [5] to facilitate accurate comparisons with the base case. We simulate the algorithms over three different topologies: random graph, power-law graph, and real topology trace. Due to space limitations, we show only the results for the real topology trace, the results for the other topologies are qualitatively similar to the ones obtained with the trace. We refer the reader to [4] for the results on the other topologies and for a more extensive study of different parameters. The topology trace is a partial view of the Gnutella network that is available at [9]. The number of nodes in this particular view is equal to 30,607. An illustration of this topology is also available at [9]. In this trace, most nodes have low degrees and a few nodes have high degrees.

The dynamic nature of the network population is believed to be an important parameter in P2P 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 [5].
3. *Failures without updates*: In this scenario, when a peer leaves the network, a new peer joins the network keeping the size of the network constant. We assume that the object owners do not leave the network during the simulation, but peers holding pointers to the objects can leave. Pointers to objects are installed only at the beginning of the experiment, object owners do not install pointers in the new peers.
4. *Failures with updates*: This is the same as the previous scenario, except that the owner of an object periodically (after $\tau$ 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 $\tau$. 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 $\tau = 60$.

The average number of hops necessary to resolve a query gives a measure of the network overhead of a search scheme, we start by investigating this parameter. In this simulation 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.

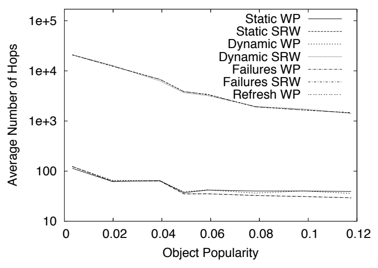

**Figure 6. Average number of hops in the Gnutella topology.**

Figure 6 presents the average number of hops to successfully resolve a query as a function of object popularity for our approach (WP) and for the base case using pure random walk (SRW). The base case is to give an idea of the improvements that can be achieved by using pro-active replication. The average number of hops is on a log scale. In these experiments, $\gamma$ is fixed to 1. As can be observed for the static case, when the object is present at a single peer ($\alpha = 0.0033\%$), the average number of hops to successfully find the object is approximately 100, which is much smaller than $\sqrt{n \ln n} = 562$, the suggested theoretical limit for guarantees of success. The three superior lines represent the results for the pure random walk scheme, while the inferior lines represent the results for our scheme. 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 fail, pure random walk is not affected by failures without refresh.

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. We set TTL to $\gamma\sqrt{n}$ and use different values for $\gamma$.

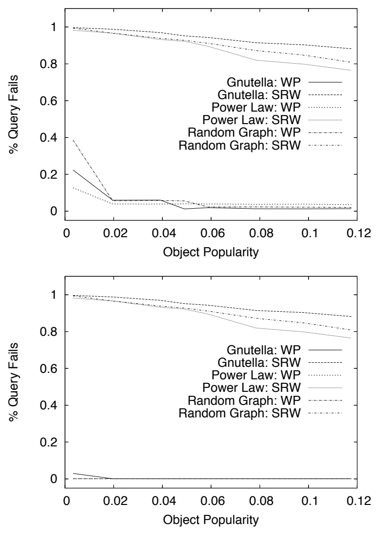

**Figure 7. Percentage of failures of a query as a function of object popularity. TTL = $\gamma\sqrt{n}$, top $\gamma = 1$ and bottom $\gamma = 2$.**

Figure 7 shows the percentage of queries failing for different topologies as a function of object popularity. 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 of queries 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). We explore this issue further in this section when we discuss the impact of the MH algorithm in node sampling. 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. In our approach almost all the queries are successful when $\gamma = 2$, while the pure random walk gives a maximum of 40% success rate when the object popularity is increased to the highest value used in the experiment.

Another parameter that we investigate in our scheme 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. Figure 8 shows, on a logarithmic scale, the percentage of object owners that install 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 install pointers to the object. This result is due to our algorithm for controlling installation of pointers. In this algorithm, a peer installs pointers to an object with probability proportional to the length of the random walk for querying the object.

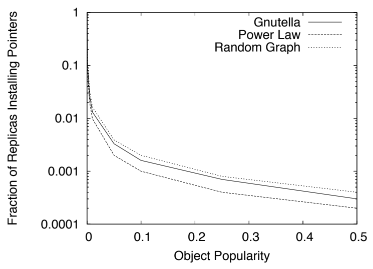

**Figure 8. Percent of object owners installing pointers.**

As discussed in Section 2.3, a random walk is capable of producing random samples of a graph. The samples, however, are not uniform, but instead depend on the degree of the nodes. In the next set of experiments, we investigate the differences between samples produced using simple random walks and uniform samples produced using random walks with the transition probabilities computed with the MH algorithm. We also compare both schemes with the pure random walk. Figure 9 shows the average number of hops for a successful query for the three different schemes. As we can see, the impact of the MH algorithm in our scheme is negligible.

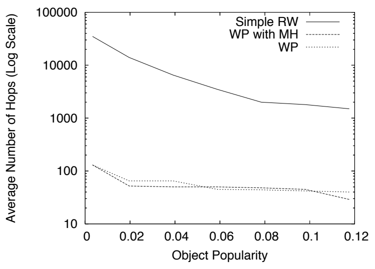

**Figure 9. Average number of hops for the different search schemes.**

Figure 10 shows the percentage of failed queries when we use a bounded TTL equal to $\sqrt{n}$. The interesting result here is that our scheme with MH performs a lot better for rare objects (popularity smaller than 0.05%), while our scheme with a simple random walk performs better when the popularity of the object is increased.

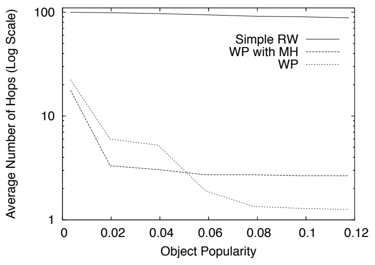

**Figure 10. Failure probability for bounded TTL messages for the different search schemes.**

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 using our approach.
- 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.
- True uniform sampling can help in the location of rare objects when compared to sampling using simple random walks.

Some practical considerations can be made about our scheme. Even though we present simulation results using sequential random walks, the scheme can be easily extended to perform the random walks in parallel and, consequently, speed up the search time. The basis of our algorithm is in uniform sampling. A uniform sample with $k$ nodes can be obtained by performing $k$ random walks of length $O(\log n)$ in parallel. Another possible optimization is in the possibility of caching the $\sqrt{n \ln n}$ pointers and initially querying the nodes in this cache. This simple optimization has the potential of solving queries in the time equivalent to a one hop lookup. These optimizations, however, add additional costs in terms of number of messages.

## 4. Related Work

Cohen and Shenker [3] improves the efficiency of search in unstructured P2P networks by replication. 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 rare objects in the network. In this sense our work complements the work of Cohen and Shenker, and other replication and caching mechanisms.

Lv *et al.* [10] shows, using simulations, that random walk is a good technique for searching unstructured P2P networks. Search and replication mechanisms are extensively studied in [10], including a realization of the square root replication policy developed by Cohen and Shenker [3], and their impact on searches for popular objects. The conclusion of this work is that a random walk approach has better performance compared to the standard approach of searching by flooding. We show experimentally that our scheme is as good as the best heuristic presented in [10].

Gkantsidis *et al.* [5] performs 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 an expander graph by performing a random walk of required length.

## 5. 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 independent of topologies and with reasonable bounds on associated overhead. Supported by elaborate experiments, we show that our approach performs well under different topologies and under dynamic scenarios that include node departures.

## Acknowledgments

This research has been partially funded by the National Science Foundation grant CNS 0509387. The first author has been partially funded by CNPq and UFMS, Brazil.

## References

[1] A. Awan, R. Ferreira, A. Grama, and S. Jagannathan. Distributed Uniform Sampling in Large Real-World Networks. Technical Report, Department of Computer Sciences, Purdue University, October 2004.

[2] 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.

[3] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of ACM SIGCOMM'02*, pages 177–190, Pittisburg, PA, August 2002.

[4] R. Ferreira, M. K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks. Technical Report, Department of Computer Sciences, Purdue University, 2005.

[5] C. Gkantsidis, M. Mihail, and A. Saberi. Random Walks in Peer-to-Peer Networks. In *Proceedings of IEEE INFOCOM'04*, Hong Kong, March 2004.

[6] Gnutella. http://gnutella.wego.com/.

[7] W. Hastings. Monte Carlo Sampling Methods Using Markov Chains and their Applications. *Biometrika*, 57(1):97–109, 1970.

[8] Limewire. `http://www.limewire.com/`.

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

[10] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Procceedings of ACM ICS'02*, pages 84–95, New York, NY, June 2002.

[11] D. Malkhi, M. Reiter, and R. Wright. Probabilistic Quorum Systems. In *Proceedings of ACM PODC'97*, pages 267–273, Santa Barbara, CA, August 1997.

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

[13] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of ACM SIGCOMM'01*, pages 247–254, San Diego, CA, August 2001.

[14] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of MMCN '02*, San Jose, CA, January 2002.

[15] 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 ACM SIGCOMM'01*, pages 149–160, San Diego, CA, August 2001.