PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities

Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, Thu D. Nguyen
{*mcuenca, peery, rmartin, tdnguyen*}*@cs.rutgers.edu*
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854

Abstract. *We introduce the PlanetP system, which explores the construction of a content addressable publish/subscribe service using gossiping between peers of an unstructured peer-to-peer (P2P) community. Unlike many recent P2P systems that have focused on enabling very large-scale name-based object location, PlanetP does not build and maintain a sophisticated distributed data structure. Instead, PlanetP achieves its goals using: (1) a gossiping layer that allows peers to replicate globally a membership directory and an extremely compact content index, and (2) an approximation of a state-of-the-art text-based, per-query content search and ranking algorithm that helps users find the most relevant information. We argue that PlanetP is a simple, yet powerful system for sharing information. PlanetP is simple because each peer must only agree to perform a periodic, randomized, point-to-point message exchange with other peers. PlanetP is powerful because it maintains a global, coherent, and content-ranked view of the shared data. Using simulation and a prototype implementation, we show that PlanetP achieves ranking accuracy that is comparable to a centralized solution and scales easily to several thousand peers.*

Keywords. peer-to-peer, P2P, gossiping, information retrieval, content addressing, content search, content ranking.

1 Introduction

Peer-to-peer (P2P) computing, where peers in a community interact *directly* with each other rather than through intermediary servers, is emerging as a powerful paradigm for collaboration over the Internet. The advantages of this paradigm include: (a) the ability to leave shared, but distributed, data at their origin, rather than incurring the privacy and safety concerns of collecting and maintaining them in a centralized repository, (b) low cost of entry and ease of incremental scalability, and (c) amortization of computing, storage, bandwidth and administration costs over the entire community as opposed to aggregating these at a central location.

Recent peer-to-peer (P2P) systems have typically focused on realizing P2P computing's promise for massive scalability. To meet their scaling targets of millions, or even billions, of users [15], these systems construct and maintain large *distributed data structures*, e.g., a hash table, across the entire community to support *name-based object location* [17, 26, 19, 22]. A significant challenge for these systems is maintaining the correctness and consistency of their global data structures as peers independently and unpredictably join and leave the on-line community [16]. Further, these mapping schemes must load balance across peers with potentially widely different resources.

In this paper, we propose a novel alternative approach of constructing a content addressable publish/subscribe service by using gossiping [5] to replicate global state across *unstructured* communities of several thousand, perhaps up to ten thousand, peers. We believe that this target scale will be applicable to many information sharing communities such as groups of researchers sharing publications, doctors sharing medical analyses, and students sharing music and videos. Further, the success of the Internet search engines argue that content addressing is a natural and powerful model for locating information in large data collections.

We have realized our proposed approach in a prototype system called PlanetP. Members of a PlanetP community *publish* documents to PlanetP when they wish to share information. Typically, the publication of a document takes the form of giving PlanetP a small XML snippet containing a pointer to a local file, which is *left in place* by PlanetP. PlanetP indexes the XML snippet and the file (if it can) in a way that peers across the entire community can locate that file based on its content. We argue that PlanetP is a simple, yet powerful system for sharing information. PlanetP is simple because each peer must only agree to perform a periodic, randomized, point-to-point message exchange with other peers, rather than collaborate to correctly and consistently maintain a global data structure. PlanetP is powerful because it maintains a view similar to an Internet search engine: a global, coherent, and content-ranked data collection without depending on centralized resources and the online presence of specific peers.

PlanetP has two major components: (1) an infrastructural gossiping layer to support the replication of shared data structures across groups of peers, and (2) an approximation of a state-of-the-art text-based search and rank

algorithm [1]. The latter requires two small data structures to be replicated globally: a membership directory and a content index. The directory contains the names and addresses of all current members. In the interest of minimizing updates and per-peer storage, the global index contains only term-to-peer mappings, where the mapping $t \rightarrow p$ is in the index if and only if peer p has published one or more documents containing term t. All members agree to continually gossip about changes to keep these shared data structures updated and loosely consistent. We explicitly choose gossiping because the randomness built into gossiping is well suited to the P2P environment, where peers may come and go freely. Further, gossiping is never deterred by the abrupt disappearance or absence of any subset of peers.

In this paper, we address several questions, including:

- How effective is PlanetP's content search and rank algorithm given that it maintains a global index that contains even less information than previous related efforts [10, 4]?
- Can PlanetP maintain a usable level of consistency for shared data structures given the randomness inherent in gossiping? That is, when a change occurs, how long does it take to reach all on-line peers and does it consistently reach all on-line peers?
- Are PlanetP's bandwidth and storage requirements consistent with the constraints of typical P2P communities?

We use simulation and measurements from our prototype implementation to answer these questions, showing that PlanetP achieves search and rank accuracy that is comparable to a centralized solution and scales easily to several thousand peers. (We intend to scale our prototype to roughly ten thousand peers although we discuss several ideas for scaling PlanetP well beyond this level in Section 6.) The globally replicated databases only require a modest amount of storage. Changes to replicated data consistently reach all on-line members within several minutes. Further, synchronizing these databases using our gossiping algorithm require only a modest amount of bandwidth, even for extremely dynamic communities with very high rates of change.

2 Gossiping

At the heart of PlanetP is its gossiping algorithm: PlanetP uses gossiping to replicate shared data structures across the entire community. PlanetP's gossiping algorithm is a novel combination of a push *rumor mongering* and a pull *anti-entropy* algorithm as previously introduced by Demers et al. [5] together with a *partial anti-entropy* algorithm that we have found improves performance significantly for dynamic P2P environments. Rumor mongering works as follows. Suppose that a peer x learns of a change to a data structure that is replicated everywhere. Every T_g seconds, x randomly chooses a target peer y from its directory—a data structure that includes information about all peers in the community and is itself replicated everywhere using gossiping—and attempts to tell y of the change; this is a rumor. y records the rumor and starts to further spread this rumor if it has not heard of it before. If x contacts n peers in a row that already knows about the change, it stops spreading the rumor.

Rumor mongering is very efficient at spreading new information. However, it may not reach a residual set of peers before dying out. Thus, to reliably spread a new piece of information everywhere, Demers et al. suggested combining it with anti-entropy; that is, every so many rounds, say 10, each peer performs a pull anti-entropy operation instead of rumor mongering [5]. A peer also performs anti-entropy if it does not have any rumor to spread. An anti-entropy message asks the target y to send a summary of its entire data structure to x. When x gets y's summary, x can ask y for any new information that it does not have.

While Demers et al. found the combined algorithm to work quite well in their environment, in a dynamic P2P environment, where new rumors are constantly arriving, the time required to spread a particular rumor everywhere can become highly variable because anti-entropy operations are forced to their slowest rate. We can fix this by increasing the rate of anti-entropy. This would be an expensive solution, however, because anti-entropy is much more bandwidth expensive than rumor mongering as a consequence of exchanging summaries of the shared data structure.

Thus, we instead extend each rumoring round with a partial anti-entropy exchange that works as follows. When x sends a rumor to y, y piggybacks the ids of a small number m of the most recent rumors that y learned about but is no longer actively spreading. If x is missing any recent rumors, it pulls them from y. This partial anti-entropy requires only one extra message *in the case that y knows something that x does not*. Further, the amount of data piggybacked on y's message is very small, in order of tens of bytes.

Finally, PlanetP dynamically adjusts its gossiping interval T_g. If a peer is not actively rumor mongering, it slowly raises its T_g to reduce bandwidth consumption. When it receives a new rumor, it immediately resets its gossiping interval to the default in order to efficiently spread this new information. This dynamic adaptation of the gossiping interval has two advantages. First, we do not need to define a termination condition given the prob-

[1] Most multi-media files such as MP3 and AVI files have embedded descriptive text these days. Thus, our text-based search and rank algorithm is not limited only to the sharing of text documents.

abilistic nature of the algorithm. Second, when global consistency has been achieved, the bandwidth use is negligible after a short time.

3 Content Search and Retrieval

As already mentioned, a peer publishes a document to PlanetP when it wishes to share that document. PlanetP leaves the shared document in place but runs a simple web server to allow retrieval of the file by remote peers when desired. PlanetP indexes each published document, maintaining a detailed inverted index describing all documents published by a peer *local* to that peer. In addition, PlanetP uses gossiping to replicate a term-to-peer index everywhere for communal search and retrieval. This term-to-peer index contains a mapping "$t \rightarrow p$" if and only if term t is in the local index of peer p.

To find documents relevant to a query, a searching peer would first derive the set of peers that have the desired terms. Then, it forwards the query to these peers and asks them to return URLs for any documents that are relevant to the query. PlanetP uses this two-stage search process to limit the size of the globally replicated index. (For the remainder of the paper, we will refer to the globally replicated index as the global index and the more detailed inverted index that describes *only* the documents published by a peer maintained locally at that peer as the local index.)

In addition to the exhaustive search supported by the indexes, PlanetP also implements a selective search algorithm that uses the vector space ranking model [25] to choose the subset of documents most relevant to a query. In the remainder of this section, we describe how we have adapted a state-of-the-art vector ranking algorithm to use PlanetP's two-level indexing scheme.

3.1 Background: The Vector Space Ranking Model and TFxIDF

In a vector space ranking model, each document and query is abstractly represented as a vector, where each dimension is associated with a distinct term; the space would have k dimensions if there were k possible distinct terms. The value of each component of the vector represents the importance of that term (typically referred to as the *weight* of the term) to that document or query. Then, given a query, we rank the relevance of documents to that query by measuring the similarity between the query's vector and each of the candidate document's vectors. The similarity between two vectors is generally measured as the cosine of the angle between them, computable using the following equation:

$$Sim(Q, D) = \frac{\sum_{t \in Q} w_{Q,t} \times w_{D,t}}{\sqrt{|Q| \times |D|}} \qquad (1)$$

where $w_{Q,t}$ represents the weight of term t for query Q and $w_{D,t}$ the weight of term t for document D. A $Sim(Q, D) = 0$ means that D does not have any term that is in Q. A $Sim(Q, D) = 1$, on the other hand, means that D has every term that is in Q. Typically, $|Q|$ is dropped from the denominator of equation 1 since it is constant for all the documents.

A popular method for assigning term weights is called the TFxIDF rule. The basic idea behind TFxIDF is that by using some combination of term frequency (TF) in a document with the inverse of how often that term shows up in documents in the collection (IDF), we can balance: (a) the fact that terms frequently used in a document are likely important to describe its meaning, and (b) terms that appear in many documents in a collection are not useful for differentiating between these documents for a particular query.

Existing literature includes several ways of implementing the TFxIDF rule [20]. In our work, we adopt the following system of equations as suggested by Witten et al. [25]:

$$IDF_t = \log(1 + N/f_t)$$

$$w_{D,t} = 1 + \log(f_{D,t}) \quad w_{Q,t} = IDF_t$$

where N is the number of documents in the collection, f_t is the number of times that term t appears in the collection, and $f_{D,t}$ is the number of times term t appears in document D.

The resulting similarity measure is

$$Sim(Q, D) = \frac{\sum_{t \in Q} IDF_t \times (1 + \log(f_{D,t}))}{\sqrt{|D|}} \qquad (2)$$

where $|D|$ = the number of terms in document D.

3.2 TFxIPF: PlanetP's Adaptation of TFxIDF

In the interest of saving communication bandwidth and storage, our two-level indexing scheme does *not* maintain sufficient information in the global index for a searching peer to run TFxIDF directly. We instead approximate TFxIDF by breaking the ranking problem into two subproblems: (1) ranking peers according to the likelihood of each peer having documents relevant to the query, and (2) deciding on the number of peers to contact and ranking the documents returned by these peers.

The node ranking problem. To rank peers, we introduce a measure called the *inverse peer frequency* (IPF). For a term t, IPF_t is computed as $\log(1 + N/N_t)$, where N is number of peers in the community and N_t is the number of peers that have one or more documents with term t in it. Similar to IDF, the idea behind this metric

is that a term that is present in the index of every peer is not useful for differentiating between the peers for a particular query. Unlike IDF, IPF can conveniently be computed using our limited index: N_t is simply the number of "$t \rightarrow p$" entries in the index. N is the number of peers, which we can derive easily from the directory.

Given the above definition of IPF, we then use the following relevance measure for ranking peers:

$$R_p(Q) = \sum_{\{t|t \in Q \wedge (t \rightarrow p) \in I\}} IPF_t \quad (3)$$

which is simply a weighted sum over all query terms contained by peer p, weighted by how useful that term is to differentiate between peers; t is a term, Q is the query, I is global index, and R_p is the resulting relevance of peer p to query Q. Intuitively, this scheme gives peers that contain all terms in a query the highest ranking. Peers that contain different subsets of terms are ranked according to the power of these terms for differentiating between peers with potentially relevant documents.

The selection problem. As communities grow in size, it is neither feasible nor desirable to contact a large subset of peers for each query. To address this problem, we assume that the user specifies an upper limit k on the number of document names that should be returned when posting a query Q. Then, given a pair (Q, k), PlanetP does the following. (1) Compute a relevance ranking of the peers for Q. (2) Contact the peers one-by-one from top to bottom of the ranking. (3) Each contacted peer returns a set of document names together with their relevance ranking using equation 2 with IPF_t substituted for IDF_t. Note that peers maintain $f_{D,t}$ in their local indexes and so has all the data required by equation 2. (4) Stop contacting peers when the documents returned by a sequence of p peers fail to contribute to the top k ranked documents.

Intuitively, the idea of the above algorithm is to get an initial set of k documents and then keep contacting nodes only if the chance of them being able to provide documents that contribute to the top k is relatively high. Using experimental results from a number of known document collections (see Section 4), we propose the following function for p:

$$p = \left\lfloor 2 + \frac{N}{300} \right\rfloor + \left\lfloor \frac{\sqrt{k}}{2.5} \right\rfloor \quad (4)$$

where N is the size of the community.

Note that while we have presented the above algorithm as contacting peers one-by-one, to reduce query response time, we might choose to contact peers in groups of m peers at a time. Such a parallel algorithm trades off potentially contacting some peers unnecessarily for shorter response time.

3.3 Implementing the Inverted Index

There are a number of ways to implement PlanetP's global index [25]. We have chosen to use Bloom filters [1], where each peer summarizes the set of terms in its local index in a Bloom filter. The global index is then just the set of Bloom filters, which is replicated everywhere using gossiping.

Briefly, a Bloom filter is an array of bits used to represent a set of strings; in our case, the set of terms in the peer's local index. The filter is computed by using n different hashing functions to compute n indices for each term and setting the bit at each index to 1. Given a Bloom filter, we can ask, is some term t a member of the set by computing the n indices for t and checking whether those bits are 1. Bloom filters can give *false positive* but never *false negative*.

We choose to use Bloom filters because they are relatively space efficient and provide two important advantages: (1) The cost of replacing a Bloom filter in the global index is constant, regardless of the number of changes captured in the move from the old to the new filter. For example, suppose a member published an additional set of documents with m new words. The cost of replacing the old Bloom filter with the new is independent of m whereas the cost of updating a word-to peer index would be at least $O(m)$. (2) Peers can independently trade-off accuracy for storage. For example, a peer a might choose to combine several filters into one if they are similar to each other to save space; the trade-off is that a must now contact this set of peers whenever a query hits on this combined filter. This ability for independently trading accuracy for storage is particularly useful for peers running on memory-constrained devices (e.g., hand-held devices).

4 Performance

Having described the two major components of PlanetP, we now turn to evaluating PlanetP's performance. We start by assessing the efficacy of PlanetP's content search and ranking algorithm. We then evaluate the costs, space and time, and the reliability of the supporting infrastructure, i.e., the replication of the directory and the global index using gossiping. Our performance study is simulation-based but most of the parameters were derived from a prototype implementation.

4.1 Search Efficacy

We measure PlanetP's search performance using two accepted information retrieval metrics, *recall* (R) and *precision* (P). R and P are defined as follows:

$$R(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. relevant docs. in collection}} \quad (5)$$

Collection	No. Queries	No. Docs	No. Unique Terms	Size (MB)
CACM	52	3204	75493	2.1
MED	30	1033	83451	1.0
CRAN	152	1400	117718	1.6
CISI	76	1460	84957	2.4
AP89	97	84678	129603	266.0

Table 1: *Characteristic of the collections used to evaluate PlanetP search and retrieval capabilities.*

$$P(Q) = \frac{\text{no. relevant docs. presented to the user}}{\text{total no. docs. presented to the user}} \quad (6)$$

where Q is the query posted by the user. $R(Q)$ captures the fraction of relevant documents a search and retrieval algorithm is able to identify and present to the user. $P(Q)$ describes how much irrelevant material the user may have to look through to find the relevant material. Ideally, one would like to retrieve all the relevant documents ($R = 1$) and not a single irrelevant one ($P = 1$). In our distributed context, it would also be ideal to contact as few peers as possible to achieve $R = 1$ and $P = 1$.

We evaluate PlanetP's performance by comparing its achieved recall and precision against the original TFxIDF algorithm. If we can match TFxIDF's performance, then we can be confident that PlanetP provides state-of-the-art search and retrieval capabilities[2], despite the accuracy that it gives up by severely limiting the amount of information kept in the global index.

We use five collections of documents to evaluate PlanetP's performance. Each set of documents has an associated set of queries and a binary mapping function of whether a document D is relevant to a particular query Q. Table 1 presents the main characteristics of these collections. Four of the collections, CACM, MED, CRAN, and CISI were previously collected and used by Buckley to evaluate Smart [3]. These collections contain small fragments of text and summaries and so are relatively small in size. The last collection, AP89, was extracted from the TREC collection [12] and includes full articles from the Associated Press published in 1989.

We measure recall and precision for PlanetP and TFxIDF using a simulator, which distributes the documents in a given collection across a set of virtual peers and then runs and evaluate a given search and retrieval algorithm. We assume the following optimistic implementation of TFxIDF: each peer in the community has the full inverted index and word count needed to run TFxIDF locally using ranking equation 2. For each query, TFxIDF would compute the top k ranking documents and then contact the exact peers required to retrieve these documents. In both cases, our simulator will pre-process the traces by doing stop word removal and stemming. The former tries to eliminate frequently used words like "the", "of", etc. and the second tries to reduce words to their root (e.g. "running" becomes "run").

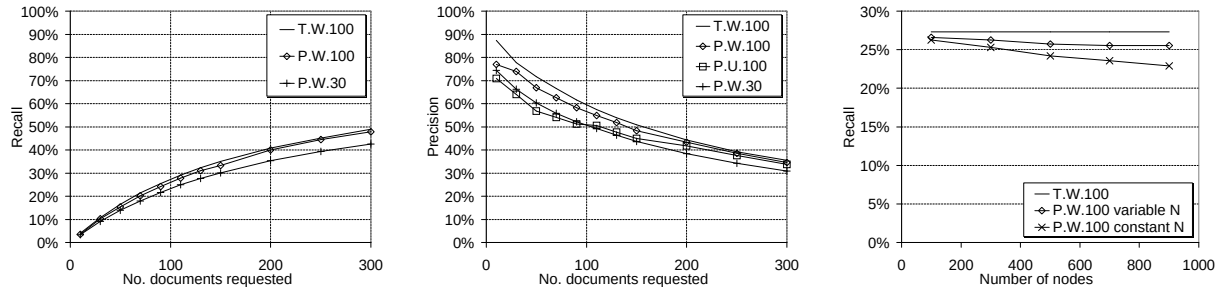
We study PlanetP's performance under two different distributions of documents among peers in the community: (a) Uniform, and (b) Weibull. We study a Uniform distribution because it presents the worst case for a distributed search and retrieval algorithm. The documents relevant to a query are likely spread across a large number of peers. The distributed search algorithm must find all these peers to achieve high recall and precision. The motivation for studying a Weibull distribution arises from measurements of current P2P file-sharing communities. For example, Saroiu et al. found that 7% of the users in the Gnutella community share more files than all the rest together [21]. We have also studied a local community: students with access to the Rutgers's dormitory network have created a file-sharing community comprised of more than 500 users, sharing more than 6TB of data. This community displays similar characteristics to those reported by Saroiu et al. Our Weibull distribution is parameterized to approximate the distribution found in this local community.

Figure 1(a) and (b) plot average recall and precision over all provided queries as functions of k for the AP89 collection. We only show results for this collection because of space constraints; these results are representative for all collections. We refer the reader to our web site for results for all collections: http://www.panic-lab.rutgers.edu/Research/PlanetP/. Figure 1(c) plots PlanetP's recall against community size for a constant k of 100. Finally, Figure 2 plots the number of peers contacted against k.

We make several observations. First, PlanetP tracks the performance of TFxIDF closely, *even when we index only the most frequently appearing 30% of the unique terms in each document*. Further, PlanetP's performance is independent of how the shared documents are distributed, achieving nearly the same performance for Uniform and Weibull. For a Weibull distribution of documents, when we index all 100% of the unique terms, PlanetP's recall and precision is within 11% of TFxIDF's (average difference is 4%). When we index only the 30% most frequently appearing terms, PlanetP's recall and precision is within 16% of TFxIDF's, with an average different of 14%. These small differences demonstrate that it is possible to preserve TFxIDF's performance while limiting the global index to only a term-to-peer mapping. The good performance given when we only index the top 30% of the unique terms indicate that we can further reduce the size of the global index at the expense of only a slight loss in ranking accuracy.

Second, PlanetP scales well, maintaining a relatively

[2] when only using the textual content of documents, not link analysis as is done by Google and other web search engines [2]



(a) **(b)** **(c)**

Figure 1: *Average (a) recall and (b) precision for the AP89 collection when distributed across 400 peers. The legends X.Y.Z are decoded as follows: X = {T: TFxIDF, P: PlanetP}, Y = {W: Weibull, U: Uniform}, and Z = {z: indexed the most frequently appearing z% of the unique terms in each document}; for example, T.W.100 means TFxIDF running on a Weibull distribution of documents, where all 100% of the unique terms of each document was indexed. (c) Average recall as a function of community size.*

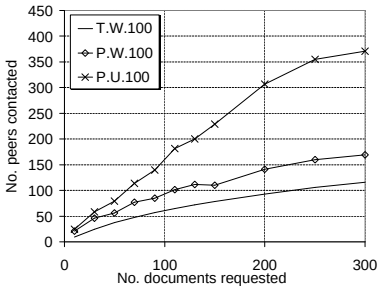


Figure 2: *Average number of peers contacted in a community of 400 peers vs. k.*

constant recall and precision for communities of up to 1000 peers. We have not study scalability beyond that point because the collections are not sufficiently large.

Third, PlanetP's adaptive stopping heuristic is critical to its performance. Figure 1(c) shows that PlanetP's recall would degrade with community size if the stopping heuristic was not a function of community size. While we did not show PlanetP's performance if the stopping heuristic were not a function of k because it would have made Figure 1(a,b) too busy, the effect is similar to that observed from Figure 1(c). In addition, PlanetP's adaptive stopping heuristic allows it to perform well independent of how the documents are distributed. Figure 2 shows that the dynamic stopping heuristic allows PlanetP to search more widely among peers when documents are more widely distributed, preserving recall and precision when documents are uniformly distributed across the community.

Finally, PlanetP does require contacting a larger number of peers than TFxIDF in the ideal case. This is the cost of maintaining a limited global index. We observe from Figure 2 that while this cost is not trivial, it does not seem unreasonable. For example, PlanetP contacts only 30% more peers at $k = 150$ than the ideal TFxIDF. Further, when documents are distributed according to a Weibull distribution, PlanetP only contact a small portion of the community if k is not too large. For example, PlanetP only contacts a little over 25% of the 400 peers when $k = 150$. Of course, when documents are uniformly distributed, PlanetP has to contact many more peers because documents are more widespread.

4.2 Storage Cost

Having demonstrated that PlanetP can preserve TFxIDF's ranking accuracy, we now turn to assess the storage requirement of our approach. In particular, we estimate the size of the global index if the entire TREC collection (944651 documents, 256686468 terms, 592052 unique terms, 3428.41 MB) was uniformly distributed across a community. (Note that this is the worse possible case because any other distribution, e.g., Weibull, would give a smaller summation of unique words per node.) In Figure 3, we count the number of unique words at each peer if this collection was distributed across N peers and compute the size of the global index if each Bloom filter was sized to summarize the per-node unique terms with 5% or less probability of error. We also show what happens if each document is replicated 3 times in the community.

Observe that at 1000 peers, the global index is quite small: 16.1MB, which is just 0.5% of the collection. If each document were replicated 3 times, the storage requirement would increase to 28.7MB, which is actually only 0.3% of the enlarged collection. At 5000 peers, the storage cost is somewhat higher, rising to 62.3MB if each document is replicated 3 times. Observe, however, that if we sacrifice a little accuracy (per Figure 1) by indexing only the 30% most frequent unique terms in each document, the storage requirement is reduced again to 26.9MB, which is just 0.3% of the replicated collection. Finally, we observe that the ratio of unique terms to collection storage size is probably quite high for TREC as all TREC documents are text documents. For collections

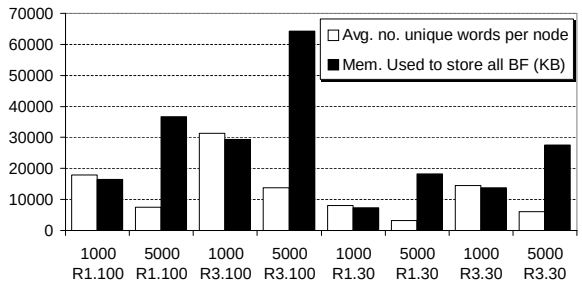


Figure 3: *Estimating the size of the global index when the TREC collection is uniformly distributed across a community of N peers. Each group of two bars show, from left to right, the average number of unique words found on each peer and the size of the global index (in KB) if individual Bloom filters were big enough to summarize the per-node unique terms with 5% of less probability of error. The name of each bar includes the community size, the replication factor (R1 or R3), and the percentage of per-document unique terms indexed.*

that include multi-media content, the ratio of our global index to collection size should be much much smaller.

Based on these results, we conclude that PlanetP should easily scale to several thousand peers in terms of the required per peer storage for the replicated global index, which is in the range of tens of MB even for a collection with a very high term to collection size ratio.

4.3 Computing Cost

We now proceed to assess the cost to generate and use the global index. In particular, we report the time to create a Bloom filter and the time to search across a set of Bloom filters when measured on an 800 MHz Pentium III PC with 512MB of memory, running a Linux 2.2 kernel and IBM's JVM v1.3.0.

$$T_{\textit{Create-BF}} = 4ms + (0.011ms \times \textit{no. terms})$$

$$T_{\textit{Search-BF}} = 0.010ms \times \textit{no. terms}$$

From these measurements, we derive a time of about 1/2 a second to create a Bloom filter for 50,000 terms. It takes only 50ms to search for a query with five terms across 1000 Bloom filters. Thus, we conclude that the times for generating and searching the global index are relatively negligible. (For example, each peer could easily generate a new Bloom filter to incorporate any changes every few minutes.)

4.4 Gossiping Performance

Finally, we turn to assess the reliability and scalability of PlanetP's gossiping algorithm. By reliability, we mean: Does each change propagate to all on-line peers? We perform this study using a simulator parameterized with measurements from our prototype. Table 2 lists these parameters. We validated our simulator by comparing its results against numbers measured on a cluster of eight 800 MHz Pentium III PCs with 512MB of memory, running a Linux 2.2 kernel and the BlackDown JVM, version 1.3.0. We switched to BlackDown's JVM for this study because it has a smaller memory footprint than the IBM JVM. Even with this switch, the JVM's resource requirements effectively limited us to about 25 peers per machine, allowing us to validate our simulation for community sizes of up to 200 peers.

Parameter	Value
CPU gossiping time	5ms + (transfer-time × no. bytes)
Base gossiping interval	30sec
Max gossiping interval	60sec
Network BW	56Kb/s to 45Mb/s
Message header size	3 bytes
1000 terms BF	3000 bytes
20000 terms BF	16000 bytes
BF summary	6 bytes
Peer summary	48 bytes

Table 2: *Constants used in our simulation of PlanetP's gossiping algorithm.*

In our current implementation of PlanetP, a global directory that includes the list of peers, their IP addresses, and their Bloom filters is replicated everywhere. Events that change the directory and so require gossiping include the joining of a new member, the rejoin of a previously off-line member, and a change in a Bloom filter. We do not gossip the leaving (temporary or permanent) of a peer. Each peer discovers that another peer is off-line when an attempt to communicate with it fails. It marks the peer as off-line in its directory but does not gossip this information. When the peer x comes back on-line, its presence will eventually be gossiped to the entire community; each peer that has marked x as off-line in its directory changes x's status back to on-line. If a peer has been marked as off-line continuously for T_{Dead} time, then all information about it is dropped from the directory under the assumption that the peer has left the community permanently.

Propagating new information. We begin by studying the time required to gossip a new Bloom throughout stable communities of various sizes. Measuring the propagation time is important because it is the window of time where peers' directories are inconsistent, so that some peers may not be able to find the new (or modified) documents.

In this experiment, we use a Bloom filter with 1000 words. Because PlanetP sends diffs of the Bloom filters to save bandwidth, this scenario simulates the addition of 1000 new terms to some peer's inverted index. Note that, while 1000 new terms may seem small, it actually is quite

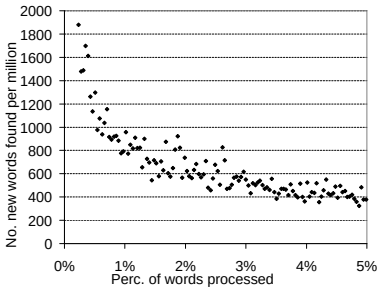


Figure 4: *Number of new unique words found per million parsed words vs. the percentage of words already processed in the TREC collection.*

large. Figure 4 shows that if a peer already contains 0.4% of the TREC collection, it would have had to add approximately 3000 more documents, totaling 800,000 more terms, to have found an additional 1000 unique terms. (The trend we found in Figure 4 is consistent with that found by a much larger study of words distribution [24].)

Figure 5(a) plots the simulated propagation times for six scenarios:

LAN Peers are connected by 45 Mbps links. Peers use PlanetP's gossiping algorithm.

LAN-AE Peers are connected by 45 Mbps links. Peers only use push anti-entropy to propagate information as opposed to PlanetP's combined algorithm. Anti-entropy only approaches have been successfully used to synchronize smaller communities in Name Dropper [11], Bayou [6] and Deno [14].

DSL-10,30,60 Peers are connected by 512 Kbps links. Peers use PlanetP's gossiping algorithm. Gossiping interval is 10, 30, and 60 seconds respectively.

MIX Peers are connected by a mixture of link speeds. Using measurements of the Gnutella/Napster communities recently reported by Saroiu et al. [21], we create a mixture as follows: 9% have 56 kbps, 21% have 512 kbps, 50% have 5 Mbps, 16% have 10 Mbps, and 4% have 45 Mbps links.

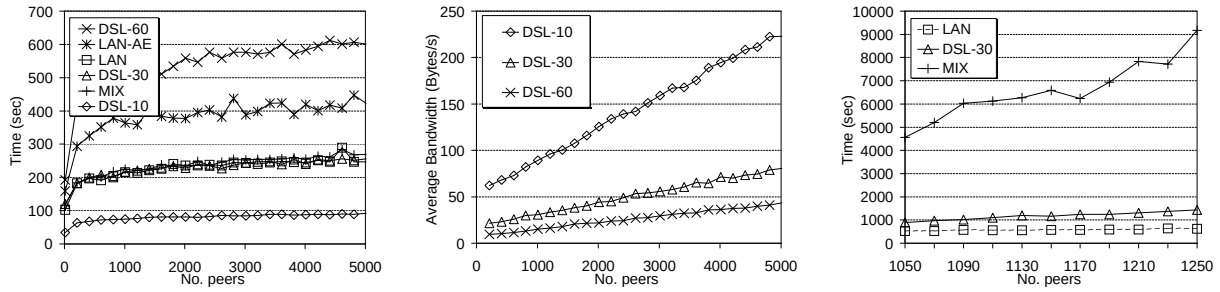
Figure 5(b) shows the average per peer gossiping bandwidth during the propagation period for DSL-10, DSL-30, and DSL-60.

Based on these graphs, we make several observations. (1) Propagation time is still a log function of community size [5], implying that gossiping new information is very scalable. For example, propagation time for a community with 500 peers using DSL-30 is about 200 seconds, rising to only 250 for a community with 5000 peers. (2) Even though a change is diffused throughout the entire community, the total number of bytes sent is very modest, again implying that gossiping is very scalable. For example, propagation of a 1000 new terms throughout a community of 5000 peers requires an aggregated total of about 100 MB to be sent, leading to a per-peer average bandwidth requirement of less than 100 Bps when the gossiping interval is 30 seconds. (3) We can easily trade off propagation time against gossiping bandwidth by increasing or decreasing the gossiping interval; slower gossiping rate means slower convergence but also lower bandwidth usage. (4) Our algorithm significantly outperforms one that uses only anti-entropy for both propagation time and for network volume. (We did not show the difference in network volume due to space constraints. On average, LAN-AE required 2.3 times the network volume of LAN.) With respect to network volume, this is because anti-entropy requires the communication of the entire directory (in summary form), even when there is only one difference, making message size proportional to the community size. In PlanetP, since most information is spread via rumoring and the partial anti-entropy, message sizes are mostly proportional to the number of changes being propagated, not the community size. With respect to propagation time, a push-only algorithm often has trouble locating the last few peers that have not receive a piece of new information, and so is outperformed by our push/pull algorithm.

Joining of new members. We now assess the expense of having large groups of new members simultaneously join an established community. This represents the transient case of a rapidly growing community and is the worst case for PlanetP because each of these new members has to download the entire global index. Our simulator currently assumes that each client is single-threaded. Thus, a new member that is busy downloading the global index for a long time can cause significant variation in the propagation time of changes; this member cannot receive gossip messages while it is busy downloading.

In this experiment, we start a community of n peers and wait until their views of membership is consistent. Then, m new peers will attempt to join the community simultaneously. We measure the time required until all members have a consistent view of the community again as well as the required bandwidth during this time. For this experiment, each peer was set to share 20,000 terms with the rest of the community through their Bloom filters. (Looking at Figure 3, observe that this is the equivalent of having a collection larger than the entire TREC collection shared by this community.)

Figure 5(c) plots the time to reach consistency vs. the number of joining peers for an initial community of 1000 nodes. These results show that, if there is sufficient bandwidth (LAN), consistency is reached within approximately 600 seconds (10 minutes), even when the community grows by 25%. In contrast to propagating a change, however, the joining process is a much more bandwidth intensive one; a joining member must retrieve 1000 Bloom filters representing a total of 20 million



(a) **(b)** **(c)**

Figure 5: *(a) Time and (b) average per-peer bandwidth required to propagate a single Bloom filter containing 1000 terms everywhere vs. community size. (c) Time required for $x - 1000$ peers to simultaneously join the community of 1000 stable online peers, each wishing to share 20000 terms.*

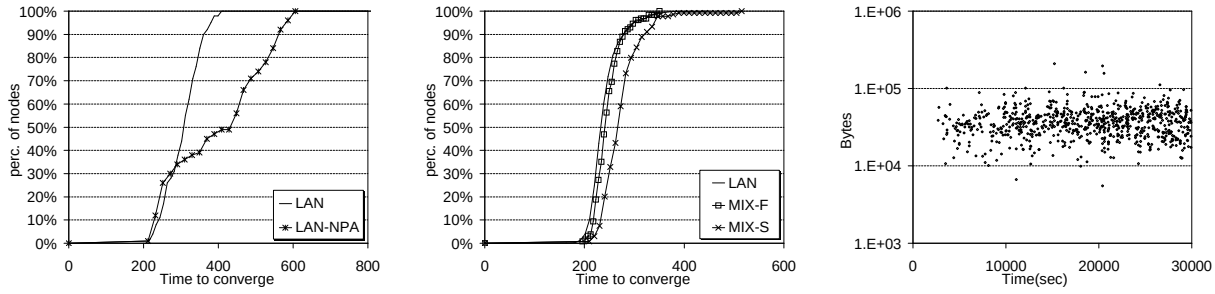
terms from the existing community. Also, having 250 members join at once means that 250 Bloom filters representing 5 million terms must be gossiped throughout the community. As a result, convergence times for communities interconnected only with DSL-speed links are approximately twice that of LAN-connected communities. Finally, convergence times for the MIX-connected communities become unacceptable, possibly requiring from 50 minutes to over two hours.

We draw two conclusions from these results. First, even in this *worst-case scenario* for PlanetP, which we *do not* expect to occur often, if peers have DSL or higher connectivity, then PlanetP does quite well. Second, we need to modify PlanetP if we are to accommodate users with modem-speed connections. While the artificial lengthening of gossiping convergence time can be easily fixed if peers are assumed to be multi-threaded, when a new peer first join, the time to download the entire directory would still likely take too long. For example, to download a thousand Bloom filters we would transfer 16MB which takes around 40 minutes on a modem connection. Thus, either we should exclude peers that do not have at least DSL speed, or we need to allow a new modem-connected peer to acquire the directory in pieces over a much longer period of time. We would also need to support some form of proxy search, where modem-connected peers can ask peers with better connectivity to help with searches.

Further, we decided to modify our gossiping algorithm to be bandwidth-aware, assuming that peers can learn of each other's connectivity speed. The motivation for this is that a flat gossiping algorithm penalizes the community to spread information only as fast as the slow members can go. Thus, we modify the basic PlanetP gossiping algorithm for peers with faster connectivity to preferentially gossip with each other and peers with slower connectivity to preferentially gossip with each other. This idea is implemented as follows. Peers are divided into two classes, fast and slow. Fast includes peers with 512 Kb/s connectivity or better. Slow includes peers connected by modems. When rumoring, a fast peer makes a binary decision to talk to a fast or slow peer. Probability of choosing a slow peer is 1%. Once the binary decision has been made, the peer chooses a particular peer randomly from the appropriate pool. When performing anti-entropy, a fast peer always chooses another fast peer. When rumoring, a slow peer always chooses another slow peer, so that it cannot slow down the target peer, unless it is the source of the rumor; in this case, it chooses a fast peer as the initial target. Finally, when performing anti-entropy, a slow peer chooses any node with equal probability. We will study the effects of this modified algorithm below.

Dynamic operation. Finally, we study the performance of PlanetP's gossiping when a community is operating in steady state, with members rejoining and leaving dynamically but without massive, simultaneous joins of new peers needing the entire global index. We expect this to be the common operational case for PlanetP. We begin by studying the potential for interference between different rumors as peers rejoin the community at different times. This experiment is as follows. We have a stable community of 1000 on-line peers; 100 peers join the community according to a Poisson process with an average inter-arrival rate of once every 90 seconds. Peers are connected LAN speed. Each on-line peer has a Bloom filter with 1000 terms that off-line peers do not have. Each joining peer shares a Bloom filter with 1000 terms. Again, this represents the case where off-line peers will have some new information to share, but they have to collect new information that may have accrued since they have been off-line. Figure 6(a) plots the cumulative percentage of events against the convergence time—the time required for an arrival event to be known by everyone in the on-line community—for PlanetP's gossiping algorithm against what happens if the partial anti-entropy is not included. Observe that without the partial anti-entropy, overlapping rumors can interfere with each other, causing much larger variation in the convergence times.

To complete our exposition, we study a dynamic com-



(a) **(b)** **(c)**

Figure 6: *(a) CDF of gossiping convergence time in a community of 1000 when there are 100 Poisson arrival (New arrivals share 1000 keys). LAN-NPA is our gossiping algorithm without the partial anti-entropy component. (b) CDF of gossiping convergence time during the normal operation of a dynamic community with 1000 members. MIX-F is the time it takes to a fast node to reach all other fast nodes and MIX-S is time it takes to a slow nodes to reach the whole community. (c) Aggregated gossiped data for (b).*

munity with the following behavior. The community is comprised of 1000 members[3]. 40% of the members are online all the time. 60% of the members are online for an average of 60 minutes and then offline again for an average of 140 minutes. Both online and offline times are generated using a Poisson process. 20% of the time, when a peer rejoins the on-line community, it sends a Bloom filter diff containing 1000 new terms. These parameters were again based roughly on measurements reported by Saroiu et al. [21] (except for the number of new terms being shared occasionally) and are meant to be representative of real communities. We note again that 1000 new unique terms typically represents the sharing of a **significant** set of new documents. (We have also studied a more dynamic community, where 50% a peer coming back on-line shares 100 new words. The results are similar to those present below.)

Figure 6(b) plots the cumulative percentage of events against the convergence time. We observe that with sufficient bandwidth, convergence time is very tight around 400 seconds. For the MIX community we separate the CDF in two classes: the time it takes for fast nodes to propagate events to other fast nodes (MIX-F) and the time it takes for slow nodes to reach the whole community (MIX-S). The graph shows that thanks to our bandwidth aware gossiping algorithm fast nodes can propagate events as in the LAN case, while the slow nodes stay unaffected. Although is not shown on the graph, the slow nodes are equally fast when propagating to fast nodes (because they can rumor to a fast node once and then let the fast nodes continue the propagation).

Figure 6(c) plots the aggregate bandwidth against time. This graph shows that the normal operation of a community requires very little bandwidth, ranging from between 10 KB/s to 100 KB/s across the entire community.

[3]We are currently simulating larger communities but did not have time to complete these results for the submission of this paper. We hope to have these numbers for publication if the paper is accepted.

5 Related Work

While current P2P systems such as Gnutella [9] and KaZaA [13] have been tremendously successful for music and video sharing communities, their search capabilities have been frustratingly limited. Our goal for PlanetP is to increase the power with which users can locate information in P2P communities. In particular, we do not depend on the particular characteristic of multi-media sharing, where a small subset of popular files are highly replicated everywhere, to achieve good search results. Also, we have focused more tightly on text-based information, which is more appropriate for collections of scientific documents, legal documents, inventory databases, etc. (Although again, our text-based search and rank algorithm can also be applied to music and video files because these typically have embedded descriptive text these days.)

In contrast to mainstream systems, recent research efforts in P2P seek to provide the illusion of having a distributed hash table (DHT) shared by all members of the community. Frameworks like Tapestry [26], Pastry [19], Chord [22] and CAN [17] use different techniques to spread (key, value) pairs across the community and to route queries from any member to where the data is stored. These systems differ from PlanetP in two key design decisions. First, in PlanetP, we explicitly decided to replicate the global directory everywhere using gossiping, which limits PlanetP's scalability. The advantage that we get, however, is that we do not have to worry about what happens to parts of the global hash table if members sign off abruptly. Recent work [16] has shown that DHTs need special stabilization algorithms in order to preserve their invariants on highly dynamic communities. In this paper we show that our combination of state replication and random gossiping provides bounded convergence times with no need for maintenance algorithms. Moreover gossiping has the advantage of distributing the

cost propagating new information across the community, thus helping weak nodes.

Second, we have focused on content search and retrieval, attempting to provide a similar service to Internet search engines. Several efforts parallel to PlanetP have also started to explore this issue[8, 18]. Their focus, however, is still on serving very large-scale communities. Thus, they partition the global inverted index among the members of the community, a completely different design point than PlanetP. For example, this design point trades off communication for scalability: whereas PlanetP's search only requires contacting peers with potentially relevant documents, these other systems would need to communicate to even locate the appropriate parts of the global inverted index. Finally, PlanetP is the only P2P framework at this point that support document ranking capabilities to help users find the most relevant information within large data collections.

Many mechanisms used to build PlanetP have been already used on different environments. For example anti-entropy and rumor mongering was introduced by Demers et al. [5] and since then applied to problems like membership [11], information aggregation [23], etc. Astrolabe [23] is an infrastructure that can perform SQL queries over SNMP style databases that are widely distributed. Although they can scale to 100000 nodes, they rely on mechanisms like broadcast and multicast to aid in membership maintenance. As far as we know nobody has looked at gossiping in scenarios were nodes join and leave constantly and in an uncontrolled manner. In this paper we have quantified the use of gossiping techniques on P2P environments and adapted them for better bandwidth usage and propagation time stability.

More related to PlanetP's information retrieval goals, Cori [4] and Gloss [10] address the problems of database selection and ranking fusion on distributed collections. Recent studies done by French et al. [7] show that both scale well to 900 nodes. Although Cori and Gloss use different indexing techniques, both of them maintain more information in their global index than PlanetP. In particular, they both keep the per-node count of documents that contain each term t (Gloss actually keeps slightly more global information than this). Using this data, they construct a centralized index that can direct a query to the set of nodes that with high probability will give the best documents. Because PlanetP is targeted toward communities that are larger, more dynamic, yet does not have any centralized resources, we have chosen to keep the minimal possible information in the global index to minimize communication as well as storage. We have shown that, with our adaptive stopping heuristic, this smaller global index does not hurt search and ranking efficacy.

6 Conclusions and Future Work

P2P computing is a potentially powerful model for information sharing between *ad hoc* communities of users. As P2P communities grow in size, however, locating information distributed across the large number of peers becomes problematic. In this paper, we have presented PlanetP, a P2P information sharing infrastructure that indexes published documents and supports distributed content search and retrieval across them. Our thesis is that the search paradigm, where a small set of relevant terms is used to locate documents, is as natural as locating documents by name. To be useful, however, the search and retrieval algorithm must successfully locate the information the user is searching for, without presenting too much unrelated information.

PlanetP encompasses two novel design decisions to support an effective P2P content addressable publish/subscribe service. First, PlanetP is based on randomized gossiping, which effectively tolerates the dynamicity inherent in P2P communities. Second, PlanetP approximates a state-of-the-art text-based document ranking algorithm, the vector-space model instantiated with the TFxIDF ranking rule. We have shown that PlanetP preserves the efficacy of the original algorithm with the replication of an extremely compact global index. The required storage and gossiping bandwidth are modest enough that PlanetP can easily scale to several thousand peers, with our current goal being around ten thousand peers.

While we did not start this work with the intention of scaling to millions or billions of users, we believe that it is possible to scale PlanetP beyond our initial target of ten thousand peers if desired. One possible approach to scaling that we will likely explore in the future is dividing the community into a number of groups. Peers within the same group operate as described here. Peers from different groups will gossip an attenuated Bloom filter that is a summary of the global index for their groups. Peers mostly gossip within their groups but, occasionally, will gossip to peers from other groups. When searching, if the attenuated Bloom filter of group g contains terms relevant to a query Q, then the searching peer, say a, would contact a random peer in group g, asking it to return a ranked list of peers in g that might have documents relevant to Q. a can then contact these peers using the current algorithm for ranking.

A second research direction that we are currently pursuing is the automatic replication of documents so that a peer can almost always locate documents highly relevant to a query, even if the original publisher is currently-offline.

Finally, we are in the process of building a number of applications to validate the utility of PlanetP. Specifically, we have built a prototype semantic file system and chat application on top of PlanetP. Other applications are un-

derway.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[3] C. Buckley. Implementation of the SMART information retrieval system. Technical Report TR85-686, Cornell University, 1985.

[4] J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks . In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, 1995.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[6] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The bayou architecture: Support for data sharing among mobile users. In *Proceedings IEEE Workshop on Mobile Computing Systems & Applications*, 8-9 1994.

[7] J. C. French, A. L. Powell, J. P. Callan, C. L. Viles, T. Emmitt, K. J. Prey, and Y. Mou. Comparing the performance of database selection algorithms. In *Research and Development in Information Retrieval*, pages 238–245, 1999.

[8] O. D. Gnawali. A keyword set search system for peer-to-peer networks. Master's thesis, Massachusetts Institute of Technology, 2002.

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

[10] L. Gravano, H. Garcia-Molina, and A. Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of the ACM SIGMOD Conference*, pages 126–137, 1994.

[11] M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *Symposium on Principles of Distributed Computing*, pages 229–237, 1999.

[12] D. Harman. Overview of the first TREC conference. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993.

[13] KaZaA. http://www.kazaa.com/.

[14] P. J. Keleher and U. Cetintemel. Consistency management in deno. *Mobile Networks and Applications*, 5(4):299–309, 2000.

[15] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, 2000.

[16] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing*, 2002.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM '01 Conference*, 2001.

[18] P. Reynolds and A. Vahdat. Eficient peer-to-peer keyword searching. Technical report, CS Department, Duke University, 2002.

[19] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[20] G. Salton, A. Wang, and C. Yang. A vector space model for information retrieval. In *Journal of the American Society for Information Science*, volume 18, pages 613–620, 1975.

[21] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking (MMCN)*, 2002.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, 2001.

[23] R. van Renesse and K. Birman. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. Submitted for publication, 2002.

[24] H. Williams and J. Zobel. Searchable words on the web. In submission, 2001.

[25] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, second edition, 1999.

[26] Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, 2000.