PeerSearch: Efficient Information Retrieval in Peer-to-Peer Networks

Chunqiang Tang[1], Zhichen Xu, Mallik Mahalingam
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2002-198
July 12th , 2002*

E-mail: sarrmor@cs.rochester.edu, {zhichen, mmallik}@hpl.hp.com

peer-to-peer computing, information retrieval, overlay routing, search engine

In this paper, we propose an efficient peer-to-peer information retrieval system PeerSearch that supports state-of-the-art content and semantic searches. PeerSearch avoids the scalability problem of existing systems that employ centralized indexing, index flooding, or query flooding. It also avoids the non-determinism that exhibited by heuristic-based approaches. PeerSearch achieves both efficiency and determinism through an elegant combination of index placement and query routing. Given a query, PeerSearch only needs to search a small number of nodes to identify matching documents.

* Internal Accession Date Only
Approved for External Publication
[1] Computer Science Department, Unversity of Rochester, NY 14627
© Copyright Hewlett-Packard Company 2002

PeerSearch: Efficient Information Retrieval in Peer-to-Peer Networks

Chunqiang Tang[*], Zhichen Xu [†] and Mallik Mahalingam[†]

Abstract

In this paper, we propose an efficient peer-to-peer information retrieval system PeerSearch that supports state-of-the-art content and semantic searches. PeerSearch avoids the scalability problem of existing systems that employ centralized indexing, index flooding, or query flooding. It also avoids the non-determinism that exhibited by heuristic-based approaches. PeerSearch achieves both efficiency and determinism through an elegant combination of index placement and query routing. Given a query, PeerSearch only needs to search a small number of nodes to identify matching documents.

1 Introduction

The sheer quantity of Internet content and its amazing growth rate are beyond the capability of any single search engine, such as Google. A study conducted by BrightPlanet Corporation in March 2000 estimates that the deep Web may contain almost 550 billion documents, far more than the 1.2 billion pages that Google has identified, not to mention the 600 million pages that Google is able to search [9]. Meanwhile, peer-to-peer (P2P) systems such as Napster and Gnutella are gaining popularity quickly, raising hope for building completely decentralized information retrieval (IR) systems.

Current P2P systems, however, are either unscalable or unable to provide deterministic guarantees. Usually they are based on one of the following techniques: centralized indexing, query flooding, indexing flooding, or heuristics. Centralized indexing systems such as Napster suffer from the single point of failure and performance bottleneck at the indexing server. Flooding-based techniques such as Gnutella send a query or index to every *servant* [1] in the system, consuming huge amount of network bandwidth and leading to slowdown and high variance in system response time. Heuristics-based techniques try to improve performance by direct searches to only a fraction of the population. As a result, they may fail to retrieve important documents.

[*]Computer Science Department, University of Rochester, sarrmor@cs.rochester.edu.

[†]Internet Systems and Storage Laboratory, HP Laboratories Palo Alto, {zhichen,mmallik}@hpl.hp.com.

[1]We call a computer working in P2P manner a *servant*.

Distributed hash table (DHT) systems such as CAN [6] do provide good scalability and deterministic guarantee, but they only offer a simple interface for storing and retrieving `(key, value)` pairs. Directly applying them to IR would require users to specify precise document IDs (`keys`) for retrieval, an impractical assumption in an environment where content is produced by millions of organizations and individuals, independently.

PeerSearch achieves both efficiency and determinism through an elegant combination of index placement and query routing. Given a query, PeerSearch only needs to search a small number of servants to identify matching documents. Leveraging the state-of-the-art IR algorithms such as vector space model (VSM) [1] and latent semantic indexing (LSI) [1], PeerSearch represents documents and queries as vectors and measure the similarity between a query and a document as the cosine of the angle between their vector representations. PeerSearch stores a document index in CAN using its vector representation as the coordinates, resulting in that indices stored close to each other are also close in semantics. This unifies the problem of content- or semantic-based search with routing in an overlay network.

Several features distinguish PeerSearch from other IR systems.

- PeerSearch works in a completely decentralized manner. There is no single point of failure and no complex hierarchy.
- PeerSearch supports content and semantic searches expressed in natural language, as opposed to simple keyword match.
- PeerSearch is scalable, efficient, and effective. Both indexing flooding and query flooding are avoided. The CAN routing is augmented with expressways [10], an optimization to overlay networks such as CAN. PeerSearch's effectiveness stems from the state-of-the-art IR algorithms.

2 Background

PeerSearch is built on top of CAN and expressways. Our P2P IR algorithms are extensions of VSM and LSI. Be-

fore diving into the details of PeerSearch, we first introduce these basic components.

CAN and Expressways. CAN organizes the logical space as a d-dimensional Cartesian space (a d-torus) and partitions it into *zones*. One or more servants serve(s) as owner(s) of a zone. An object `key` is a point in the space, and the object is stored at the servant that owns the zone that contains the point. Routing from a source servant to a destination servant is equivalent to routing from one zone to another in the Cartesian space. A servant join corresponds to picking a random point in the Cartesian space, routing to the zone that contains the point, and splitting the zone with its current owner(s). In addition to improving CAN's logical routing cost to $O(\log(n))$, Expressways takes only routes that closely approximate the underlying Internet topology.

Vector Space Model. The VSM represents documents and queries as vectors. Each component of the vector represents the importance of a word (*term*) in the document or query. The weight of a component is often computed using the *term frequency * inverse document frequency* (TF*IDF) scheme [1]. The intuition behind the TF*IDF scheme is that two factors decide the importance of a term in a document: how frequently the term appears in the document and how frequently the term also appears in other documents. If a term frequently appears in a document, there is a good chance that the term could be used to differentiate the document from others. However, if the term also appears in a lot of other documents, (e.g. `computer`), the importance of the term should be penalized. The VSM usually normalize vectors to the unit Euclidean norm to compensate for the difference in document length.

During a retrieval operation, the query vector is compared to all document vectors. Those closest to the query vector are considered to be similar and are returned. One common measure of similarity is the cosine of the angle between vectors.

Latent Semantic Indexing. Literal matching schemes suffer from synonymy, polysemy, and noise in documents. LSI has been proposed to address these problems. It uses singular value decomposition (SVD) to transform and truncate a matrix of *document vectors* computed from VSM to discover the semantics underlying terms and documents. Intuitively, LSI transforms a high-dimensional document vector into a medium-dimensional *semantic vector* by projecting the former into a medium-dimensional semantic subspace. The basis of the semantic subspace is computed using SVD.

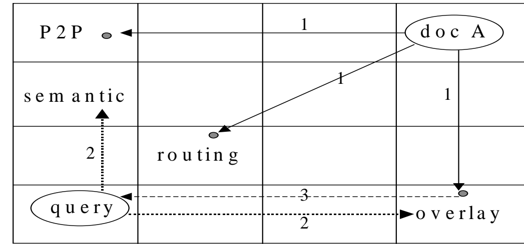

Figure 1: P-VSM in a 2-dimensional CAN. Each zone is owned by a servant. The little dots represent indices. Each servant is responsible for storing indices containing some specific keywords. Given the document A, its important keywords—`P2P`, `overlay`, and `routing`—are identified using VSM and the index is published to corresponding servants (*step 1*). Given a query of `"semantic overlay"`, it is forwarded to servants responsible for keyword `semantic` and `overlay`, respectively (*step 2*). The two servants then search and return matching indices using VSM (*step 3*).

Semantic vectors are normalized and their similarities are measured as in VSM.

In summary, both VSM and LSI represent documents and queries as vectors and the similarity between a query and a document is measured as the cosine of the angle between their vector representations. These are the only properties of the algorithms that PeerSearch relies on.

3 PeerSearch Algorithms

In this section, we describe how to extend the VSM and LSI algorithms to work with CAN.

3.1 Peer-to-Peer VSM (P-VSM)

Recent studies show that the frequency of terms in a document usually follows a *Zipf* distribution, meaning that a small number of keywords can categorize a document's content. In P-VSM, each servant is responsible for storing indices containing some specific keywords. Given a document, we use VSM to identify its important keywords automatically and publish the index of the document to servants responsible for those keywords. During a retrieval operation, the query is forwarded to servants responsible for the keywords in the query and they search and return matching indices using VSM. Figure 1 illustrates this process.

Specifically, given a document, its vector representation is computed using VSM. The m most heavy-weight vector components (terms) $t_i,\ i = 1, ..., m$ are identified, [2] and all ($h(t_i)$, `index`) pairs are stored in the

[2] The number of important terms for documents can vary on a per-document basis (refer to Section 3.5).

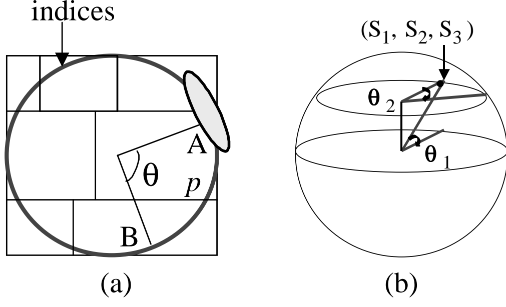

Figure 2: **(a)** In a 2-dimensional CAN, the *simplified* P-LSI only places indices on the circle, leading to an unbalanced load for servants. The similarity ($\cos\theta$) between the two indices A and B is proportional to their distance (p) on the circle: $\cos\theta = \cos p$. The gray region is the flooding area for searching indices close to A. **(b)** Using Equation 1 to transform a 3-dimentional semantic vector (s_1, s_2, s_3) into a 2-dimentional parameter vector (θ_1, θ_2).

DHT using expressway routing, where h is a hash function mapping strings into points in the CAN Cartesian space. During a retrieval operation, each term in the query is hashed into a point using $h()$ and the query is routed to servants whose zones contain the points. Each of the servants retrieves the best matching indices locally using VSM and sends them back to the servant initiating the query. The initiating servant gives them global ranks, discards those with low ranks, and presents the rest to the user.

Synonymy may cause problems for P-VSM. For example, an index may be stored under one term while retrievals use its synonyms. Fortunately, this problem is well studied and a thesaurus can be used to fix, by also routing queries to points corresponding to synonyms of the terms in the query.

3.2 Peer-to-Peer LSI (P-LSI)

Usually, CAN randomly generates document IDs and their coordinates have no meaning other than their role in routing. However, if we control the placement of the indices such that indices stored close to each other in CAN are also close in semantics, then we have a *semantic overlay* in which DHT routing is equivalent to searching in the semantic space. Using the semantic vector of a document as the `key` to store the document index in CAN achieves this goal. In the following, we first outline a simplified version of our P-LSI algorithm, and then describe how to improve it.

Simplified P-LSI. Let's use $\mathcal{L}$ and $\mathcal{K}$ to denote the LSI semantic space and the CAN Cartesian space, respectively, with l and k as the dimensionality of the two spaces, respectively. We map each document to a point in $\mathcal{K}$ by setting $l = k$ [3] and treating its semantic vector as its coordinates in $\mathcal{K}$. Given a document, its semantic vector S is computed using LSI, and the `(S, index)` pair is stored in the DHT using expressway routing. During a retrieval operation, the semantic vector Q of the query is computed and the query is routed using Q as the DHT `key`. Upon arriving at the destination, it floods the query *only* to servants within a pre-computed radius r based on the similarity threshold specified by the user. All servants that receive the query do a local search using LSI and merge the results back to the user as in P-VSM. Because indices of documents similar to the query above the threshold can only be stored within this radius r and we do an exhaustive search within this area, P-LSI achieves the same performance as LSI. Usually, this radius r is small and the involved servants are only a small fraction of the entire population.

Two problems exist in the above process. First, recall that semantic vectors are normalized and reside on the surface of the unit sphere in $\mathcal{K}$ (denote $\mathcal{U}$ as this surface), leading to an unbalanced load as depicted in Figure 2(a). Second, because document vectors are not uniformly distributed in $\mathcal{L}$, it suffers from hot spots even if servants are uniformly distributed in $\mathcal{K}$.

Full P-LSI. We solve these problems by transforming a semantic vector $S = (s_1, s_2, ..., s_l)$, $||S||_2 = 1$ in $\mathcal{L}$ into a *parameter vector* $(\theta_1, \theta_2, ..., \theta_{l-1})$ in the $(l-1)$-dimensional polar subspace. Let's denote $\mathcal{P}$ as this subspace. This transformation does exist because points on $\mathcal{U}$ only have $l - 1$ degree of freedom. Equation 1 achieves the exact goal. An example of this transformation is shown in Figure 2(b). Note that even after the transformation, parameter vectors $(\theta_1, \theta_2, ..., \theta_{l-1})$ are still not uniformly distributed in $\mathcal{P}$.

$$\theta_j = \arctan\left(\frac{s_{j+1}}{\sqrt{\sum_{i=1}^{j} s_i^2}}\right) \quad j = 1, ..., l-1 \qquad (1)$$

Several modifications to the simplified P-LSI algorithm are needed. First, we set $l - 1 = k$. Second, given a document or a query, we use the parameter vector computed from Equation 1 instead of the original semantic vector as the *key* for DHT routing. Finally, at servant join time, we randomly pick a document that the servant is going to publish and use the parameter vector of the document as the random point towards which the *join* request is routed. This bootstrap process achieves three goals. First, each servant stores roughly the same num-

[3] Both l and k are freely tunable in PeerSearch.

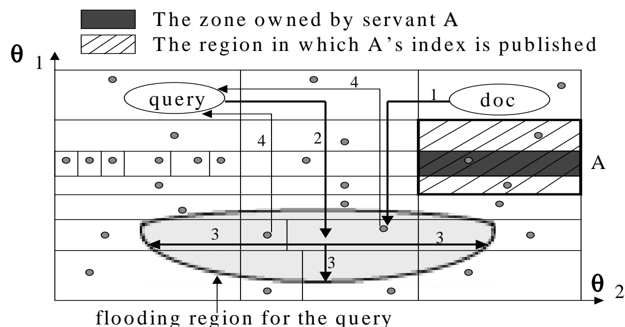

Figure 3: P-LSI in a 2-dimensional CAN. Each zone is owned by a servant. The little dots represent indices. Indices in neighboring zones are close in semantics. Given a document, its index is stored in the DHT using its parameter vector as the `key` for DHT routing (*step 1*). Given a query, it is first routed using its parameter vector as the DHT `key` (*step 2*) and then flooded to a small region computed from the given similarity threshold (*step 3*). Servants in the flooding region search and return matching indices using LSI (*step 4*). Note that indices are not uniformly distributed but the number of indices per zone is roughly the same. Because of the transformation in Equation 1, the flooding radius r is not uniform in different directions. For servant A, it is likely that its content is published in neighboring zones because of the *index locality* induced by the bootstrap process.

ber of indices (*load balancing*) because the servant distribution in $\mathcal{K}$ approximates the index distribution, given that a large number of servants exist. [4] Second, the indices stored at a servant is similar to the content published by the servant itself (*index locality*). Therefore, with certain probability, a servant stores indices for its own content, meaning that the content publishing process is extremely efficient. Third, suppose that documents owned by a user are good indications of his/her interests. Then queries submitted by the user would usually be searched in his/her neighboring zones (*query locality*), resulting in near-constant-cost DHT routing. Figure 3 pictorially depicts how P-LSI works in a 2-dimentional CAN.

The medium dimensionality of the semantic space, usually between 100 and 300, is not a problem for expressway routing, because its performance of $O(\log(n))$ is independent of the dimensionality. The size of expressway's routing table is proportional to the dimensionality, however. We expect this not to be a problem for contemporary computers. Moreover, Koll's early work on concept-based IR shows that it is possible to use only several dimensions for IR [1]. Alternatively, we can use semantic vectors of different dimensions for routing and local searching. That is, for each document or query we compute two semantic vectors, one with medium dimensionality, say 200, the other with low dimensionality, say 10. The low-dimensional vector is used for routing only. After arriving at the destination, the local searching still uses the medium-dimensional vector to avoid losing precision. In either case, we expect the dimensionality not to be a problem for P-LSI.

3.3 P-VSM Implementation Issues

P-VSM relies on some global information, such as the dictionary of the terms that TF*IDF counts in documents, and the *inverse document frequency* (IDF). We call this global information *statistics*. Fortunately, people already show that VSM does not need precise global information to work well, i.e., a good approximation is sufficient.

In P-VSM, the initial copy of the statistics are pre-computed using sample documents similar to those that will be used in the specific P2P community. Over time, a combining tree that approximates the underlying Internet topology and includes randomly chosen servants is used to sample documents and to merge statistics. The size of the statistics is largely independent of the size of the P2P community. We expect the statistics to change slowly, at the rate of weeks or even months [9], because statistics are more stable than the document itself, especially for a large servant population. The root [5] of the combining tree periodically disseminates the statistics through the combining tree, attaching a new version number to each update.

Upon receiving an update from the spanning tree, a servant sets a timer X to $2T$, where T is the estimated time that it takes to propagate an update throughout the entire combining tree. After timer X expires, the servants include the version number of the statistics in *keep-alive* messages, recursively detecting and updating neighbors with outdated versions. Upon receiving an update, every servant—whether or not it is in the spanning tree—also sets a timer Y to $P/2$, where P is the statistics update period and $P >> T$. [6] Servants keep the old statistics until Y expires. We call the time span after receiving an update and before timer Y expires a *transition period*. During the transition period, a given

[4] Without the dimension reduction technique in Equation 1, we can still achieve load balancing using a similar bootstrap process, but routing would be less efficient.

[5] The root is a servant occupying a well-known zone in CAN. If the root fails, one of its neighbors will take over the zone.

[6] Both T and P are also statistics that gradually change and are propagated to all servants.

query is first processed under the old statistics. If it fails because one of the involved servants already discards the old statistics, the query initiator is notified and a second round is started under the new statistics. Because Y is set long enough, all servants involved in a query are guaranteed to unanimously have a set of common statistics, either the old one or the new one.

In summary, timer X allows the statistics to propagate through the efficient combining tree first before the blind flooding starts; timer Y allows P-VSM to work properly during the transition period. P-VSM does need to flood the statistics occasionally, but it happens at a rate several order less frequent than the query flooding in Gnutella.

3.4 P-LSI Implementation Issues

Similarly, P-LSI also needs some global information: the dictionary, the IDF, and the basis of the semantic subspace. The basis is used to compute the projection from the document vector to the semantic vector. The solution is also similar: precompute them and update them over time using samples. However, in this case, the *transition period* needs to be handled more carefully. When the transition period starts at a servant, it computes new indices for the indices that it stores based on the new statistics, and redistributes the indices that move out of its zone. Because the statistics change gradually, the difference between the new indices and the old indices is typically small, such that most new indices still reside in the old zone and their redistribution is avoided. Meanwhile, the servant continues to process queries using old indices until timer Y expires. Then all old indices are discarded.

Computing SVD for a high-dimension matrix is an intensive process. To avoid starting from scratch at every sample update, we incrementally update the matrix using *SVD-updating*. Another solution is to replace LSI with various low-computation LSI approximations such as *Concept Index* and parallelize them, given that abundant cycles exist in P2P networks.

3.5 Performance Enhancements

Indexing at Coarse Granularity. In some cases, it is neither practical nor necessary to store an index for individual document in the DHT. People have already shown that indices on a per-database basis can work very well [4]. In PeerSearch, we use *hierarchical k-means* to cluster documents at a servant into collections until the variance inside a collection falls below a given threshold. Each collection is treated as a single document and its index is stored in the DHT. If a retrieval hits in a collection, the query is forwarded to the publisher of the collection for further search.

Relevance Feedback. In addition to the conventional use of relevance feedback to build new queries, we also exploit user retrieval patterns to improve the indice distribution. When answering a query, PeerSearch only returns a list of best matching documents. It is the user who decides which document to download. Every servant remembers the number of times that each document is downloaded and uses it as an estimation of document popularity. For a popular document, P-VSM increases the number of terms under which the document index is stored, in order to increase the chance of retrieving it with other queries. In P-LSI, upon receiving a downloading request originated from a successfully query, the servant records that the document index should move closer to the vector representation of the query. The move actually happens, when this information is accumulated over a threshold.

Load Balancing. P2P IR improves performance by allowing multiple servants to search concurrently. When the load at a servant is low, we allow it to replicate indices at its direct and indirect neighbors and to process queries on their behalf. For a servant powerful enough to replicate indices in a big radius, it is possible to process some queries at this single site. Comparing with Napster's centralized indexing, P-LSI in this case also uses a single servant for searching but limits the search to a small fraction of the entire set of documents. For hot spots in the system, CAN itself has some techniques to achieve load balancing [6]: multiple realities, peers, and so forth. Note that P-LSI already tries to balance load by approximating the index distribution in CAN with the servant distribution. The servant bootstrap process in P-VSM can be modified similarly in order to approximate term distribution with servant distribution.

4 Beyond Document Retrieval

The flexibility of PeerSearch's underlying technologies allows it to be applied to many applications other than document retrieval. We give only a few examples here.

Video/Audio. PeerSearch works by representing media content as vectors and unifying the searching problem to DHT routing. This method can be applied to any media that can be abstracted as vectors and have its object similarity measured as some kind of distance in the vector space. A lot of pattern recognition problems fall into this category. For instance, people also employ SVD to extract algebraic features from images, and use various extractors to derive frequency, amplitude, and tempo feature vectors from music data.

Semantic-Based Publish/Subscribe. Going one step beyond existing systems, PeerSearch provides a completely decentralized infrastructure for semantic-based Publish/Subscribe. The servants are natural places for keeping document subscriptions and for document availability detection. The subscription can be described not only in topics and content, but also in semantics, allowing users to subscribe unstructured documents that they do not know how to describe precisely. PeerSearch users may simply describe their needs as *"notify me when documents similar to my collection show up!"*.

Deep Search in Grid. The Data Grid and P2P model are similar in that they both deal with resource sharing and cooperation among a large number of heterogeneous and autonomous systems. To provide a uniform resource discovery and IR interface over existing heterogeneous services, we expect to use a third-party overlay as the PeerSearch infrastructure to connect existing services. Servants in the overlay maintain service indices and route queries, using the coarse-grain indexing technique described in Section 3.5. For services that cannot provide the indices needed by PeerSearch, people have already devised *query-based sampling* to extract a good summary of database contents.

Semantic-Based Resource Broker. PeerSearch essentially provides a decentralized resource broker service, in which resource providers publish a summary of their resources and consumers use DHT routing to discover the resources. Both the publishing and discovering can be expressed in either object IDs, contents, or semantics. A lot of applications can be implemented with this paradigm: P2P cooperative caching, flexible resource discovery in *Ad Hoc* networks, and so forth.

5 Related Work

Both routing indices [2] and attenuated bloom filter [7] use heuristics to selectively forward queries to a subset of neighbors that are likely to contribute in resolving the query. Reference [8] tries to organize nodes with similar content into a group. A search starts with random walk but proceeds more deterministically once it hits in a group with matching content. A study by Lv et al. [5] shows that expanding ring search and random walk are better than Gnutella's query flooding. All these systems try to improve performance by limiting searches to a fraction of the population. As a result, they may fail to retrieve important documents.

Distributed IR systems such as GlOSS [4] usually employ *a centralized* or *hierarchical* index to direct queries. Reference [3] follows the conventional database selection approach but uses a bloom filter to summarize each servant's content and *flood* them. JXTA search [9] is a query broker system built around *centralized* hubs. Currently, it does not address the problem of routing queries among hubs at a large scale.

6 Conclusion

We propose two algorithms, P-VSM and P-LSI, that combine the efficiency of DHT routing with the flexibility of content- or semantic-based searches. Our main contributions are: (1) the use of vector-space transformation to seamlessly integrate the two pieces together while retaining IR algorithms' effectiveness and DHT routing's efficiency, (2) a new angle to achieve good scalability and determinism through a combination of index placement and query routing, (3) a servant bootstrap process that achieves load balancing, index locality, and query locality, and (4) the use of semantic-based indexing to solve various problems, including deep search in Grid, resource discovery, etc. To our knowledge, PeerSearch is the first system that allows decentralized, deterministic, and non-flooding P2P information retrieval based on content and semantics, and is also the first to apply semantic knowledge to solve various network/system problems at such a deep level.

References

[1] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[2] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS*, July 2002.

[3] T. D. N. Francisco Matias Cuenca-Acuna. Text-based content search and retrieval in ad hoc p2p communities. Technical Report DCS-TR-483, Rutgers University, 2002.

[4] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2):229–264, 1999.

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

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM'01*, August 2001.

[7] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *INFOCOM'02*, 2002.

[8] M. Schwartz. A scalable, non-hierarchical resource discovery mechanism based on probabilistic protocols. Technical Report TR CU-CS-474-90, University of Colorado, 1990.

[9] S. Waterhouse. JXTA search: Distributed search for distributed networks. http://search.jxta.org/JXTAsearch.pdf.

[10] Z. Xu, M. Mahalingam, and M. Karlsson. Turning heterogeneity to an advantage in overlay routing. Technical Report HPL-2002-126, HP Laboratories Palo Alto, 2002.