

Probabilistic Scalable P2P Resource Location Services

Daniel A. Menascé and Lavanya Kanchanapalli¹

Abstract

Scalable resource discovery services form the core of directory and other middleware services. Scalability requirements preclude centralized solutions. The need to have directory services that are highly robust and that can scale with the number of resources and the performance of individual nodes, points to Peer-to-Peer (P2P) architectures as a promising approach. The resource location problem can be simply stated as “given a resource name, find the location of a node or nodes that manage the resource.” We call this the *deterministic* location problem. In a very large network, it is clearly not feasible to contact all nodes to locate a resource. Therefore, we modify the problem statement to “given a resource name, find with a given probability, the location of a node or nodes that manage the resource.” We call this a *probabilistic* location approach. We present a protocol that solves this problem and develop an analytical model to compute the probability that a directory entry is found, the fraction of peers involved in a search, and the average number of hops required to find a directory entry. Numerical results clearly show that the proposed approach achieves high probability of finding the entry while involving a relatively small fraction of the total number of peers. The analytical results are further validated by results obtained from an implementation of the proposed protocol in a cluster of workstations.

1 Introduction

Middleware has been defined as a collection of components such as resources and services that lie above the transport layer but below the application environment [2]. Examples of middleware services include authentication, authorization, accounting, policy management, directory, resource management, networked information discovery and retrieval services. Many middleware services are supported by directory services. A directory service associates attributes to resources (e.g., files, people, servers, printers, etc) and implements a mapping between human-readable resource names to machine readable names that are used to locate resources.

The September 2001 special issue of the Communications of the ACM refers to the future Internet as the “Invisible Internet, Always On, Always Available,” predicting that by 2006, there will be almost a billion devices interacting worldwide

on the Internet [4, 9]. Therefore, it is essential to design scalable and robust directory and resource discovery services in support of other middleware services. As indicated in [15], successful middleware solutions must be scalable in terms of number of nodes, number of resources, and the performance of individual components.

In this paper we are concerned with the design of scalable resource discovery services that form the core of directory and other middleware services. Scalability requirements preclude centralized solutions. The need to have directory services that are highly robust and that can scale with the number of resources and the performance of individual nodes, points to Peer-to-Peer (P2P) architectures as a promising approach.

The traditional client-server model, which forms the basis of the World Wide Web, underutilizes the Internet’s bandwidth and places severe burden on dedicated servers as the load on them increases [6]. P2P computing relies on the increasing computing power and storage capacity of individual computers to better utilize bandwidth and distribute the load in a self-organizing way. Nodes in a network, called *peers*, act as both clients and servers in their interactions with other peers. Peers form an application level network and are responsible for routing messages such as requests to locate a resource. The design of these routing protocols is of paramount importance to the efficiency of P2P applications. Naive approaches such as Gnutella’s flood routing can add a large amount of traffic [1, 6]. P2P computing also has the potential of enhancing reliability and fault-tolerance since it does not rely on dedicated servers [6]. It has been shown that many P2P systems that exhibit the “small-world” property—most peers have few links to others and a few peers have many connections to others—are highly robust to random attacks but highly vulnerable to targeted attacks [5, 7].

P2P relies on awareness of the surrounding area and does not require any a priori relationships between peers [6]. In this case, each peer maintains a local directory with entries to the resources it manages. A peer may also cache directory entries of other peers. Cache entries do not need to be accurate; they may be hints.

There are many important applications of P2P technologies including distributed directory systems and P2P domain names [6], new e-commerce models [3, 13], and Web service discovery [22]. All these applications require efficient resource location mechanisms.

The resource location problem can be simply stated as “given

¹Department of Computer Science, MS 4A5, E-center for E-Business, George Mason University, 4400 University Dr., {menasce,lkanchan}@cs.gmu.edu

a resource name, find the location of a node or nodes that manage the resource.” We call this the *deterministic* location problem. In a very large network, it is clearly not feasible to contact all nodes to locate a resource. Therefore, we modify the problem statement to “given a resource name, find with a given probability, the location of a node or nodes that manage the resource.” We call this a *probabilistic* location approach. We present a probabilistic resource location protocol that can trade performance and scalability for the probability P_f that a resource is located. The protocol behavior depends on probabilistic parameters, which can be set to obtain a desired value of P_f that the resource is located. The goal is to achieve a probability P_f close to one with a cost much lower than that of the deterministic case. Cost can be measured in the number of messages exchanged, used bandwidth, or the number of peers contacted. The results of an analytic model discussed here show that probabilistic resource location methods can be quite efficient.

The rest of this paper is organized as follows. Section two presents the basic assumptions. The next section discusses the probabilistic search protocol. Section four presents the analytic model and section five discusses numerical results obtained with the model. Section six presents the architecture of an implementation of the proposed protocol and shows the results of several experiments. The next section discusses related work and the last section presents some concluding remarks.

2 Basic Assumptions

Consider N nodes in a network, called peer nodes, and the following assumptions about them:

1. There is a network path between any two peers.
2. Any peer has a Local Directory (LD) that points to the resources (e.g., files, web pages, processes, devices) managed locally. Each peer has total control over its local resources and its LD. This means that resources may be added or deleted from the LD without the authorization of other peers. This assumption distinguishes our work from other work [5, 17, 19, 20, 26], that assumes that resources can be replicated and can migrate to any node at the will of the P2P system.
3. Each resource has a unique, location-independent, global identifier (GUID), which can be generated by several means. For example, Freenet uses secure SHA-1 hashes of a file content to generate GUIDs [5, 12]. In a distributed online bookstore application one could use ISBNs as GUIDs.
4. Each peer has a Directory Cache (DC) that points to the presumed location of resources managed by other peers. Entries in this cache may just be hints to the location of remote resources. An entry in the DC is a pair

(id, loc) where “id” is the GUID of a resource and “loc” the network address of a peer node that may store the resource locally. Directory entries migrate from Local Directories to Directory Caches at other nodes to adjust to access patterns.

5. Each peer s has a local *neighborhood*, $\mathcal{N}(s)$, defined as a set of peers that are close (e.g., at one hop distance or within the same local area network) to the peer. As indicated in [18], the Internet is a collection of interconnected groups of LANs under shared technical administration. Traffic that crosses the borders of these groups is more expensive than local traffic from an ISP’s point of view. The notion of neighborhood can be extended to indicate a set of peers that have business dealings with a peer as in an e-commerce setting.
6. For purposes of resource location, every peer only contacts peers in its neighborhood as well as peers indicated in the Directory Cache (DC).
7. The number of peers is very large and the network topology is unknown.

3 A Distributed P2P Directory Search Protocol

We present here a protocol that solves the following problem: “Given a resource with GUID res , locate with probability P_f , the set of peer nodes that manage the resource.”

The peer node where the search starts is called the *source*. We provide a high-level description of the protocol in Fig. 1 using the following notation:

$r : M(a, b, c)$ message M with fields a, b, c received at peer r
 $[x]_p$ execute x with probability p

The algorithm in Fig. 1 uses two messages with the following fields:

- SearchRequest (s, res, rp, TTL): request sent by source “s” to locate resource “res”. The message can only be propagated to at most TTL peers. “rp” is the sequence of peers that received this message so far. This sequence is used as a reverse path to the source.
- ResourceFound (s, res, rp, v): indicates that resource “res” being searched by source “s” was found at peer “v”. The reverse path to reach the source is “rp”.

When the SearchRequest message is received, a peer searches its local directory first and then its Directory Cache. If the resource is found in the local directory a ResourceFound message is sent along the path traversed by the SearchRequest message until it reaches the source. This message updates the

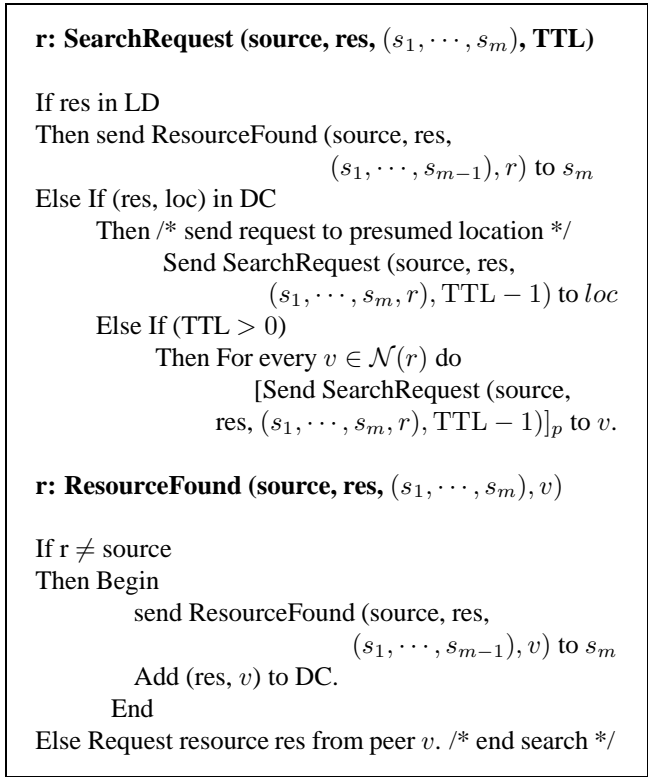


Figure 1: Distributed Resource Location Protocol

DC at every peer it visits. If the resource is found in the DC, the SearchRequest message is sent to the peer pointed by the DC. It is possible that that peer no longer has the resource, so the request will continue to be processed from that point on. If a peer does not find the resource in its LD nor in its DC, it will send the request to each peer in its neighborhood with a certain probability, called the *broadcast probability*. Note that the probability may vary with the length of the path traversed by the request.

The protocol in Fig. 1 does not address cache replacement policies in the Directory Cache nor cache invalidation options. For example, one could extend the protocol to allow for invalidation of DC entries when a SearchRequest is received as a result of finding an entry for the resource in the DC. Suppose peer *r* finds an entry for the resource in its DC pointing to peer *v*. If peer *v* does not have the resource any longer, it could indicate so to peer *r* so that it can delete this entry from its DC. Other details of the protocol are not shown. For example, we need a provision to avoid broadcasting a message more than once.

Figure 2 illustrates how the SearchRequest message would flow from the source to other peer nodes. In the figure, the maximum number of nodes to be traversed by a SearchRequest message is 4 not counting the source. Peer *c* in the figure does not propagate the request to its neighbors since it has a directory entry for the resource. Peer *d* does not have a directory entry for the resource and propagates the request to

two of its three neighbors. Peer *n* does not have a directory entry for the resource in its directory but does not propagate the request to any of its neighbors. Every peer that has an entry for the resource and receives a SearchRequest message (peers *c*, *e*, *i*, *l*, *o*, *p*) sends a ResourceFound message that will propagate up the tree to the source. As a result of this, the next request from the same source would not need to contact any other peer. A request to the same resource from peer *b* would also be satisfied locally as indicated in the new tree shown in Fig. 3. As we can see, there are directory entries for the resource cached at peers *a* through *i*, *l*, *o*, and *p*. In general, as the frequency by which resources are referenced increases, the probability of locating it at a directory closer to the source also increases. Clever cache replacement policies have to be designed. For example, one may want to reduce the caching probability for peers further away from the source of the request.

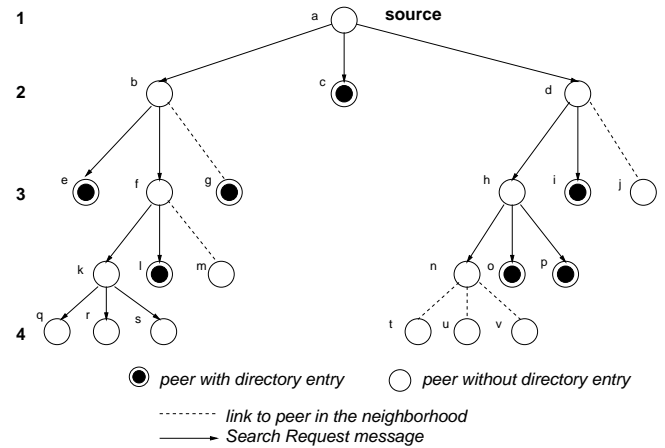


Figure 2: Example of a search tree.

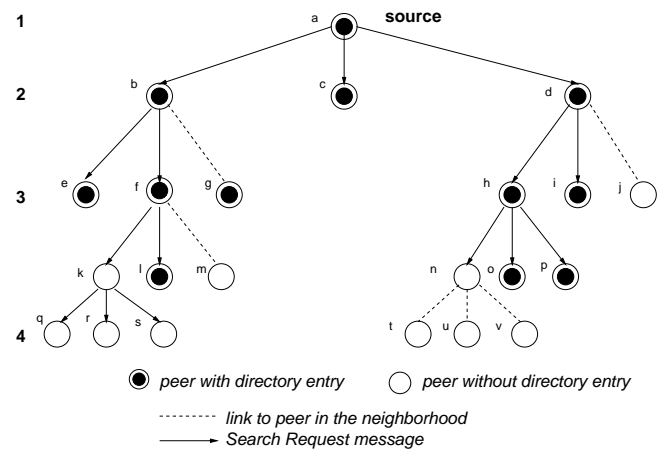


Figure 3: Example of a modified search tree.

4 An Analysis of the Probabilistic Search Method

This section presents an analysis of the efficiency of the probabilistic search method discussed previously. We start this analysis by considering that the source of a search request is at the root of a k -ary tree of height d . So, all peers have exactly k neighbors and a search request can visit at most $d - 1$ peers. We number the levels of the tree $1, \dots, d$ from the root down.

We compute first the probability distribution of the number of peers at level i that receive a SearchRequest message. To make the analysis easier to follow, we consider two cases: 1) none of the peers at levels $1, \dots, d - 1$ have an entry for the resource in their directory. So, the search request has to be propagated down to the leaves of the tree. 2) any peer from levels $2, \dots, d$ may have a directory entry for the resource.

Let q_i be the probability that a peer at level i has a directory entry for the resource. Note that $q_1 = 0$. Let $R(i)$ be the number of peers at level i that receive a SearchRequest message. $R(i) = 0, \dots, k^{i-1}$ for $i > 1$ and $R(1) = 0$.

Case 1: Directory entries for resources are at level d only (i.e., $q_i = 0$ for $i = 2, \dots, d - 1$ and $0 < q_d \leq 1$)

Let us compute the conditional probability $Pr[R(i) = n | S(i - 1) = j]$ that n peers receive SearchRequest messages at level i given that j peers at level $i - 1$ may propagate SearchRequest messages one level down. Since in this case, directory entries for the resource may be found only at level d , all peers that receive a SearchRequest message may probabilistically propagate it down. So, $S(i - 1) = R(i - 1)$.

Each of the j peers at level $i - 1$ can select from zero to k of its neighbors to send a SearchRequest message. The decision to send a message to peers at level i occurs independently for each peer with probability p_i . Let m_1, \dots, m_j be the number of children to receive the SearchRequest message from each of the j peers at level $i - 1$. Let $\mathcal{S}(j, n, k) = \{\vec{m} = (m_1, \dots, m_j) | \sum_{s=1}^j m_s = n \ \& \ 0 \leq m_s \leq k\}$ Thus,

$$Pr[R(i) = n | R(i - 1) = j] = \sum_{\forall \vec{m} \in \mathcal{S}(j, n, k)} \prod_{s=1}^j \binom{k}{m_s} \times p_i^{m_s} \cdot (1 - p_i)^{k - m_s} \quad (1)$$

Note that the value of Eq. (1) does not depend on the order that the values m_1, \dots, m_j appear in \vec{m} . So, the computation of Eq. (1) can be made more efficient by computing the term in the summation for each set of values m_1, \dots, m_j and then multiplying by the number of different vectors \vec{m} that have the same set of values. Consider for example that $j = 3$, $n = 4$, and $k = 2$. Then, $\mathcal{S}(3, 4, 2) = \{(0, 2, 2), (2, 0, 2), (2, 2, 0), (1, 1, 2), (1, 2, 1), (2, 1, 1)\}$. But, the term inside the summation is the same for $(0, 2, 2)$, $(2, 0, 2)$, and $(2, 2, 0)$. Similarly, the term inside the summation has the same value for

$(1, 1, 2), (1, 2, 1), (2, 1, 1)$. Thus,

$$Pr[R(i) = n | R(i - 1) = j] = p_i^n (1 - p_i)^{j \cdot k - n} \sum_{\vec{m} \in \mathcal{T}(j, n, k)} \beta(j, m_1, \dots, m_j) \prod_{s=1}^j \binom{k}{m_s} \quad n = 0, 1, \dots, j \cdot k \quad (2)$$

where $\mathcal{T}(j, n, k) = \{(m_1, \dots, m_j) | \sum_{s=1}^j m_s = n \ \& \ 0 \leq m_s \leq k \ \& \ m_r \leq m_s \text{ for } r < s\}$ and

$$\beta(j, m_1, \dots, m_j) = \frac{j!}{\prod_{s=1}^j \alpha(m_s)!} \quad (3)$$

where $\alpha(m_s)$ is the number of times that the value m_s appears in the sequence m_1, \dots, m_j . Using the previous example, $\mathcal{T}(3, 4, 2) = \{(0, 2, 2), (1, 1, 2)\}$, $\beta(3, 0, 2, 2) = 3$, and $\beta(3, 1, 1, 2) = 3$. An algorithm to generate the elements of the set $\mathcal{T}(j, n, k)$ is given in the Appendix.

We can now compute the probability $Pr[R(i) = n]$ that n peers receive a SearchRequest message at level i as

$$Pr[R(i) = n] = \sum_{j=0}^{k^{i-2}} Pr[R(i) = n | R(i - 1) = j] \cdot Pr[R(i - 1) = j] \quad n = 0, \dots, k^{i-1} \ \& \ i \geq 2 \quad (4)$$

Equation (4) can be computed recursively by noting that $Pr[R(1) = 1] = 1$, $Pr[R(1) = 0] = 0$, $Pr[R(i) = 0 | R(i - 1) = 0] = 1$, and $Pr[R(i) = n | R(i - 1) = 0] = 0$ for $n > 0$.

The average number of SearchRequest messages, \overline{M} , sent to all peers is

$$\overline{M} = \sum_{i=2}^d \sum_{n=0}^{k^{i-1}} n \cdot Pr[R(i) = n]. \quad (5)$$

Note that \overline{M} is also the average number, \overline{N} , of peers (not including the source) involved in processing SearchRequest messages. The value \overline{N} has to be compared with the number of peers that would have to process the SearchRequest message in the deterministic case (i.e., $p_i = 1$ for $i = 2, \dots, d$). This number is $k(1 - k^{d-1})/(1 - k)$.

The probability P_f that a directory entry for a resource is found can be computed as $1 - P_{n_f}$, where P_{n_f} is the probability that an entry for the resource is not found, which can be computed by conditioning on the number of peers at level d that receive a SearchRequest message. Thus,

$$P_f = 1 - P_{n_f} = 1 - \sum_{n=0}^{k^{d-1}} (1 - q_d)^n \cdot Pr[R(d) = n] \quad (6)$$

Case 2: Directory entries may be at any level ($0 < q_i \leq 1$ for $i = 2, \dots, d-1$)

The main difference between this case and Case 1 is that $S(i-1)$ is not necessarily equal to $R(i-1)$. So, in this case, Eq. (1), is indeed $Pr[R(i) = n \mid S(i-1) = j]$ as repeated in Eq. (7). Thus,

$$Pr[R(i) = n \mid S(i-1) = j] = \sum_{\vec{m} \in \mathcal{T}(j, n, k)} p_i^n (1 - p_i)^{j \cdot k - n} \beta(j, m_1, \dots, m_j) \prod_{s=1}^j \binom{k}{m_s} \quad (7)$$

$n = 0, \dots, j \cdot k$

We can now compute $Pr[R(i) = n]$ as

$$Pr[R(i) = n] = \sum_{j=0}^{k^{i-2}} Pr[R(i) = n \mid S(i-1) = j] \cdot Pr[S(i-1) = j]. \quad (8)$$

But, the probability $Pr[S(i-1) = j]$ can be computed by conditioning on the number r of peers that receive SearchRequest messages at level $(i-1)$. Thus,

$$Pr[S(i-1) = j] = \sum_{r=j}^{k^{i-2}} Pr[S(i-1) = j \mid R(i-1) = r] \cdot Pr[R(i-1) = r] = \sum_{r=j}^{k^{i-2}} \binom{r}{j} q_{i-1}^{r-j} (1 - q_{i-1})^j Pr[R(i-1) = r] \quad (9)$$

The equation for \bar{M} and \bar{N} is the same as in Eq. (5). The expression for the probability that a directory entry for the resource is found is given below.

$$P_f = 1 - \prod_{i=2}^d Pr[\text{directory entry not found at level } i] = 1 - \prod_{i=2}^d \sum_{n=0}^{k^{i-1}} (1 - q_i)^n \cdot Pr[R(i) = n] \quad (10)$$

We compute now the average number of hops, \bar{H} , required to find a directory entry for a resource. Even though a directory entry may be found at more than one level, we consider, for the purposes of this calculation, the minimum number of hops needed to find the entry. We start by computing the probability $P_f^{\min}(i)$ that the closest level to the source in which the entry is found is level i , given that the entry is found. This can be computed from the probability that the entry is not found at any level from 2 to $i-1$. Let $P_{nf}(s)$ be the probability

that an entry is not found at level s . With arguments similar to those of Eq. (10), we can write that

$$P_{nf}(s) = \sum_{n=0}^{k^{s-1}} (1 - q_s)^n \cdot Pr[R(s) = n] \quad (11)$$

So,

$$P_f^{\min}(i) = \frac{[\prod_{s=2}^{i-1} P_{nf}(s)] [1 - P_{nf}(i)]}{P_f}. \quad (12)$$

The average number of hops \bar{H} can then be computed as

$$\bar{H} = \sum_{i=2}^d (i-1) P_f^{\min}(i). \quad (13)$$

Another important consideration is the load handled by each peer. Suppose that each node generates search requests at a rate of λ requests/sec. In the deterministic case (i.e., $p = 1$), all peers receive requests generated by all peers. So, each peer has to be able to process $\lambda(n-1)$ requests/sec. In the probabilistic search case, each peer sees a fraction F of the requests generated by each peer. So, the overall load on each peer is $\lambda(n-1)F$. Due to the caching of directory entries the value of F tends to be very low thus reducing the processing load in each peer.

5 Numerical Results

This section presents numerical results obtained with the analytic model of Section 4. Figures 4 and 5 assume $k = 3$, $d = 5$, and $p_i = p$ for $i = 2, \dots, d$. So, the total number of peers, not including the sources, is 120.

Figure 4 shows two types of curves for the two cases analyzed before: i) directory entries only at level d ($q_5 = 1$ and $q_i = 0$ for $i = 2, 3, 4$) and ii) directory entries for resources at any level ($q_2 = 0.5$, $q_3 = 0.25$, $q_4 = 0.1$ and $q_5 = 0.01$). Higher values of q_i closer to the source can be justified by the use of caching of directory entries.

For each case, we present a curve for P_f and one for F defined as the ratio between the average number of peers, \bar{N} , involved in the search and the total number of peers not including the source. So,

$$F = \frac{\bar{N}}{\sum_{i=2}^d k^{i-1}} = \frac{\bar{N}}{(k - k^d)/(1 - k)} \quad (14)$$

As it can be seen from the figure, as the probability p increases, the probability that a directory entry for the resource is found increases and exceeds 0.9 for a value of p equal to

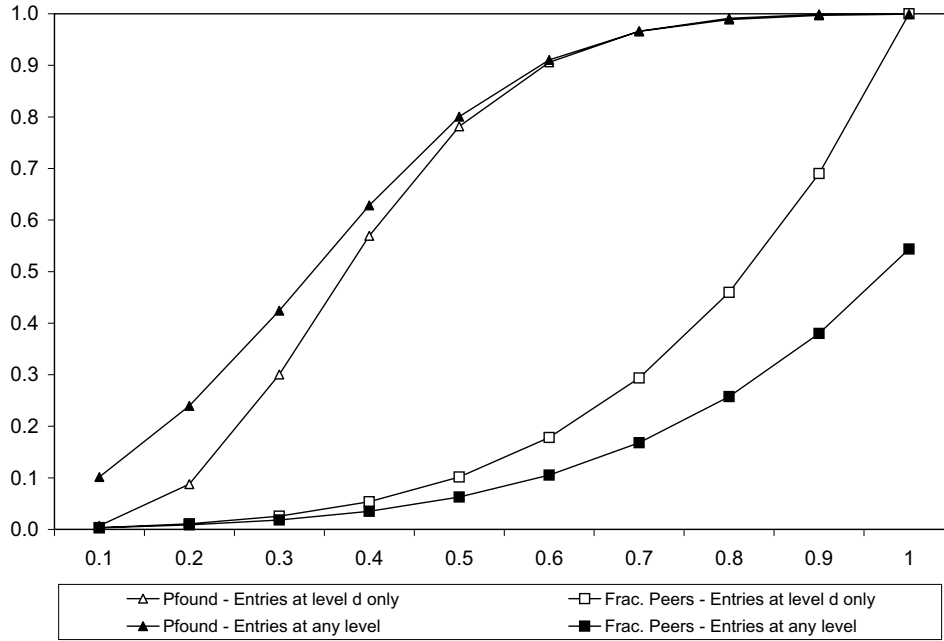


Figure 4: P_f and F vs. p

0.6. At that value, only 17.8% of the peers would have been involved in the search (i.e., $F = 0.178$) for the case in which directory entries are found at level d and 10.5% of the peers are involved in the case in which entries may be found at any level. For case ii), the entry may be found with probability 0.8 when only 6.3

For case ii), resources can be found with probability 1.0 with a participation of 54.4% of the peers. In case i) all peers would have to be involved to locate a directory entry for the resource with probability 1.

Figure 5 shows the variation of F versus P_f for cases i) and ii). For example, for case ii), a directory entry for the resource can be found with probability 0.63 when only 3.5% of the 121 peers are involved. A value of 0.989 for P_f would be achieved if 25.7% of the peers are involved.

Figure 6 shows the probability that an entry is found, the average number of peers involved, and the average number of hops required to find an entry for the following four scenarios assuming $k = 3$ and $d = 5$.

1. Deterministic case (i.e., $p_i = 1$ for $i = 2, \dots, d - 1$) and no caching (i.e., $q_i = 0$ for $i = 2, \dots, d - 1$ and $q_d = 0.01$). It is assumed that an entry can be found at all peers in level d .
2. Decreasing message broadcasting probability ($p_2 = 1, p_3 = 0.7, p_4 = 0.4$ and $p_5 = 0.3$) and no caching (i.e., $q_i = 0$ for $i = 2, \dots, d - 1$ and $q_d = 0.01$).
3. Decreasing message broadcasting probability ($p_2 = 0.7, p_3 = 0.4, p_4 = 0.3$ and $p_5 = 0.1$) and caching

(i.e., $q_2 = 0.5, q_3 = 0.25, q_4 = 0.1$, and $q_5 = 0.01$).

4. Fixed message broadcasting probability (i.e., $p_i = 0.7$ for $i = 2, \dots, d - 1$) and caching (i.e., $q_2 = 0.5, q_3 = 0.25, q_4 = 0.1$, and $q_5 = 0.01$).

When caching is used, the probability of finding the entry for the resource closer to the source increases. Therefore, the broadcasting probability p_i should decrease as one gets further away from the source. Scenarios 3 and 4 illustrate this point. As can be seen in the figure, with a decreasing probability, the probability that the entry for the resource is found is 0.813, 3.7% of the 120 nodes are involved, and when the search succeeds, the average number of hops is 1.13. If the probability is fixed, the success probability increases to 0.885, but 12.0% of the peers are involved and the average path length increases to 1.25. It may be desirable to disseminate the request to all neighbors of the source (i.e., $p_2 = 1$) and decrease the probability from that point on.

6 Experimental Validation

To validate the ideas described in previous sections, we implemented the probabilistic directory service in Java using a distributed testbed available at the Department of Computer Science at George Mason University. This testbed is a cluster of super workstations connected by a 1.2 Gbps Myrinet switch.

Each node in the cluster hosts several peers. Each peer is a thread that spawns other threads. As indicated in Fig. 7, a

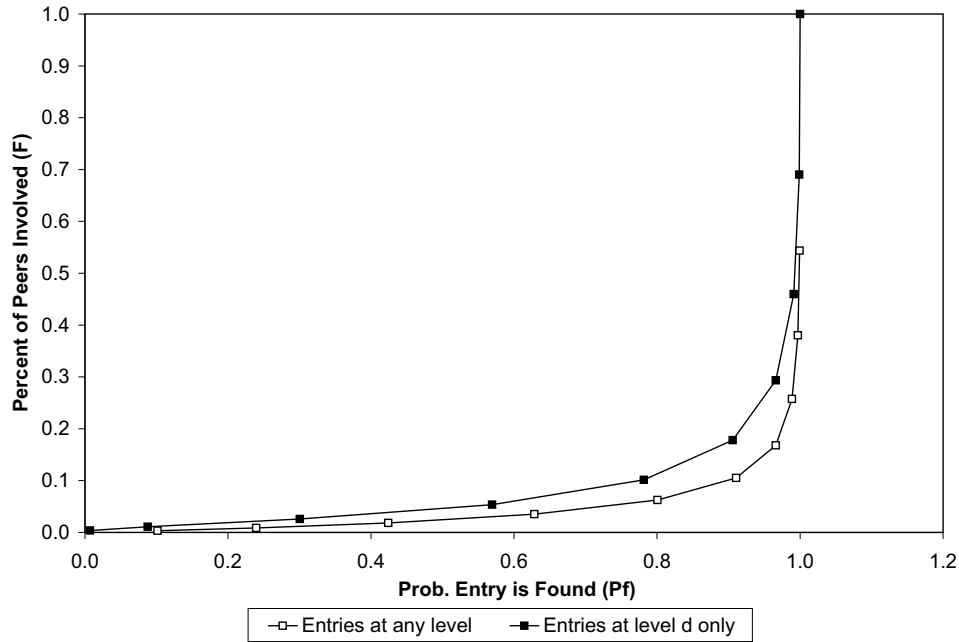


Figure 5: F vs. P_f

cluster node runs a Message Dispatcher (MD) process, several peer processes, and one External Message Router (EMR) process.

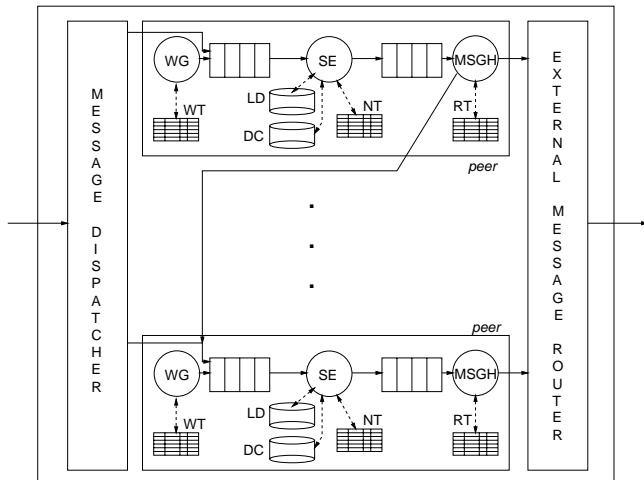


Figure 7: Processes and Threads in a Cluster Node.

The Message Dispatcher process receives UDP messages from other nodes in the cluster, determines to which peer the message is addressed, and places the message in the input queue of the Search Engine (SE) thread of the corresponding peer process.

A peer process has three main threads:

- Workload Generator (WG): generates requests to

search for resource entries in the distributed global directory. The parameters used to guide the workload generation process are stored in a Workload Table (WT) initialized before an experiment is run. These parameters determine the distribution of resources requested so that uniform and skewed distributions can be generated.

- Search Engine (SE): implements the probabilistic directory search service. The SE thread accesses the Local Directory (LD), the Directory Cache (DC), and the Neighborhood Table (NT).
- Message Handler (MSGH): uses the Routing Table (RT) to determine whether a message to another peer should be sent to a peer within the same cluster node or whether that peer resides in another cluster node. In the case of local peers, MSGH delivers the message by using local message passing. In the case of external peers, the message is delivered to the External Message Handler process.

The External Message Handler process uses UDP to communicate with the Message Dispatcher of other cluster nodes. The Workload, Neighborhood, and Routing tables are all initialized by a process called Experiment Controller, not shown in Fig. 7. We experimented with randomly generated topologies. The process used to generate the topology for a network with N peers is as follows. We start with a regular graph with k neighbors in every node. We then rewire the graph by taking each edge in turn and with a certain probability move it to connect to a randomly chosen vertex. If the resulting graph is not connected, the process is restarted. This approach can be used to generate graphs that exhibit the so-called “small-world” be-

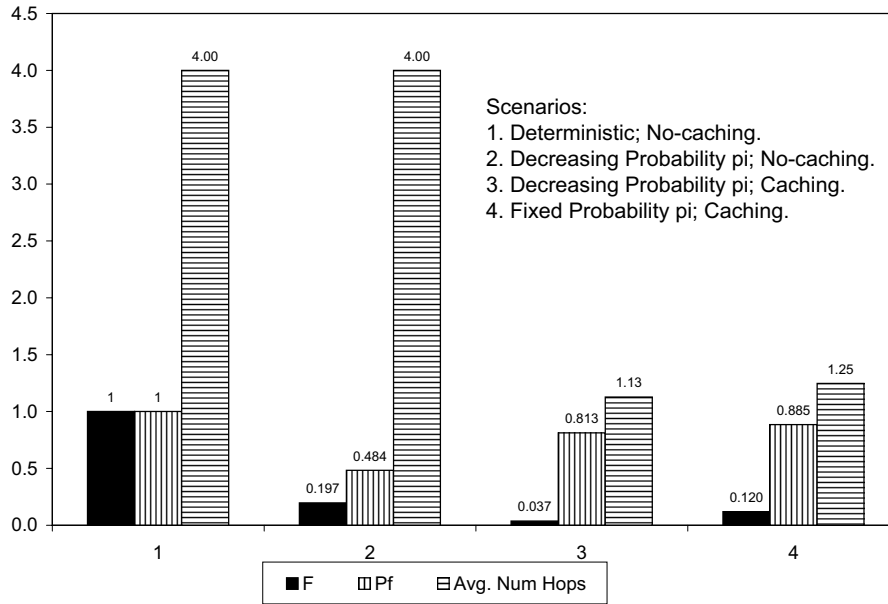


Figure 6: Various Scenarios.

havior. These networks are characterized by a power-law distribution of graph degree, i.e., $f(x) \sim x^{-t}$ where $f(x)$ is the distribution of number of routing table entries [5].

For each combination of the set of parameters, we generated ten random topologies with the same initial node degree k and rewiring probability and generated as many requests per peer in order to achieve at least a 5% precision with a 95% confidence interval. The results reported in Figs. 8 and 9 assume 30 peers, $k = 4$, rewiring probability equal to 0.1, and a cache size per peer equal to 1% of the total number of resources stored by all peers. LRU is the cache replacement policy.

Figure 8 shows two curves as a function of the broadcasting probability p . The first shows the probability, P_f , that a resource is found and the other, the cache hit ratio. The experimental results show a similar trend as the analytical results derived in the previous section, i.e., P_f achieves very high values for much smaller values of p . For example, for $p = 0.5$ in Fig. 8, $P_f = 0.84$. For the same value of p in Fig. 4, the two curves show values of P_f very close to 0.8. The cache hit ratio increases with p since more peers get involved in the search, and more than one peer will end up finding the resource. When that happens, more caches in the reverse path to the source will know about the resource, which increases the cache hit ratio.

Figure 9 shows the variation of the average number of hops, \bar{H} , needed to find a resource normalized by the total number of peers that would need to be contacted if all peers, except the source, were sent a search request message. The curve also shows the 95% confidence intervals for the measurements. It can be seen that initially the average number of hops

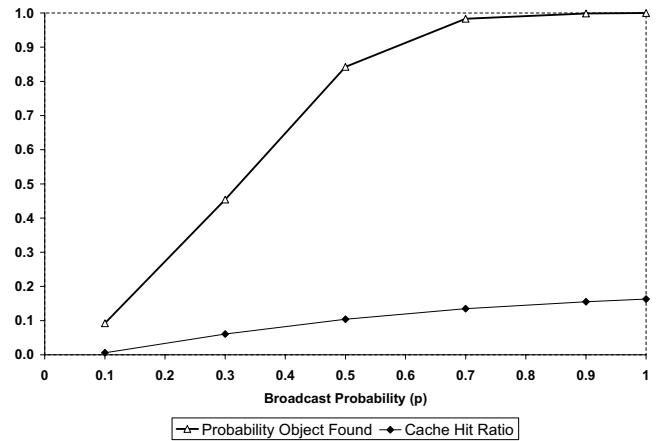


Figure 8: Experimental result for P_f vs. p .

increases because a higher broadcast probability means that peers further away from the source are contacted and the resource tends to be found at a bigger distance from the source. As p continues to increase, the increased value of the cache hit ratio, as seen in Fig. 8, allows for resources to be found closer to the source, decreasing the value of \bar{H} . Note that the maximum value of the curve in Fig. 9 occurs for $p = 0.5$. At this value, a resource is found with probability 0.84 within 3.4 hops from the source on average.

7 Related Work

Several relevant P2P systems should be mentioned here including Gnutella, Freenet, Gridella, and Sun's JXTA Search.

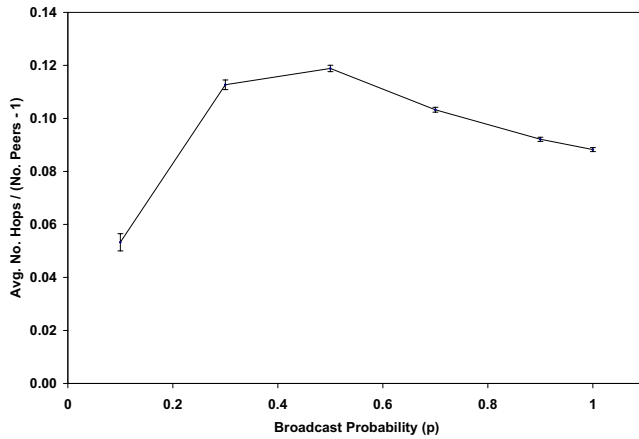


Figure 9: Experimental result for $\overline{H}/(\text{No. peers} - 1)$ vs. p .

Gnutella is an open, decentralized group membership and search protocol, used mainly for file sharing. Its goals are dynamic operability, performance and scalability, reliability, and anonymity [8, 18]. Gnutella uses flood routing to broadcast queries, which can generate a large amount of unnecessary traffic [1]. Ripeanu et al [18] have identified a mismatch between Gnutella’s overlay network topology and the Internet infrastructure. They show that this mismatch has critical performance implications.

Aberer et al [1] discuss Gridella—a Gnutella compatible P2P system based on the Peer-Grid (P-Grid). P-Grid is a virtual binary search tree that distributes replication over a community of peers. This approach is different from ours since we do not assume that resources can be replicated on other people’s machines. A P-Grid improves on Gnutella by reducing bandwidth requirements due to a superimposition of a binary tree on top of the P2P network.

Freenet is a distributed information storage system designed to address information privacy and survivability concerns [5, 12]. Freenet operates a self organizing P2P network that pools unused disk space across potentially hundreds of thousands of computers to create a collaborative virtual file system. Files managed by Freenet are stored in locations determined by their GUIDs. By using a similar procedure to insert files and to retrieve files, Freenet is able to locate the files from where they should have been stored. Freenet uses a steepest-ascent hill-climbing search: each node forwards queries to the node that it thinks is closest to the target. It has a local store with GUID and node addresses. If a file is not found in the local store, a request is sent to the node in the table with the closest key. Once a file is found it is passed upstream and possibly cached [5]. In Freenet, peers have no control over where files will be stored. Freenet makes it hard to discover which computer stores which file to address privacy concerns. Freenet, like CAN [17], Chord [20], Pastry [19], and Tapestry [26], uses hashing to store and locate objects. These systems differ from our approach since we are interested in finding resources that are managed by the peers, who have complete autonomy

over their location.

JXTA [21, 23] is a suite of protocols that facilitates P2P communication. JXTA Search is a decentralized P2P search engine. Information providers publish a description of queries they are willing to answer. JXTA Search uses specialized peers, called *hubs*, which can register with other hubs as information providers. JXTA Search is a middle ground between the decentralized Gnutella model and the centralized Napster approach. A concept similar to a hub peer is that of a super peer introduced in [24].

An approach presented in [11], called Cluster-based Architecture for P2P (CAP) systems, uses the notion of network-aware clustering. In this case, peers that are topologically close and under common administration are clustered. CAP is not totally distributed since it uses a centralized server to perform network aware clustering and cluster registration. Each cluster has a *delegate node* that acts as a directory server for the objects stored in the cluster. Krishnamurthy et al [10] modified an open source Gnutella client to passively monitor and log all Gnutella messages routed through it. The authors report on the measurements and on the positive impact of network aware clustering.

An interesting approach by Yang and Garcia-Molina [25] reduces the number of peers that process a query by using a Directed BFS technique. They evaluated their approach using Gnutella’s data. Plaxton et al. [16] present an algorithm to locate copies of replicated objects in a distributed environment. Their approach assumes that a virtual height-balanced tree T is mapped to the set of peers. Each peer u attempts to locate an object in its local memory or by contacting all peers in the subtree rooted at u in the tree T . In case of failure, the request is propagated to the parent of u in T . They assume that information about objects has to be updated on all nodes that know about them when objects are inserted or deleted. They do not assume an active caching mechanism nor a probabilistic forwarding of requests, although their approach randomly assigns labels to the peer nodes. This random assignment is used in the object location mechanism.

Finally, the reader is encouraged to read a rather comprehensive survey of Peer-to-Peer computing prepared by Milojevic et al. [14].

8 Concluding Remarks

We presented in this paper a P2P protocol to search for nodes that manage a given resource. The protocol avoids the inefficiency of flood broadcasting methods by using a probabilistic message dissemination method combined with a caching mechanism. By adjusting the probability of routing search request messages, one can vary the probability that the search is successful.

The protocol presented here lends itself to many interesting

variations related to how the values of the broadcasting probability should vary with the distance from the source of the request and regarding the caching policy.

Acknowledgements

This work was partially supported by the National Science Foundation grant number EEC-0080379 and by the sponsors of the E-Center for E-business, including webMethods.

Appendix

We present in Fig. 10 the algorithm to generate the set $\mathcal{T}(j, n, k)$. Elements of this set are built in the vector V . The functions NextMinVal and Incr are presented in Figs. 11 and 12.

References

- [1] K. Aberer, M. Puceva, M. Hauswirth, and R. Schmidt, "Improving Data Access in P2P Systems," IEEE Internet Computing, January/February 2002, pp. 58–67.
- [2] B. Aiken, J. Strassner, B. Carpenter, I. Foster, C. Lynch, J. Mambretti, R. Moore, B. Teitelbaum, "Network Policy and Services: A Report of a Workshop on Middleware," IETF Request for Comments 2768, February 2000.
- [3] J. Bond, ed., "P2P in B2B Mindshare Workshop," Netmarkets Europe, Dec. 2000, www.netmarketseurope.com
- [4] V. Cerf, "Beyond the Post-PC Internet," Communications of the ACM, vol. 44, 2001.
- [5] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley, "Protecting Free Expression Online with Freenet," IEEE Internet Computing, January/February 2002, pp. 40–49.
- [6] L. Gong, "Peer-to-Peer Networks in Action," IEEE Internet Computing, January/February 2002, pp. 37–39.
- [7] T. Hong, "Performance," chapter 14 in the book Peer-to-Peer: harnessing the power of disruptive technologies, A. Oram, ed., O'Reilly, 2001.
- [8] G. Kan, "Gnutella," chapter 8 in the book Peer-to-Peer: harnessing the power of disruptive technologies, A. Oram, ed., O'Reilly, 2001.
- [9] L. Kleinrock, "Breaking Loose," Communications of the ACM, vol. 44, 2001.
- [10] B. Krishnamurthy, J. Wang, and Y. Xie, "Early Measurements of a Cluster-based Architecture of P2P Systems," ACM Sigcomm Internet Measurements Workshop, 2001.
- [11] B. Krishnamurthy and J. Wang, "On Network-Aware Clustering of Web Clients," Proc. ACM Sigcomm, August 2001.

```

c ← 1; Computing ← True
While Computing Do
  Begin
    V [c] ← NextMinVal (c, j, k, n)
    If V [c] = -1
      Then /* unfeasible sequence */
        If c = 1
          Then Computing ← False
        Else Begin /* backtrack */
          Repeat
            c ← c - 1
            If c = 0
              Then Computing ← False
            Else V [c] ← Incr (c, j, k, n)
          Until (V [c] ≠ -1)
          c ← c + 1
        End
      Else If c = j
        Then Begin /* found feasible sequence */
          output values in V
          c ← c + 1
          If c > j
            Then Begin
              Repeat
                c ← c - 1
                If c = 0
                  Then Computing ←
                    False
                Else V [c] ←
                  Incr (c, j, k, n)
              Until (V [c] ≠ -1) or
                (¬ Computing)
              c ← c + 1
            End
          End
        End
      End
    End While
  End

```

Figure 10: Algorithm to Generate $\mathcal{T}(j, n, k)$.

```

Function NextMinVal (c, j, k, n) Returns Integer
/* Returns the next smallest feasible value at */
/* position c. If no such a value exists, return -1.*/
MinV ← 0
If c > 1 Then MinV ← V [c-1]
sum ←  $\sum_{s=1}^{c-1} V[s]$ 
For Val = MinV to k Do
  If (sum + Val + (j - c)* k) < n
    Then If (Val = k)
      Then NextV ← -1
      Else NextV ← Val
    Else If (sum + (j - c + 1) * Val) > n
      Then Begin
        NextV ← -1
        Exit For
      End
    Else Begin
      NextV ← Val
      Exit For
    End
End For
NextMinVal ← NextV
End Function

```

Figure 11: Function NextMinVal (c, j, k, n).

```

Function Incr (c, j, k, n) Returns Integer
/* Increment by 1 and return the value at position c if */
/* it is possible to increment that position. */
/* Otherwise, return -1 */
If (V [c] + 1) > k
Then Inc ← -1
Else Begin
  V [c] ← V [c] + 1
  Inc ← V [c]
  If ((j - c) * V [c] +  $\sum_{s=1}^c V[s]$ ) > n
    Then Inc ← -1
  End
Increment ← Inc
End Function

```

Figure 12: Function Incr (c, j, k, n).

[12] A. Langley, "Freenet," chapter 9 in the book Peer-to-Peer: harnessing the power of disruptive technologies, A. Oram, ed., O'Reilly, 2001.

[13] W. Meira Jr., D. A. Menascé, V. A. F. Almeida, and R. Fonseca, "E-representative: a scalability scheme for e-commerce," Second International Workshop on Advanced Issues of E-commerce and Web-Based Information Systems (WECWIS 2000), Milpitas, CA, June 8-9, 2000.

[14] D. S. Milojevic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," HP Laboratories Palo Alto, Technical Report HPL-2002-57, March 8, 2002.

[15] NSF CISE Advisory Committee, Subcommittee on the Middleware Infrastructure, "White paper on an NSF ANIR Middleware Initiative," Version 5, April 5, 2001.

[16] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," *Proc. 9th Annual ACM Symposium on Parallel Algorithms and Distributed Architectures*, Newport, Rhode Island, June 1997, pp. 311-320.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," *Proc. ACM Sigcomm*, August 2001.

[18] M. Ripeanu, A. Iamnitchi, and I Foster, "Mapping the Gnutella Network," *IEEE Internet Computing*, January/February 2002, pp. 50-57.

[19] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed, object location and routing for large-scale peer-to-peer systems," *IFIP/ACM International Conference on Distributed System Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001, pp. 329-350.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," *Proc. ACM Sigcomm*, August 2001.

[21] Sun Microsystems, JXTA Search, www.jxta.org/project/www/license.html

[22] Universal Description, Discovery, and Integration of Business for the Web, www.uddi.org

[23] S. Waterhouse, D. Doolin, G. Kan, and Y. Faybishenko, "Distributed Search in P2P Networks," *IEEE Internet Computing*, January/February 2002, pp. 68-72.

[24] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," <http://dbpubs.stanford.edu/pub/2002-13>

[25] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks," *Proc. ICDCS*, 2002.

[26] Y. Zhao, J. D. Kubiawicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," Technical Report UCB/CSD-01-1141, UC Berkeley, April 2000.