A content model for evaluating peer-to-peer searching techniques

Brian F. Cooper

Center for Experimental Research in Computer Systems
College of Computing
Georgia Institute of Technology
cooperb@cc.gatech.edu

Abstract. Simulation studies are frequently used to evaluate new peer-to-peer searching techniques as well as existing techniques on new applications. Unless these studies are accurate in their modeling of queries and documents, they may not reflect how search techniques will perform in real networks, leading to incorrect conclusions about which techniques are best. We describe how to model content so that simulations produce accurate results. We present a content model for peer-to-peer networks, which consists of a tripartite graph with edges connecting queries to the documents they match, and documents to the peers they are stored at. Our model also includes a set of statistics describing how often queries match the same documents, and how often similar documents are stored at the same peer. We can construct our tripartite content model by running queries over live data stored at real Internet nodes, and simulation results show that searching techniques do indeed perform differently in simulations using this "real" content model versus a randomly generated model. We then present an algorithm for using real content gathered from a small set of peers (say, 1,000) to generate a synthetic content model for large simulated networks (say, 10,000 nodes or more). Finally, we use a synthetic model generated from World Wide Web documents and queries to compare the performance of several search algorithms that have been reported in the literature.

KEYWORDS: peer-to-peer search, simulation, modeling, performance evaluation

1 Introduction

A flurry of recent research activity has centered on peer-to-peer search networks and their applications to a variety of tasks. A consensus has emerged that initial protocols (such as Gnutella's flooding protocol) are not scalable enough, and this has spurred a great deal of interest in developing new search protocols and strategies. However, it is difficult to accurately evaluate the performance of these new techniques, since it is hard for research groups to deploy and test a real peer-to-peer network of any significant size (i.e., more than a few hundred nodes). As a result, many investigators use simulations of large peer-to-peer networks to evaluate either new techniques or existing techniques for new applications [1, 3, 5, 6, 12, 18, 16, 17, 24, 26].

Our focus is primarily on so-called “unstructured” peer-to-peer networks like those in Gnutella or Kazaa. Although “structured” networks (such as CHORD [23] and CAN [20]) have important strengths, research interest in unstructured networks remains high because of their ability to do content-based searches; see for example [6]. In unstructured networks, peers process searches over locally stored content. An overlay network is used to forward search messages between peers according to some routing protocol.

Simulations of unstructured peer-to-peer networks must model both the topology of the network and the content within the network. The topology model describes how peers are connected, while the content model describes two things: the documents that different queries match, and at which peers documents are located. In this context, a “document” is any atomic piece of content, such as a text document, music file, video file, and so on. The topology model is important, since the topology of a network determines how queries will be forwarded, and several recent techniques rely explicitly on certain topology characteristics for performance [1, 6]. However, the content model is equally important, since a simulator must be able to determine when a query reaches a peer with documents matching that query.

There are two general approaches for creating a content model for use in P2P simulations. One approach is to collect real documents and process real queries over these documents [3, 12, 24, 26]. When the simulation runs, the matching between real queries and documents is used to determine when a corresponding simulation query matches a simulation document. In the simulation, documents are assigned to nodes either randomly, or to follow the real location of the collected documents. This approach accurately captures the characteristics of real content, but it is difficult to collect very large sets of real documents. Typical studies examine tens of thousands of documents [12] or perhaps hundreds of thousands of documents [24, 26]. However, existing deployed networks may have many more documents; Kazaa for example has reported as many as hundreds of millions of documents. Techniques that work well at a relatively small scale may not work well at a much larger scale.

The second approach is to generate the content model randomly. In this approach, the matching between simulation queries and simulation documents follows some random distribution, such as uniform [1, 6] or Zipfian [5, 18, 16, 17]. The choice of which peers documents are assigned to is again random, although several existing techniques involve replicating content proactively [7]. While a random approach can be used to generate content models for simulations of very large networks, the model may not accurately reflect the distribution of queries and documents in real applications. In simulation studies described in Section 4, we found that several techniques appeared to perform significantly better using a random content model than when using a content model that matched real documents and queries. For example, a simple random walk search over a power law network topology required twice as many messages when using a real content map than when using a random map.

Our goal is to develop a content model that 1. matches query and document distributions from real applications, and 2. can be scaled up for use in simulations of very large networks. Our approach is to measure useful statistics using small but real collections, and then generate large synthetic content models that match the measured statistics. First, we model content as a tripartite graph: vertices in the graph represent queries,

documents and peers, and edges represent the documents that queries match and the peers that documents are stored at. Then, we measure two kinds of statistics about the graph: *degree* statistics, such as the number of documents that a given query matches, and *similarity* statistics, such as the number of common documents matched by two different queries. Using these statistics, we can generate a tripartite content graph of the desired size. This content graph can be used as the input to a simulation of peer-to-peer searching techniques. Our model is general enough to capture content distributions found in existing networks, such as filesharing networks, where it has been observed that a few "popular" documents are replicated widely [6]. At the same time, our model can represent content in other applications that may have different characteristics.

We have developed a simulator for peer-to-peer systems to evaluate the performance of various searching techniques. This simulator uses our content model to determine when a searching technique has found matching results. We present a case study of generating a large synthetic content model and using it with our simulator to evaluate several techniques that have been proposed in the literature. However, our content model is a general model, and can be used with any simulator to evaluate peer-to-peer systems.

In this paper, we discuss our content model and how it can be used in simulations of large peer-to-peer networks. In particular, we make the following contributions:

– We define the Map-Degree-Similarity content model (or *MDS content model*). This model includes the tripartite graph representation of queries, documents and peers (the "map"), and formal definitions for the degree and similarity statistics. (Section 2)
– We present an algorithm, SynthMap, for synthesizing large MDS content maps from degree and similarity statistics. SynthMap treats degree statistics as constraints, and uses hill climbing to form a map that best approximates the similarity statistics. (Section 3)
– We present simulation results that demonstrate the need for an accurate content model by showing the discrepancy in performance of several existing algorithms on "real" and "random" maps. We also validate our model by showing that the performance of these algorithms on a synthetic MDS content map closely matches their performance on the "real" map. (Section 4)
– We present a case study of generating a large synthetic MDS content map from a smaller real map, and using it in simulations to compare the performance of several search techniques reported in the literature. (Section 4)

We have implemented a set of tools for gathering statistics from real content and generating synthetic content models, and these tools are available to researchers who wish to use them. In Section 5 we examine related work, and in Section 6 we discuss our conclusions.

2 Map-Degree-Similarity content model

Our content model consists of two components. The *map* represents queries matching documents, and documents located at peers. The *statistics* describe the properties of the map.

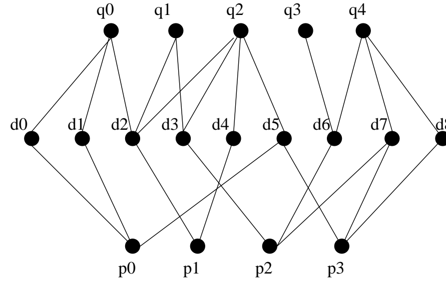

Fig. 1. Example content map.

2.1 Map

The *map* M is defined as a 5-tuple $< Q, D, P, qd, dp >$:

- Q is a set of queries
- D is a set of documents
- P is a set of peers
- qd is a set of pairs over the domain $Q \times D$, where a pair $(q_i \in Q, d_j \in D)$ represents the fact that query q_i matches document d_j
- dp is a set of pairs over the domain $D \times P$, where a pair $(d_j \in D, p_k \in P)$ represents the fact that a copy of document d_j is stored at peer p_k.

The Q, D and P sets are disjoint; that is, $Q \cap D = \emptyset$, $Q \cap P = \emptyset$ and $D \cap P = \emptyset$.

In other words, a map is a tripartite graph with vertices for queries, documents and peers. Edges from queries to documents represent the documents that each query matches, and edges from documents to peers represent the peers that each document is stored at.

The set Q contains distinct queries. For example, if the system uses keyword queries, each query $q_i \in Q$ would have a different combination of keywords. In a run of a simulation, each q_i may be submitted multiple times, possibly at different peers. Similarly, the documents in D represent distinct documents. If the same document d_j is replicated at multiple peers, this is represented by several dp edges incident on d_j.

A simple example map is shown in Figure 1. This content map represents five queries, nine documents and four peers. Query q_0 matches documents d_0, d_1 and d_2. If query q_0 reaches peer p_0, the query will find matching documents d_0 and d_1; if the query reaches peer p_1 it will find matching document d_2. Some documents in this example are located at only one peer (i.e., d_0) while some documents have copies at multiple peers (i.e., d_5).

2.2 Statistics

We define four statistics as part of the MDS model:

- *query-degree* represents the number of documents matched by each query.

– *document-degree* represents the number of peers at which each document is located.
– *query-similarity* represents the similarity between pairs of queries, defined by the number of common documents matched by each query in the pair.
– *query-peer-similarity* represents the probability that multiple documents matching a given query will be located at the same peer.

We believe that our choice of statistics captures important properties of content maps. There may be other interesting statistics that can also be defined. However, in Section 4, we present results which validate that the statistics we have chosen are useful for accurately simulating peer-to-peer search techniques.

We now formally define each of these statistics.

Query-degree counts the number of documents matched by a given query. Formally,

$$\textit{query-degree}(q_i) = |\{(q,d) \in qd | q = q_i\}|$$

where $|A|$ denotes the number of elements in set A. In Figure 1, $\textit{query-degree}(q_2) = 4$.

Document-degree counts the number of peers at which a given document is stored. Analogously to *query-degree*,

$$\textit{document-degree}(d_j) = |\{(d,p) \in dp | d = d_j\}|$$

In Figure 1, $\textit{document-degree}(d_7) = 2$.

By measuring the distribution of these statistics in addition to their average, we can capture the variance present in real networks. For example, some queries will match large numbers of documents, while others may only match one or two. Since the distribution of these statistics for a real set of queries, documents and peers may not match a smooth distribution, such as the normal or Zipfian distributions, exactly, we represent these statistics as histograms.

The next two statistics attempt to capture notions of similarity. *Query-similarity* represents the similarity between pairs of queries. Two queries are defined to be highly similar if they match many of the same documents. Query-similarity measures the property often observed in real collections that related queries will match some (though not necessarily all) of the same documents. Consider two keyword queries, “apple” and “banana.” Both queries are likely to match documents about fruit, but “apple” may also match documents about “Apple Computer” while “banana” may not. Query similarity measures the amount of overlapping documents matched by pairs of queries. The amount of query similarity can affect the performance of search protocols that route queries based on past results, such as “intelligent search” [12] or routing indices [8].

In order to formally define query-similarity, we must first define a related statistic, *query-overlap.* The query-overlap for a pair of queries q_a and q_b is the number of shared documents between q_a and q_b. Formally,

$$\textit{query-overlap}(q_a, q_b) = |\{d \in D | (q_a, d) \in qd \wedge (q_b, d) \in qd\}|$$

We could simply use query-overlap as the definition of query-similarity, but this definition is unsatisfactory since query-overlap is unnormalized. For example, if q_a and q_b both match the same single document, and q_c and q_d both match the same ten documents, $\textit{query-overlap}(q_a, q_b) = 1$ and $\textit{query-overlap}(q_c, q_d) = 10$. However, intuitively, q_a is as similar to q_b as q_c is to q_d, since for each pair, each query matches all

of the documents of the other query. We therefore normalize query-overlap to define query-similarity. To do so, we divide query-overlap by the query-degree of the queries. Since the queries might have different query-degrees, we get

$$\textit{query-similarity}(q_a, q_b) = \textit{query-overlap}(q_a, q_b) / \textit{query-degree}(q_a)$$

$$\textit{query-similarity}(q_b, q_a) = \textit{query-overlap}(q_a, q_b) / \textit{query-degree}(q_b)$$

In other words, the query-similarity statistic is not symmetric. Intuitively this makes sense; if the query-similarity statistic were symmetric then it would not capture the difference between pairs of queries with overlapping sets of matching documents and pairs of queries where one query's matching documents are a subset of the other query's matching documents. In the map in Figure 1, $\textit{query-similarity}(q_1, q_2) = 2/2 = 1$, while $\textit{query-similarity}(q_2, q_1) = 2/4 = 0.5$.

The fourth statistic, *query-peer-similarity*, captures the notion that queries are likely to find multiple matching documents at the same peer. Since users tend to collect multiple documents on each topic they are interested in, if a query matches one document at a peer it will probably match several others as well. Query-peer-similarity measures the probability of such co-occurrence of matching documents for a given query. The query-peer-similarity impacts the performance of protocols that attempt to find results by routing searches to a promising subset of peers, such as in [13, 15], since the query-peer-similarity determines the probability that multiple matching documents are indeed found in that promising subset.

The query-peer-similarity for a given query q_i is defined as the probability that if q_i matches two documents, those two documents are located at the same peer. Formally,

$$\textit{query-peer-similarity}(q_i) =$$

$$\frac{\left| \begin{array}{r} \{(d_a, d_b); d_a, d_b \in D \mid d_a \neq d_b \wedge (q_i, d_a) \in qd \wedge (q_i, d_b) \in qd \wedge \\ \exists p((d_a, p) \in dp \wedge (d_b, p) \in dp)\} \end{array} \right|}{\textit{query-degree}(q_i) \times (\textit{query-degree}(q_i) - 1)}$$

The numerator of this expression calculates the number of ordered pairs of documents (d_a, d_b) such that d_a and d_b both match q_i and are stored at the same peer. The denominator calculates the total number of ordered pairs of documents (d_c, d_d) such that d_c and d_d both match q_i. Thus, the statistic calculates the probability that an ordered pair of documents (d_a, d_b) that q_i matches is stored at the same peer. The definition is the same if we consider unordered pairs of documents; then, both the numerator and the denominator in the expression above overcount by a factor of two, and the factors of two cancel. In Figure 1, $\textit{query-peer-similarity}(q_0) = 2/6 = 0.3333$.

As with the degree statistics, we can measure the distribution of values of query-similarity and query-peer-similarity for a given map. These distributions capture the diversity of query, document and peer similarities. For example, in one map there may be groups of related queries that all match roughly the same set of documents alongside individual queries that are the only ones to match certain documents. Similarly, for some queries, the matching documents may be clustered at a few sites, while for others matching documents may be scattered all over the network.

3 Synthesizing content maps from statistics

We want to generate large synthetic content maps that we can reasonably expect will accurately model real applications. In this section, we describe how to generate a synthetic map from a set of degree and similarity statistics. Probably these statistics will be computed by analyzing a map representing real content. However, it is possible that researchers may want to generate maps that have arbitrary properties not found in an existing map. For example, a researcher may postulate that an as yet undeveloped application would use content distributed in a certain way. The researcher could generate statistics matching his assumptions and then construct a synthetic content map matching those statistics. As such, our techniques do not require the original, real content map in order to generate the synthetic map. Instead, only the degree and similarity statistics are required.

The quality of the generated map depends on the quality of the statistics. If the statistics are generated from inaccurate samples of real data, then the resulting synthetic content map will be flawed. Therefore, researchers using our model must be careful to gather an accurate sample of data before generating statistics. Similarly, a synthetic content map for a new application will produce accurate results only if the underlying statistics accurately model the new application.

First, we describe how we scale the values of the statistics to the size of the synthetic map by treating the statistic histograms as vectors and multiplying by a constant. Next, we present an iterative algorithm based on hill climbing, called SynthMap, for generating the synthetic map from the statistic vectors.

3.1 Scaling MDS statistics

Recall that the MDS content model uses four statistics: *query-degree*, *document-degree*, *query-similarity* and *query-peer-similarity*. Since the distribution of these statistics may not be smooth, we manipulate the distribution of each statistic using histograms.

The query-degree statistic histogram has one bin for each degree $0, 1...MQD_M$, where MQD_M is the maximum query-degree in map M. The count in bin i is the number of queries in M that have query-degree i. For convenience, we represent this histogram as a vector; the value in position i of the vector is the count for bin i, and the vector has $MQD_M + 1$ elements. Similarly, the histogram for the document-degree statistic has one bin for each degree $0, 1...MDD_M$, where MDD_M is the maximum document degree in M. The vector for document-degree has $MDD_M + 1$ elements, where element i is the number of documents with degree i. In our discussion, we call the query-degree histogram vector *qd-histo*, and the document-degree histogram vector *dd-histo*.

The similarity statistics query-similarity and query-peer-similarity have values in the interval $[0, 1]$. Therefore, we must choose a *bin-interval* BI that represents the width of the histogram bins for these statistics. For example, we use $BI = 0.1$, so we have bins $(0, 0.1]$, $(0.1, 0.2]$ ... $(0.9, 1]$, and the counts in each bin represent the number of items within the intervals of the bin. Note that we assume that all bins are of equal width and are contiguous. In general, histograms can be constructed with varying

intervals and non-contiguous intervals, but we do not necessarily need this generality for our purposes. However, we do find it useful to have a special "zero" bin that represents the interval $[0, 0]$. For the query-similarity statistic, the bin counts represent the number of ordered pairs of queries that have a query-similarity within the bin interval. For the query-peer-similarity statistic, the bin counts represent the number of queries whose query-peer-similarity falls within the bin interval. In our discussion, we call the query-similarity histogram vector *qs-histo*, and the query-peer-similarity histogram vector *qps-histo*.

The histogram counts depend on the size of the map from which the histograms were derived. For example, the sum of the elements of qd-histo equals the number of queries in the map. Before we can generate a synthetic map, we need to scale the histogram counts to the size of the map we are generating. We choose a scaling factor S, and the histograms will represent a map with S times as many queries, documents and peers as the map for the original histograms. We make a simplifying assumption that empty bins in the small histogram remain empty in the scaled histogram. In general, this may not be true; increasing the number of samples in a distribution may add samples to previously empty histogram buckets, especially in the tail of the distribution. However, since we do not attempt to derive a smooth distribution for our histograms, it is impossible to predict the probability that bins which were empty in the small histogram would have samples in the large histogram. Therefore, we make our simplifying assumption.

For the qd-histo, dd-histo and qps-histo, the sum of the counts is equal to the number of queries, documents, and queries (respectively) in the map. We can therefore generate scaled histograms by multiplying the histogram vectors by S. Thus, to generate histograms for a synthetic map M' from histograms representing map M:

$$\textit{qd-histo}_{M'} = S \times \textit{qd-histo}_{M}; \quad \textit{dd-histo}_{M'} = S \times \textit{dd-histo}_{M}; \quad \textit{qps-histo}_{M'} = S \times \textit{qps-histo}_{M}$$

In contrast, the qs-histo represents all ordered pairs of queries (excluding queries paired with themselves; e.g. (q_i, q_i)). Thus, the sum of counts in the qs-histo is equal to $|Q| \times (|Q|-1)$, and the scaled histogram must have counts summing to $S \times |Q| \times (S \times |Q| - 1)$. Therefore, to scale the qs-histo:

$$\textit{qs-histo}_{M'} = \frac{S \times |Q| \times (S \times |Q| - 1)}{|Q| \times (|Q| - 1)} \times \textit{qs-histo}_{M} = \frac{S \times (S \times |Q| - 1)}{|Q| - 1} \times \textit{qs-histo}_{M}$$

3.2 Generating synthetic maps

We want to generate a synthetic map $M' =< Q', D', P', qd', dp' >$ with MDS statistics matching a set of histograms. These histograms may be scaled versions of histograms from a real map M, although $S = 1$ is also possible. For example, in Section 4, we validate our model by comparing a real map and a synthetic map "scaled" with $S = 1$.

Our *SynthMap* algorithm takes as input qd-histo, dd-histo, qs-histo and qps-histo histogram vectors, and produces a synthetic map M' with MDS statistics whose distributions closely approximates these histograms. It is straightforward to generate maps that exactly match the qd-histo and dd-histo distributions by randomly generating edges. For example, to create a random query-document mapping matching a given qd-histo:

1. For each query $q_a \in Q'$, choose a non-empty qd-histo bin i

(a) Create i edges from q_a to randomly chosen documents $d \in D'$.
(b) Decrement the count in qd-histo bin i

A random document-peer matching that matches dd-histo can be constructed in a similar way. However, it is more difficult to generate maps that match the qs-histo and qps-histo distributions, since those statistics represent multiple interacting objects.

Our approach is to generate a map matching dd-histo and qd-histo exactly, and that is as close as possible to qs-histo and qps-histo. As such, we treat the problem of generating a map as an optimization problem where dd-histo and qd-histo act as constraints, and we want to maximize the similarity to qs-histo and qps-histo. The general idea of the algorithm is that we generate an initial map according to dd-histo and qd-histo, and then use hill climbing to successively improve the map until it closely matches qs-histo and qps-histo.

The SynthMap algorithm, shown in Figure 2, creates sets of queries, documents and peers. The sizes of these sets are specified by the user in the $\textit{Q-size}_{M'}$, $\textit{D-size}_{M'}$, $\textit{P-size}_{M'}$ parameters; these parameters should describe the same size map as that modeled by the histogram parameters. SynthMap then generates a query-document matching qd', using the SynthMap-QD algorithm, and a document-peer matching dp', using the SynthMap-DP algorithm.

In SynthMap-QD, we iterate, producing a series of matchings that are better and better matches to qs-histo. In each iteration, we try to replace each query-document edge with an edge that reduces the "badness" of the matching. To see if a matching qd'_x with a changed edge is better than the current qd', we calculate a histogram $\textit{qs-histo}^x$ for the query-similarity of qd'_x. We define the "badness" of a matching qd'_x as the Euclidean distance between the $\textit{qs-histo}^x$ vector and the goal $\textit{qs-histo}^{M'}$ vector. Formally:

$$\textit{badness}(\textit{qs-histo}^x, \textit{qs-histo}^{M'}) = \sqrt{\sum_{i=0}^{n-1} (\textit{qs-histo}_i^{M'} - \textit{qs-histo}_i^x)^2}$$

where n is the number of buckets in a histogram, and $\textit{qs-histo}_i^{M'}$ and $\textit{qs-histo}_i^x$ represent histogram buckets. Eventually, we will find a matching that minimizes badness, and this matching is used as the qd' for our synthetic map M'. (Since we are trying to minimize badness, our algorithm is more properly described as "gradient-descent" rather than "hill-climbing.") Ideally, we find a matching with $\textit{badness} = 0$ and the synthetic map matches $\textit{qs-histo}^{M'}$ exactly. However, if we cannot match $\textit{qs-histo}^{M'}$ exactly, we iterate SynthMap-QD until the badness is satisfactorily small; e.g., less than some *target-badness*. Note that the algorithm is not guaranteed to converge to a matching with a badness less than the *target-matching*. In particular, the algorithm might reach a local minimum in the search space whose badness is undesirably high. In practice, if the algorithm stops making progress we can terminate it early. Then we must decide whether to accept the matching it has produced, or to restart with the hope that it will find a better search space minimum. In our experiments, the algorithm produced matchings with satisfactorily small badness.

The initial matching is created by calling some function createInitialMatching(). This function could create a matching randomly from the qd-histo (as described above).

```
SynthMap(qd-histo_M', dd-histo_M', qs-histo_M', qps-histo_M', Q-size_M', D-size_M', P-size_M')
  returns M' {
    Create Q', D' and P' sets with Q-size_M', D-size_M', P-size_M' elements respectively
    qd' = SynthMap-QD(Q', D', qd-histo_M', qs-histo_M')
    ds' = SynthMap-DP(Q', D', P', qd', dd-histo_M', qps-histo_M')
    return M' =< Q', D', P', qd', ds' >
}


SynthMap-QD(Q', D', qd-histo_M', qs-histo_M')
  returns qd' {
     qd'=createInitialMatching(Q', D', qd-histo_M')

     Calculate query-similarity histogram qs-histo_qd' for qd'

     // Iterate
     while (badness(qs-histo_qd',qs-histo_M') > target-badness) {
        For each edge (q_a, d_j) ∈ qd' {
           Choose a random document d_k ∈ D', such that (q_a, d_k) ∉ qd'

           // qd'_x represents removing the edge (q_a, d_j) from qd' and replacing it with (q_a, d_k)
           qd'_x = (qd' − (q_a, d_j)) ∪ (q_a, d_k)
           Calculate query-similarity histogram qs-histo^x for qd'_x

           Choose a random number pick on the interval [0, 1)

           // If changing the edge decreases the badness, make the change
           // If changing the edge leaves the badness the same, make the change with some
           // probability pick-probability
           If [badness(qs-histo^x,qs-histo_M') < badness(qs-histo_qd',qs-histo_M')] OR
           [badness(qs-histo^x,qs-histo_M') = badness(qs-histo_qd',qs-histo_M') AND
           pick < pick-probability] {
              qd' = qd' − (q_a, d_j)
              qd' = qd' ∪ (q_a, d_k)
              Set query similiarity histogram qs-histo_qd' = qs-histo^x
           }
        }
     }

  return qd'
}
```

Note: SynthMap-DP is similar to SynthMap-QD and is omitted.

Fig. 2. SynthMap and SynthMap-QD algorithms for generating synthetic content maps.

Alternatively, any easy to create matching can be produced. For example, createInitialMatching() could just assign queries to documents by iterating through documents in round robin order. In fact, in our experience, SynthMap-QD converged more quickly in some cases by using this round robin approach rather than the random approach. Our approach in our implementation is to start by using the random initial matching, and try other easy to construct initial matchings if the rate of convergence is not satisfactory.

The SynthMap-QD algorithm always replaces an edge if the replacement reduces the badness of the map. However, sometimes it replaces an edge with another that leaves the badness unchanged. The *pick-probability* constant determines the probability that the algorithm takes such a "horizontal" move in the search space. The reason we include this possibility in the algorithm is that in our experience the algorithm frequently reaches a "plateau" in the search space. Horizontal moves allow the algorithm to move off of the plateau and resume gradient descent. Adding the horizontal moves to the algorithm results in consistently better qd' matchings.

The most expensive step in this algorithm is the repeated calculation of the query-similarity histogram qs-histo, shown in boldface in Figure 2. Constructing a qs-histo requires computing the overlap for $O(|Q'|^2)$ pairs of queries, and for large $|Q'|$, repeated calculation of this step takes a prohibitively long time. To address this problem, in our implementation we avoid the full histogram computation. Instead, we determine which pairs of queries are affected by the change, and then incrementally update the old $\textit{qs-histo}_{qd'}$ histogram by updating the affected bins to produce the new $\textit{qs-histo}_{qd'_x}$ histogram. To support this optimization, we maintain two lookup tables for qd': one that maps from a query to its matching documents, and one that maps from a document to its matching queries. We also memoize as many query overlap counts as will fit in memory, preferring query overlaps involving queries with the highest query-degree, as these queries have a high probability of being affected by changing edges. When an edge is changed, the affected memoized overlaps are updated. The combination of incremental histogram updates and overlap memoization significantly sped up our algorithm.

Next, SynthMap-DP takes the qd' query-document matching produced by SynthMap-QD, and produces a dp' document-peer matching to closely approximate $\textit{qps-histo}^{M'}$. SynthMap-DP is very similar to SynthMap-QD, and is omitted from Figure 2. SynthMap-DP iterates, generating successive dp'_x matchings, until the badness of the matching is satisfactorily small. Again, we use the Euclidean distance to measure the "badness" of the dp'_x matching, which is defined analogously to the badness for the qd'_x matching. When SynthMap-DP terminates, we have generated synthetic qd' and dp' matchings, and the process of generating M' has completed.

In our experience, an initial dp' matching that works well is to assign edges so that all of the documents are clustered on a small number of sites. This is because the maps we tried to synthesize had high query-peer-similarity, and assigning documents to a small number of sites creates a matching with higher query-peer-similarity than a random matching. As a result, the algorithm converges more quickly (and to a matching with lower badness) than when using a random initial matching.

The most expensive step in the SynthMap-DP algorithm is again the histogram calculations. In this case, the histogram calculation required comparing all pairs of documents matched by a given query to see which were co-located on the same peer. If

	Gnutella set	*Web set*
Queries	321	1,000
Documents	30,247	31,066
Peers	2,000	999
Content	Music files	HTML pages

Table 1. Content sets used in our experiments.

the average number of documents matched by a query is $|qd'|/|Q'|$, then the number of comparisons needed to calculate the histogram one time is $O(|Q'| \times (|qd'|/|Q'|)^2) = O(|qd'|^2/|Q'|)$. Again, incremental histogram updates and memoization sped up the algorithm. For each query $q_i \in Q'$, we memoize the number of co-located documents, and the total number of matching documents. When a document-peer edge is changed, the affected memoized counts are updated.

4 Experiments

In this section, we report the results of two types of experiments. First, in Section 4.3 we describe experiments we conducted to validate our MDS content model. These validation experiments measure the performance of several peer-to-peer search techniques on real content maps generated from Gnutella and Web search traces, content maps generated randomly using a uniform or Zipfian distribution, and synthetic content maps generated using our SynthMap algorithm. These validation experiments show:

– The performance of several search techniques differs widely between the real and random maps, with some techniques sending two or three times as many messages when using the real map than when using the random map. One technique's performance differed by almost a factor of 10.
– The performance of these search techniques on the synthetic content map closely matches their performance on the real map.

Thus, our validation experiments demonstrate both the need for an accurate content model and the effectiveness of the model we propose.

Second, Section 4.4 describes a case study of simulations using a synthetic content map for a 10,000 node network generated from a smaller map representing a trace of queries over 1,000 web sites. The results of this study show that in such large networks there is a tradeoff between search cost and search response time, and demonstrates the value of the MDS content model in simulating peer-to-peer search techniques.

4.1 Content sets and simulation setup

We used two different content sets for our experiments representing two different applications of peer-to-peer search. The characteristics of these content sets are summarized in Table 1. The first, called *Gnutella*, was generated from a trace gathered from the Gnutella network by Yang and Garcia-Molina in September 2001 [26][1]. This trace was

[1] We would like to thank Beverly Yang and Hector Garcia-Molina for the Gnutella trace.

gathered by running a Gnutella peer and logging queries and query responses. We used a subset of the full trace representing 2,000 randomly chosen peers. The Gnutella set represents content from the traditional peer-to-peer application, multimedia filesharing.

The second content set, called *Web*, was generated from web pages we downloaded from 1,000 web sites in March 2004. We downloaded between 1 and 2,303 pages per site by crawling the first two levels of the site. Then, we generated a set of keyword queries from the downloaded pages using term frequencies commonly observed in information retrieval systems (from [4]). We used standard information retrieval techniques to determine which queries matched which documents; namely, TF/IDF weights and the cosine distance [2]. The Web set represents an application of peer-to-peer techniques to World Wide Web information discovery and retrieval, where web sites connected in a peer-to-peer network perform searches over their own content.

For our simulations, we generated peer-to-peer search network topologies that match a power law distribution. Peer-to-peer systems are frequently simulated with power law topologies [1, 16], as a power law topology is a good (but not perfect) model of the Gnutella topology [22]. To generate the power law network, we used the PLOD algorithm [19] with an average degree of 5 and a maximum degree of 10.

We used an event-based peer-to-peer simulator that we have developed to conduct the experiments. This simulator takes as inputs a network topology, MDS content map, and list of (query,site) pairs, and generates for each pair a search message at the listed site. A message handler implementing a given search protocol (such as flooding or random walks) is registered with the simulator, and determines how search messages are routed. One time tick in our simulator represents the time for a peer to process a query and then forward it one hop in the overlay network according to the routing protocol. The simulator counts the total number of search messages generated, as well as the total search processing time (measured in ticks). Note that our MDS content model is not restricted to use with our simulator but is general enough for use with any peer-to-peer simulator.

4.2 Searching techniques

We used four different search techniques for our experiments.

Flooding is the original Gnutella search protocol. When a peer receives a search message, it both processes the message and forwards it to all of its neighbors in the overlay network. Each message is given a time-to-live value *ttl*, and search messages get flooded to every node within *ttl* hops of the source.

Iterative deepening [26] is like flooding, but search messages are sent from the source with a progressively higher *ttl* until "enough" content is found (where "enough" is defined by the user.)

Random walks avoids sending messages to all nodes [1]. When a peer receives a search message, it processes the message and then forwards it to one randomly chosen neighbor. Messages continue random walking until either a predefined number of results are found (again, predefined by the user), or a *ttl* is reached. Random walk *ttl* values are high and exist mainly to prevent searches from walking forever [16].

Biased random walks adapts random walks to leverage the power law nature of search networks [1]. Peers forward search messages to the neighbor that has the most

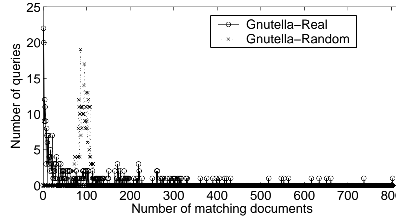

Fig. 3. Query-degree for the Gnutella maps.

overlay links. Messages are annotated with a list of where they have been, and are only forwarded to a given node once if possible. Moreover, every peer tracks the content of its neighbors. As a result, the peers with many neighbors track a large amount of content, and by seeking out those peers searches are likely to quickly discover matching content.

4.3 Validating the content model

We ran a set of experiments to validate our content model. We wanted to determine 1. if the statistic distributions differed between real and randomly generated maps, 2. if the performance of search techniques differed between real and randomly generated maps, and 3. if the performance of search techniqes on synthetic maps matched their performance on real maps. We compared several content maps:

- *Gnutella-Real*: a content map representing the real content and query traces collected from the Gnutella network.
- *Gnutella-Random*: a content map generated randomly, with the same number of queries, documents and peers as the Gnutella-Real map. Also, the average number of documents matched by queries, and the average number of copies of documents, were the same as in the Gnutella-Real map.
- *Gnutella-Zipfian*: a content map generated randomly, where the “popularity” of documents (e.g., the number of queries that match them) matched a Zipfian distribution. The number of queries, documents and peers were the same as in the Gnutella-Real map, and the average number of documents matched by queries, and the average number of copies of documents, were also the same as in the Gnutella-Real map.
- *Gnutella-Synth*: a synthetic content map generated from MDS statistics collected over the *Gnutella-Real* map.

Similarly, we compared *Real*, *Random*, *Zipfian* and *Synth* versions of the Web map.

Statistic distributions First, we asked whether the MDS statistics differed for real and random maps. The query-degree distributions for the maps are shown in Figure 3. As the figure shows, queries in the Gnutella-Random map have an average query degree clustered around the mean degree, while the distribution of the query degrees in the Gnutella-Real map is much more skewed. The random assignment of edges naturally

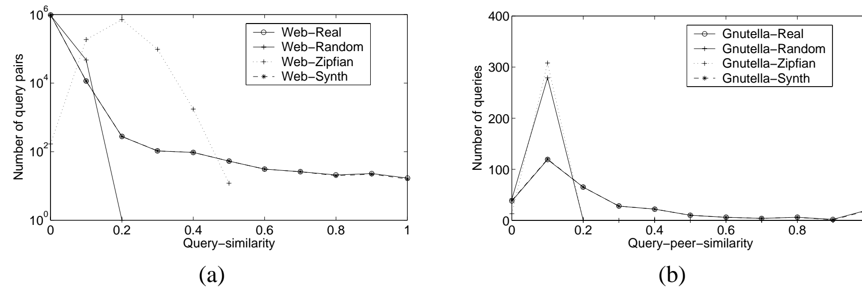

Fig. 4. Similarity statistics: (a) query-similarity for the Web maps, (b) query-peer-similarity of the Gnutella maps.

leads to a Gaussian degree distribution. Because our SynthMap algorithm generates maps with the exact query-degree distribution of the real maps, the distribution of the Gnutella-Synth map matches that of the Gnutella-Real map and is omitted from Figure 3. The query-degree distribution of the Gnutella-Zipfian map is also omitted because it closely matches that of the Gnutella-Random map: the queries matched by a document follow the Zipfian distribution but the documents matched by a query is uniformly distributed. The difference between the query-degree distributions in the Web maps are similar to that in Figure 3.

For both the Gnutella and Web traces, each found document was stored only at one site. Although it has been noted elsewhere [6] that content is often replicated at multiple sites, we did not observe such replication in our traces. Therefore, the document-degree distribution was identical in the real, random, Zipfian and synthetic maps for both the Gnutella and Web data.

The distribution of query-similarity values for the Web-Real, Web-Random and Web-Zipfian maps is shown in Figure 4(a). This figure also shows, for comparison, the query-similarity distribution in our generated Web-Synth map, which almost completely overlaps that of the Web-Real distribution. Note that the vertical axis in the figure has a logarithmic scale. As the figure shows, most pairs of queries in the random map have little or no query-similarity. In other words, if a query matches a document, it is unlikely that any other queries will also match the document. In the Zipfian map, there is more query-similarity due to the large popularity of some documents, with most pairs of queries having a similarity around 0.2. In contrast, in the real map, the query-similarity distribution is more spread out, with several pairs of queries having similarity as high as 1. The figure also shows that our SynthMap algorithm is able to produce a synthetic map with a query-similarity closely matching that of the real map. The results are similar for the Gnutella maps, except that the query-similarity of the Gnutella-Real and Gnutella-Synth maps is somewhat closer to that of the Gnutella-Random map.

The query-peer-similarity distributions are also different between the real, random and Zipfian maps. The distributions for the Gnutella maps are shown in Figure 4(b); the distributions for the Web maps are similar. As the figure shows, the real map has more query-peer-similarity than either the random map or the Zipfian map, since real

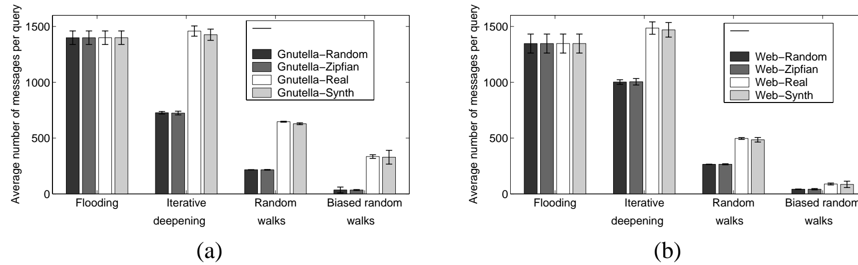

Fig. 5. Cost of search techniques on various maps: (a) Gnutella set, (b) Web set.

sites tend to store similar documents, while the random and Zipfian maps scatter similar documents all over the network. The figure also shows that the query-peer-similarity of the synthetic map closely matches that of the real map (in fact, overlapping it in the figure), again showing the effectiveness of our algorithm at generating synthetic maps that match real statistics.

Query performance Next, we examined whether the performance of search techniques on the random, Zipfian and synth maps matched their performance on the real map. In these experiments, we generated 50 search network topologies. For each topology, we injected 10,000 randomly chosen queries at randomly generated source peers, and we measured the average number of messages required per query for each search technique as well as the average time for each query to complete. The goal for each search technique was to find 10 results. For flooding and iterative deepening, we set the *ttl* to 5, and for random and biased random walks we set the *ttl* to 1,000. We also conducted simulations where we varied these parameters; while the absolute results differed, we reached the same conclusions as those reported below.

First, the number of messages sent per query is shown in Figure 5(a) for the Gnutella maps, and Figure 5(b) for the Web maps. In both figures, the error bars represent 95 percent confidence intervals. As the figures show, the performance of search techniques on the random and Zipfian maps (dark gray bars) differs from the performance on the real map (white bars), sometimes radically. For example, the difference in performance between the random and real Gnutella maps is a factor of two for iterative deepening, a factor of three for random walks and a factor of 9.6 for biased random walks. The exception is flooding, which always sends the same number of messages from a given source node regardless of the query or results. The search techniques also require more messages on the real Web map than on the random or Zipfian Web maps, by a factor of 1.5 or more. The reason that the techniques appear to perform better on the random map is that content is scattered uniformly throughout the network, making it easier to locate. In the real map, content matching a query is concentrated at a few sites, which can pose problems for random walks and biased random walks in particular. If the query misses those few sites along its walk then it may walk for a long time without finding content. Although we might think that the Zipfian maps would do a better job of capturing the skewed distribution of real content, our results show that Zipfian maps do not produce

performance results comparable to the real maps. The reason is clear from Figure 4; the Zipfian maps do not accurately model the distribution of queries, documents and peers in the real content sets.

Figure 5 illustrates the importance of an accurate content map. For example, with our parameter settings, it appears with the random or Zipfian map that iterative deepening is significantly better than flooding on both the Gnutella and Web data sets. However, on the real map the performance of iterative deepening is worse than flooding. In the real map (where content may be clustered at sites far from the source node), iterative deepening often ends up sending queries with $ttl = 5$, which costs the same as flooding. However, the extra messages sent under iterative deepening for $ttl < 5$ mean that the total messages is higher under iterative deepening than under flooding. Clearly, it is important to have an accurate content map so that we avoid such incorrect conclusions.

In contrast, Figure 5 shows that the search techniques perform comparably using the real map (white bars) and synthetic map (light gray bars). Because the synthetic content map accurately models the distribution of content in the network, it provides a better framework for evaluating the performance of the different techniques.

Next, we calculated the time required to process each query. Recall that a time tick represents the time for one node to process a query and forward it one hop in the overlay network, and we measured the time for a query in terms of the number of simulation time ticks before all results reached the source. The results (not shown) are similar to the results for the number of messages: the search techniques require less time on the random map or Zipfian than they do on the real map, by a factor as high as 10, while the search techniques require the same amount of time in the real and synth maps.

All of these results demonstrate the usefulness of the MDS content model for accurately simulating the performance of peer-to-peer search techniques.

4.4 Case study: Evaluating the performance of search strategies

We now describe a case study of using a synthetic content map to examine existing techniques for a new application: peer-to-peer web search, where web sites themselves process and route web searches. This application was suggested by Li et al [14] who proposed using structured DHT networks to perform search. Our study considers unstructured search techniques. There are a number of motivations for such an application. First, the web is quite large and growing, and a peer-to-peer architecture has the potential to be far more scalable than a centralized search engine. Second, long crawling times cause search engines to be out of date, while a peer-to-peer search system can locate the most up to date content. Third, decentralization puts search back in the hands of web site owners, who resent the power of large search engines like Google.

We used the HTML pages from the Web set, as described in Section 4.1. We generated a synthetic 10,000 peer content map using the MDS statistics from the real Web content map. Certainly, this is a relatively small map when compared to the actual size of the Web. However, it represents a useful first step in studying this application, as we can rule out techniques that do not even scale to 10,000 sites. In ongoing work we are constructing even larger maps for simulation.

In this experiment, our goal was to find 10 web pages matching a query, roughly equivalent to the first page of query results. The search mechanism can continue running

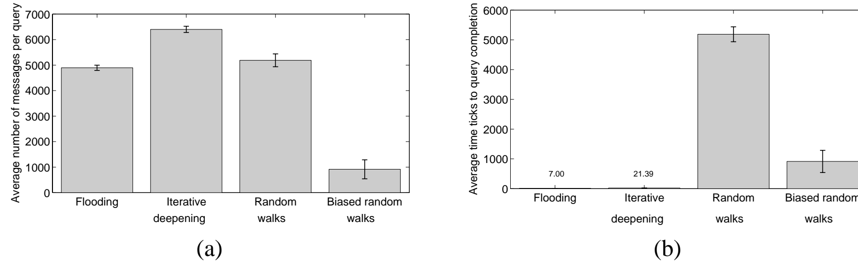

Fig. 6. Web search case study: (a) messages sent, (b) query processing time.

if the user wants more results. Because of the size of the network, we set the *ttl* for flooding and iterative deepening to 6, and the *ttl* for random walks and biased random walks to 20,000. We also set the first iteration of iterative deepening to $ttl = 3$. By using an initial *ttl* larger than one, we can reduce the total number of messages [26].

The total number of messages for each technique is shown in Figure 6(a), with 95 percent confidence intervals. As the figure shows, the best performance was achieved by biased random walks, which required 915 messages per search on average. Surprisingly, flooding is the second best technique, requiring 4,895 messages per search, a factor of 5.3 worse than biased random walks. We might expect random walks to perform better than flooding, but it does not, requiring 5,188 messages per query. Analysis of the data reveals that random walks perform well for queries that match many documents but perform poorly for queries that match few documents. This variance is reflected in the large confidence interval. As before, iterative deepening performed worse than flooding, despite the optimization of starting with $ttl = 3$.

However, flooding is far superior to biased random walks when we consider the time to process queries, which is shown in Figure 6(b). (In this figure, the bars for flooding and iterative deepening are so small we print the number of time ticks above where the bars would be.) The flooding technique required only 7 time ticks to complete (one tick to submit the query and six ticks to forward it six hops) but the biased random walks required 915 time ticks. This is because flooding allows many peers to process the searches in parallel, while biased random walks requires queries to be processed sequentially, one peer after another. As a result, for an improvement of a factor of 5.3 in message cost, biased random walks causes two orders of magnitude degradation in response time over flooding. Very long response times may be unacceptable to users accustomed to quick response times from Google. It has been suggested to use parallel random walks to improve response time [16]. However, if we simulate multiple parallel walks, the response time does not approach that of flooding unless we create so many walks that the message cost becomes prohibitive. We might proactively replicate content to improve the cost and response time of walks [7]; but unless a large number of web site owners are willing to store mirrors of each other's content proactive replication will not be effective. Overall, we may not believe that flooding is scalable enough, but we can conclude from simulations using the MDS model that random and biased random walks are even less scalable for this application due to the long response time.

5 Related work

Simulation is a common method of evaluating peer-to-peer search techniques. Recent work that uses simulations includes [1, 3, 5, 6, 12, 18, 16, 17, 24, 26]. Our tripartite graph model is a formalization of the content models used in these various studies. These simulation studies use either real [3, 12, 24, 26] or random [1, 5, 6, 18, 16, 17] content maps. Real maps require extensive effort to gather a large trace from a real system, and cannot easily scale to very large network sizes. Random maps can be made arbitrarily large but suffer from the inaccuracies discussed in Section 4.

There are several interesting traces of real peer-to-peer systems available, including [11, 21, 22, 26]. These traces can be used to inform accurate simulations. Similarly, topology models have been well studied [9, 25], as have models for download traffic [11] and peer behavior [10]. Content models for peer-to-peer networks, to our knowledge, have received less treatment. Chawathe et al [6] discuss the properties of Gnutella content but do not derive a general model as we do.

6 Conclusions

Accurate simulations of peer-to-peer techniques require both a topology model and a content model. We have presented the MDS model, a general model of content in peer-to-peer systems which allows researchers to simulate large networks whose content shares the characteristics of content in real networks. The MDS model represents queries, documents and peers as a tripartite graph, with edges representing documents matching queries and documents stored at peers. A set of degree and similarity statistics describes the properties of the tripartite graph. Our approach is that researchers can generate a large synthetic tripartite graph that has statistics matching a smaller graph generated from real data. To this end, we present an algorithm, SynthMap, that takes a set of statistics and produces a synthetic map matching those statistics.

In a set of simulation experiments we demonstrate two conclusions. First, simply generating a random content map results in inaccurate simulation results. Second, generating a synthetic map using our techniques produces results that closely match the results obtained using a real map. This validation study shows that the MDS model is an effective one for peer-to-peer simulations. We also present a case study of peer-to-peer web search that uses a 10,000 peer synthetic content model produced using SynthMap. As this study shows, the MDS content model is a useful tool for evaluating the performance of peer-to-peer search techniques.

References

1. L. Adamic, R. Lukose, A. Puniyani, and B. Huberman. Search in power-law networks. *Phys. Rev. E*, 64:46135–46143, 2001.
2. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, N.Y., 1999.
3. B. Bhattacharjee. Efficient peer-to-peer searches using result-caching. In *Proc. IPTPS*, 2003.

4. B. Cahoon, K. S. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Transactions on Information Systems*, 18(1):1–43, January 2000.
5. A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proc. SIGCOMM*, 2003.
6. Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proc. ACM SIGCOMM*, 2003.
7. E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. SIGCOMM*, August 2002.
8. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.
9. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. SIGCOMM*, 1999.
10. Z. Ge, D.R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proc. INFOCOM*, 2003.
11. K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. In *Proc. SOSP*, 2003.
12. V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proc. CIKM*, 2002.
13. M. Khambatti, K. Ryu, and P. Dasgupta. Structuring peer-to-peer networks using interest-based communities. In *Proc. International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 2003.
14. J. Li, B.T. Loo, J.M. Hellerstein, M.F. Kaashoek, D.R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *Proc. IPTPS*, 2003.
15. A. Loser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within peer-to-peer networks. In *Proc. International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 2003.
16. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proc. of ACM Int'l Conf. on Supercomputing (ICS'02)*, June 2002.
17. Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make Gnutella scalable? In *Proc. of the 1st Int'l Workshop on Peer to Peer Systems (IPTPS)*, March 2002.
18. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proc. WWW*, 2003.
19. C. Palmer and J. Steffan. Generating network topologies that obey power laws. In *Proc. of GLOBECOM 2000*, Nov. 2000.
20. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. SIGCOMM*, Aug. 2001.
21. M. Ripeanu and I. Foster. Mapping the Gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *Proc. of the 1st Int'l Workshop on Peer to Peer Systems (IPTPS)*, March 2002.
22. S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. Multimedia Conferencing and Networking*, Jan. 2002.
23. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM*, Aug. 2001.
24. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. SIGCOMM*, 2003.
25. H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *Proc. SIGCOMM*, Aug. 2002.
26. B. Yang and H. Garcia-Molina. Efficient search in peer-to-peer networks. In *Proc. Int'l Conf. on Distributed Computing Systems (ICDCS)*, July 2002.