

# 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  
 Department of Computer Science, Rutgers University  
 110 Frelinghuysen Rd, Piscataway, NJ 08854  
 {mcuenca, peery, rmartin, tdnguyen}@cs.rutgers.edu

## Abstract

We introduce PlanetP, a content addressable publish/subscribe service for unstructured peer-to-peer (P2P) communities. PlanetP supports content addressing by providing: (1) a gossiping layer used to globally replicate a membership directory and an extremely compact content index, and (2) a completely distributed content search and ranking algorithm that helps users find the most relevant information. PlanetP is a simple, yet powerful system for sharing information. PlanetP is simple because each peer must only perform a periodic, randomized, point-to-point message exchange with other peers. PlanetP is powerful because it maintains a globally 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 while remaining resilient to rapid membership changes.

## 1 Introduction

Peer-to-peer (P2P) computing 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 cost, privacy and safety concerns of collecting and maintaining them in a centralized repository, (b) ease of incremental scalability, and (c) the possibility of scaling to extremely large sizes.

In this paper, we propose a novel approach to the construction of a content addressable publish/subscribe service that uses gossiping [4] to replicate global state across *unstructured* communities of several thousand, perhaps up to ten thousand, peers. The success of Internet search engines in indexing newsgroups and mailing lists (e.g., Google

Groups) as well as the web in general argues that content-based search and ranking is a powerful model for locating information across data collections exhibiting a wide range of sizes and content. We focus on unstructured P2P communities because the underlying infrastructure can be made resilient to unpredictable and rapid changes in membership without introducing undue complexity. In contrast, typical implementations of structured communities using distributed hash tables must implement very complex stabilizing algorithms [16].

Finally, we target thousands of peers because most other efforts have ignored this range in attempting to scale to millions of users (e.g., [27, 20, 23]). (Although we discuss several ideas for scaling PlanetP well beyond this level in Section 6.) Our target range can have significant impact: there are currently many communities around this size such as those served by Yahoo Groups, Dalnet's IRC servers, and thousands of other Usenet servers around the globe. Yahoo alone hosts more than two million user groups that share files and engage in public debates. On a different front, our approach can also be applied to manage distributed computing environments such as grid systems (e.g., maintaining membership, service description, and aggregate statistics); recent work shows the promise of such a P2P management approach [24, 6]. Thus, our work explores the question of whether certain functionalities, such as content ranking, that are extremely difficult to implement in very large systems becomes possible to implement at our target scale.

Our approach is comprised of 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<sup>1</sup>. The latter requires two small data structures to be replicated globally: a membership directory and an extremely compact content index. All members agree to continually gossip about changes to keep these shared

\*PlanetP was supported in part by NSF grants EIA-0103722 and EIA-9986046.

<sup>1</sup>Our algorithm can be used to search and rank multi-media as well as text documents since today's multi-media formats such as MP3 and AVI support the embedding of descriptive text.

data structures updated and loosely consistent. We explicitly chose gossiping because of its robustness to the dynamic joining and leaving of peers and independence from any particular subset of peers being on-line.

We have realized our proposed approach in a prototype system called PlanetP, which indexes shared documents in a way that allows peers across the entire community to locate specific document based on their content in an entirely distributed fashion. 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 complex distributed data structure. PlanetP is powerful for two reasons: (a) it can propagate information in bounded time in spite of the uncoordinated communal behavior, and (b) it maintains a globally content-ranked data collection without depending on centralized resources or the on-line presence of specific 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 [9, 3]?
- 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. In particular, we show that PlanetP achieves search and rank accuracy that is comparable to a centralized solution and scales easily to several thousand peers.

## 2 Gossiping

PlanetP uses gossiping to replicate shared state across groups of peers in a P2P community. PlanetP's gossiping algorithm is a novel combination of an algorithm previously introduced by Demers et al. [4] and a *partial anti-entropy* algorithm that we have found improves performance significantly for dynamic P2P environments. Briefly, Demers et al.'s algorithm works as follows when synchronizing a shared data structure that is replicated globally. Suppose  $x$  learns of a change to the replicated data structure. Every  $T_g$  seconds,  $x$  would *push* this change (called a rumor) to a peer chosen randomly from its directory; the directory is a

data structure that describes all peers in the community and is itself replicated everywhere using gossiping. If  $y$  has not seen this rumor, it records the change and also starts to push the rumor just like  $x$ .  $x$  stops pushing the rumor after it has contacted  $n$  consecutive peers that have already heard the rumor. To avoid the possibility of rumors dying out before reaching everyone, there is also a *pull* component (called anti-entropy): every  $T_r$  rounds,  $x$  would attempt to pull information from a random peer instead of pushing. In a pull message,  $x$  would ask the target  $y$  to reply with a summary of its version of the data structure. Then  $x$  can ask  $y$  for any new information that it does not have.

Unfortunately, in a dynamic P2P environment, the time required to spread new information can become highly variable. This is because rapid changes in the membership leads individual peers to have a less accurate view of the directory, elevating the problem of residual peers that do not receive rumors before they die out. The obvious solution of increasing the rate of anti-entropy is quite expensive: rumors are only as large as the update they carry but pull messages must contain a summary proportional to the community size.

Thus, we instead extend each push operation with a partial pull that works as follows. When  $x$  sends a rumor to  $y$ ,  $y$  piggybacks the identifiers of a small number  $m$  of the most recent rumors that  $y$  learned about but is no longer actively spreading onto its reply to  $x$ ; this allows  $x$  to pull any recent rumor that did not reach it. This partial pull requires only one extra message *in the case that  $y$  knows something that  $x$  does not* since the normal rumoring process is really implemented as a query/request/reply sequence using unique rumor identifiers to save bandwidth when the target has already received the rumor. Furthermore, the amount of data piggybacked on  $y$ 's message is of constant size, on order of tens of bytes.

Observe that while the pushing of rumors has a termination condition, pulling does not. To address this, PlanetP dynamically adjusts its gossiping interval  $T_g$ ; if a peer is not actively pushing any rumors, it slowly raises its  $T_g$  (to some maximum value). When it receives a new rumor, it immediately resets its gossiping interval to the default. This dynamic adaptation leads to negligible bandwidth usage shortly after global consistency has been achieved.

Finally, note that although in this paper, we assume that shared data structures are universally replicated and are gossiped with a single  $T_g$  for simplicity, this is not the general case. In fact, our implementation allows each data structure to be associated with only a subset of peers and gossiped at a distinct rate. This allows partial replication as well as rapid dissemination of time-sensitive information such as messages in group communications without increasing the overheads of maintaining more slowly changing data structures.

### 3 Content Search and Retrieval

Peers publish documents in PlanetP by providing XML snippets containing pointers to the appropriate files. (A document itself can be embedded in the XML snippet if it is not too large.) PlanetP leaves the shared files in place but runs a simple web server to support peers' retrieval of these files. PlanetP indexes each published document, maintaining a detailed inverted index describing all documents published by a peer locally. 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 term  $t$  is in the local index of peer  $p$ .

To find documents that contain a set of query terms, a searching peer first uses the global index to derive the set of peers that have these terms. Then, it forwards the query to these peers and asks them to return URLs for any documents that are relevant to the query. Each target peer uses its local index to find the appropriate documents. PlanetP uses this two-stage search process to perform exhaustive searches while limiting the size of the globally replicated index. (We will refer to the globally replicated index as the global index, while the more detailed index that describes *only* the documents published locally by a peer will be referred to as the local index.)

PlanetP also implements a content ranking algorithm that uses the vector space ranking model [26]; users can use this algorithm to find only documents that are highly relevant to a query. In the remainder of this section, we describe how we have adapted a state-of-the-art ranking algorithm to use PlanetP's two-level indexing scheme.

#### 3.1 Background: 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 value of each component of a vector is a weight representing the importance of that term to the corresponding document or query. Given a query, we then compute the relevance of each document as the cosine of the angle between the two vectors using the following equation:

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

where  $Q$  is the query,  $D$  is a document,  $|Q|$  and  $|D|$  are the number of terms in  $Q$  and  $D$ , respectively,  $w_{Q,t}$  represents the weight of term  $t$  for query  $Q$ , and  $w_{D,t}$  the weight of term  $t$  for document  $D$ . A similarity of 0 means that the document does not have any term in the query while a 1 means that the document contains every term in the query.

TFxIDF is a popular method for assigning term weights. This technique combines the term frequency (TF) in a docu-

ment with the inverse of how often that term shows up in the entire collection (IDF) to 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.

There are several accepted ways of implementing TFx-IDF [21]. In our work, we adopt the following system of equations from [26]:

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

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

where  $N_C$  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  $D$ .

This leads to a similarity measure of

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

where  $|Q|$  has been dropped from the denominator since it is constant for query  $Q$  across all documents.

#### 3.2 Approximating TFxIDF

In designing PlanetP, we deliberately decided not to maintain the term frequencies and " $t \rightarrow D$ " mappings necessary for TFxIDF in our global index to optimize space and reduce communication. In fact, with stop word removal and stemming<sup>2</sup>, our global index only contains the bare minimum of mappings from "important" words to peers. We then approximate TFxIDF by breaking the ranking problem into two sub-problems: (1) ranking peers according to their likelihood of having relevant documents, and (2) deciding on the number of peers to contact and ranking the identified documents.

**Ranking Peers.** 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 constrained global index:  $N$  is just the number of entries in the directory while  $N_t$  is the number of " $t \rightarrow p$ " entries in the global index.

Having defined IPF, we then rank peers using:

<sup>2</sup>Stop word removal eliminates words like "the", "of", etc.; stemming tries to reduce words to their root, e.g., "running" becomes "run."

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

which is a sum over all query terms contained in at least one document published by peer  $p$ , weighted by the usefulness of each term for differentiating between peers;  $t$  is a term,  $Q$  is the query,  $I$  is the global index, and  $R_p$  is the relevance of peer  $p$  to  $Q$ . 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 “differentiating potential” of the subsets.

**Selection.** As communities grow, it becomes infeasible to contact large subsets of peers for each query. To address this problem, we assume that the user specifies a limit  $k$  on the number of potential documents that should be identified in response to a query  $Q$ . Then, given a pair  $(Q, k)$ , PlanetP does the following. (1) Rank peers for  $Q$ . (2) Contact peers in groups of  $m$  from top to bottom of the ranking<sup>3</sup>. (3) Each contacted peer returns a set of document URLs together with their relevance using equation 2 with  $IPF_t$  substituted for  $IDF_t$ . This substitution is sufficient since peers maintain per-document term frequencies in their *local* indexes. (4) Stop contacting peers when the documents identified by  $p$  consecutive peers fail to contribute to the top  $k$  ranked documents.

The idea behind our algorithm is to get an initial set of  $k$  documents and then keep contacting peers only if there is a good chance of acquiring documents more relevant than the current  $k^{th}$ -ranked one. Simulation results show that  $p$  should be a function of the community size  $N$  and  $k$  as follows:

$$p = C_0 + \lfloor C_1 N \rfloor + \lceil C_2 \sqrt{k} \rceil \quad (4)$$

The tuple  $(C_0, C_1, C_2) = (2, 1/300, 1/2.5)$  can serve as a good initial value for equation 4 since it works well for the benchmark collections studied in this paper (see Section 4). In general, we assume that users will adjust  $k$  when the results are not satisfactory (as they do when using Internet search engines). If users have to increase  $k$ , then we should increase  $p$ . If users decrease  $k$  or never access the lowest ranked documents identified for queries, we should decrease  $p$ .

### 3.3 Implementing the Global Index

PlanetP’s global index can be implemented in a number of ways [26]. We use Bloom filters [1], where each peer summarizes the set of terms in its local index in a Bloom filter. 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

<sup>3</sup> $m$  represents a trade off between parallelism in contacting peers against potentially contacting some peers unnecessarily.

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 chose Bloom filters because they give PlanetP the flexibility to adjust to different needs. For example, the cost of replicating the global index can be reduced by simply decreasing the gossiping rate; updating the global index with a new Bloom filter requires constant time, regardless of the number of changes introduced. Furthermore, Bloom filters can be compressed and versioned to achieve a single bit per word average ratio. Memory-constrained Peers can also independently trade-off accuracy for storage by combining several filters into one.

## 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. Also, we validated our simulator against measurements taken from the prototype when running up to several hundred peers.

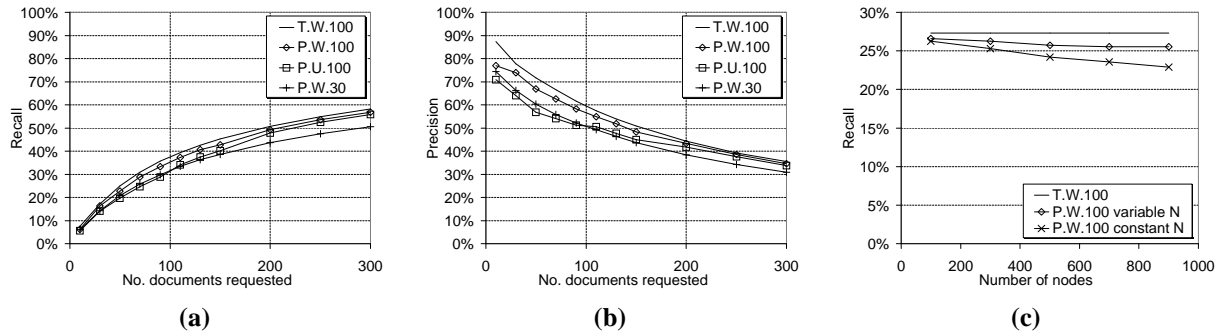
### 4.1 Search Efficacy

We measure PlanetP’s search performance using two accepted information retrieval metrics, *recall* ( $R$ ) and *precision* ( $P$ ) [26].  $R$  and  $P$  are defined as follows:

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

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

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  $P = 1$ .



**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:  $X = \{T: \text{search engine using TFxIDF}, P: \text{PlanetP}\}$ ,  $Y = \{W: \text{Weibull}, U: \text{Uniform}\}$ , and  $Z = \{z: \text{indexed the most frequently appearing } z\% \text{ of the unique terms in each document}\}$ ; 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.

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's search and ranking capabilities.

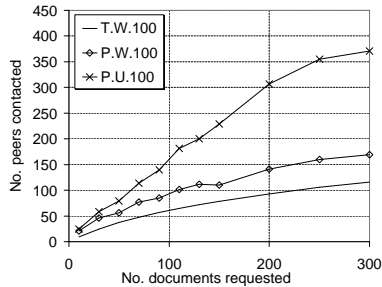
We assess PlanetP's ranking efficacy by simulating and comparing its performance for five benchmark collections (Table 1) against a centralized  $TFxIDF$  implementation (called CENT). Each collection has a set of documents, a set of queries, and a binary mapping of whether a document  $D$  is relevant to a particular query  $Q$ . Four of the collections, CACM, MED, CRAN, and CISI, were collected and used by Buckley [2]. 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 study PlanetP's performance under two different documents-to-peers distributions: (a) Uniform, and (b) Weibull. We study a Uniform distribution as 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. Saroiu et al. found that 7% of the users in the Gnutella community share more files than all the rest together [22]. We have also

studied a local community comprised of more than 1500 students sharing more than 10TB of data, which has a similar document distribution. 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 the centralized implementation 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 CENT's (average difference is 4%). When we index only the 30% most frequently appearing terms, PlanetP's recall and precision is within 16% of CENT's, with an average difference 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. Moreover when comparing the documents returned by PlanetP and CENT at low recall levels, we found an average inter-



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

section of 70%. (The intersection approaches 100% with increasing recall.) This gives us confidence that our adaptations did not change the essential ideas behind TFxIDF's ranking

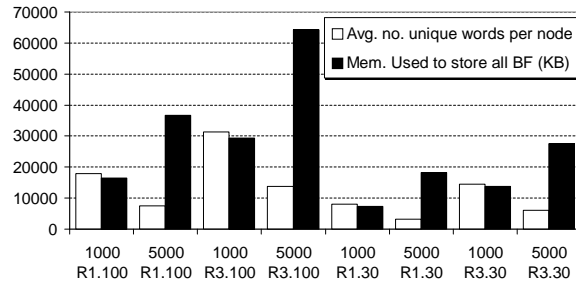
Second, *PlanetP scales well*, maintaining a relatively constant recall and precision for communities of up to 1000 peers. We have not study scalability beyond that point because the collections are not sufficiently large.

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 were not a function of community size. (The effect is similar if the stopping heuristic was not a function of  $k$ .) 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 independent of document distribution.

PlanetP's good distributed search and ranking performance does have a small cost: PlanetP contacts more peers than CENT. 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  $k = 150$  for the Weibull document distribution. Further, the percentage of peers contacted is small: PlanetP only contacts a little over 25% of the 400 peers at  $k = 150$ .

## 4.2 Storage Cost

Having demonstrated that PlanetP can preserve TFx-IDF's ranking accuracy, we now turn to assess the storage requirement of our approach. In particular, we estimate the size of the global index using the entire TREC collection (944,651 documents, 256,686,468 terms, 592,052 unique terms, 3,428.41 MB) for the worst case of uniform document distribution. This is the worst case because any other distribution (e.g. Weibull) would likely give a smaller summation of unique terms per node. Moreover, TREC is a



**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 shows, 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 at most 5% probability of error. Each bar is named after the community size, the replication factor (R1 or R3), and the percentage of per-document unique terms indexed.

collection of text documents, so the ratio of unique terms to collection size is very high. For collections with multimedia documents, this ratio is likely to be much smaller. For example, a collection of 326,913 MP3 files requiring 1.4TB of storage collected from an existing P2P community only yielded 55,553 unique terms.

In Figure 3, we count the number of unique words at each peer and compute the size of the global index if each Bloom filter was sized to summarize the per-node unique terms with less than 5% 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(a,b)) 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.

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.

## 4.3 Gossiping Performance

Finally, we assess the reliability and scalability of PlanetP's gossiping algorithm. By reliability, we mean does

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

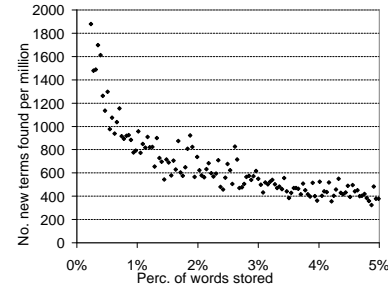
**Table 2.** Constants used in our simulation of PlanetP's gossiping algorithm.

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. Because of the JVM's resource requirements, we were limited to 25 peers per machine, allowing us to validate our simulation for community sizes of up to 200 peers.

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 filter throughout stable communities of various sizes. Measuring propagation time is important because it represents the window of time where peers' directories are inconsistent, so that some peers may not be able to find 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 terms found per million words vs. the percentage of words already stored at a node (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 word distribution [25].)

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.

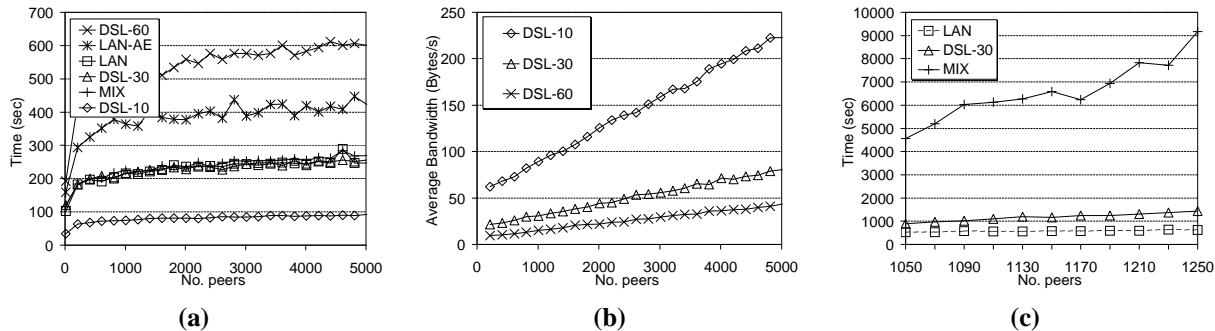
**LAN-AE** Peers are connected by 45 Mbps links. Peers use only push anti-entropy: each peer periodically push a summary of its data structure. The target requests all new information from this summary. This approach has been successfully used to synchronize smaller communities in Name Dropper [11], Bayou [5] 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 reported by Saroiu et al. [22], 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 gossiping bandwidth used per peer during the experiment for DSL-10, DSL-30, and DSL-60.

Based on these graphs, we make several observations. (1) *Propagation time is still a logarithmic function of community size [4], 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)



**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.

Even though a change is gossiped 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. And, (4) Our algorithm significantly outperforms ones that use only push anti-entropy for both propagation time and network volume. Using rumor-gossiping enables PlanetP to reduce the amount of information exchanged between nodes while the mixture of pull (anti-entropy) and push (rumors) rounds reduces convergence time. While we did not show the difference in network volume, on average, LAN-AE required 2.3 times the network volume of LAN.

**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 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. Thus, we should either exclude peers with less than DSL connectivity or 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.

We also 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 at 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 community with the following behavior. The community is comprised of 1000 members. 40% of the members are on-line 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. [22] (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% of the time, 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 our bandwidth aware gossiping algorithm allows fast nodes to propagate events as in the LAN case without harming the speed of propagation to slow nodes. Although it 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).

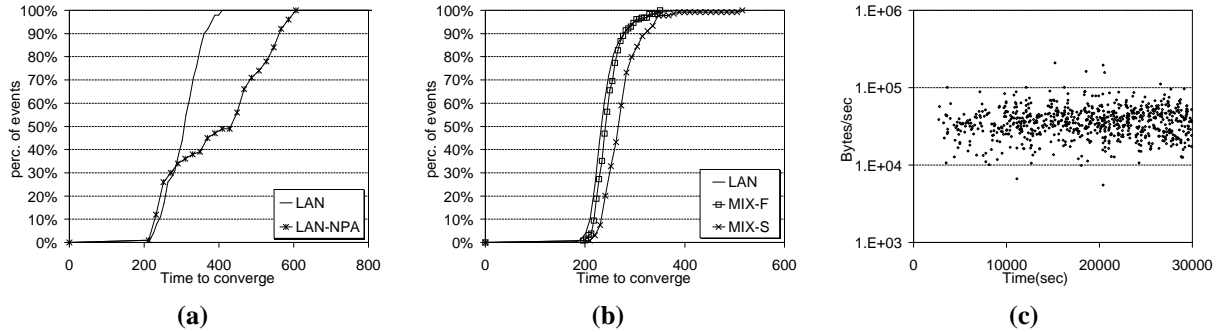
Figure 6(c) plots the aggregate bandwidth against time. This graph shows that the normal operation of a community requires very little bandwidth, ranging from between 10 KB/s to 100 KB/s across the entire community.

## 5 Related Work

While current P2P systems such as Gnutella [8] and KaZaA [13] have been tremendously successful for music and video sharing communities, their search and information diffusion capabilities have been frustratingly limited. Our goal for PlanetP is to increase the power with which users can locate information in P2P communities by providing content based search and ranking capabilities.

Several efforts parallel to PlanetP have also looked at better querying mechanisms [7, 19]. Their focus, however, is on serving very large-scale communities. In order to be scalable these systems trade off performance and functionality by using iterative queries and distributed inverted indexes. None of this previous work supports content ranking.

Numerous research efforts have produced highly scalable distributed hash tables (DHT) over P2P communities [27, 20, 23, 18]. In general DHTs spread (*key, value*) pairs across the community and provide retrieval mechanisms based on the *key*. Although this abstraction has been successfully used to build file system services [17, 15], we believe it is less suitable for the type of communities studied in this paper. The high cost of publishing thousands of



**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 a fast node to reach all other fast nodes and MIX-S is time it takes a slow node to reach the whole community. (c) Aggregated bandwidth usage while running (b).

keys per file and the lack of update propagation make it difficult to implement content-addressable publish/subscribe systems on DHTs. PlanetP overcomes these difficulties using gossiping to propagate information and replicating a compact inverted index on every peer.

Gossiping has been used in a variety of settings such as membership [11], information aggregation [24], and P2P DHTs [10], because of its robustness to failures. In PlanetP we have adapted them for better bandwidth usage and propagation time stability in scenarios where nodes join and leave constantly and in an uncontrolled manner (similar to the work done by Liben-Nowell et.al. [16] for DHTs).

More related to PlanetP's information retrieval goals, Cori [3] and Gloss [9] address the problems of database selection and ranking fusion on distributed collections. Both systems use servers to keep a reduced index of the content stored by other servers. Because PlanetP is targeted toward communities that are larger, more dynamic, yet does not have any centralized resources, we have chosen to keep even less information in the global index to minimize communication as well as storage. We have shown that our distributed search and rank algorithm using this minimal global index is nearly as effective as a centralized implementation of TFxIDF.

## 6 Conclusions and Future Work

The number of on-line communities has exploded with the growth of the Internet. Traditionally, these communities have been hosted on centralized servers, even when the information being shared exists (and is collected naturally) in a distributed form. In this paper, we seek to provide a powerful alternative for avoiding centralization when centralization is costly or presents privacy and safety concerns. In particular, we have presented PlanetP, a P2P pub-

lish/subscribe information sharing infrastructure that supports distributed content search, rank, and retrieval. PlanetP uses gossiping to robustly disseminate new information, even under rapid membership changes, and replicate a limited amount of global state to support content search. This combination allows PlanetP to support a powerful content addressing model without requiring peers to maintain complex distributed data structures.

We have shown that PlanetP's extremely compact global index does not affect its ranking accuracy: on average, PlanetP's ranking performance is only a few percent less than that of a centralized implementation of TFxIDF. Further, the overall required storage and gossiping bandwidth are modest enough that PlanetP can easily scale to several thousand peers. Our real target is 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 is to divide 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. Indeed, Gupta et. al.[10] recently proposed using a hierarchy of peers in a very similar manner, although their system uses a distributed hash table across groups instead of gossiping.

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 underway.

## 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] C. Buckley. Implementation of the SMART Information Retrieval System. Technical Report TR85-686, Cornell University, 1985.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
- [7] O. D. Gnawali. A Keyword Set Search System for Peer-to-Peer Networks. Master's thesis, Massachusetts Institute of Technology, 2002.
- [8] Gnutella. <http://gnutella.wego.com>.
- [9] 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.
- [10] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
- [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. Kubiawicz, 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 AS-PLoS*, 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] A. Muthitacharoen, R. Morris, T. Gil, and I. B. Chen. Ivy: A Read/Write Peer-to-peer File System. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, 2002.
- [18] 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.
- [19] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. Technical report, CS Department, Duke University, 2002.
- [20] 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.
- [21] 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.
- [22] 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.
- [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 *Proceedings of the ACM SIGCOMM '01 Conference*, 2001.
- [24] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology For Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems (TOCS)*, 21(2), May 2003.
- [25] H. Williams and J. Zobel. Searchable Words on the Web. *International Journal of Digital Libraries*, To appear.
- [26] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, second edition, 1999.
- [27] Y. Zhao, J. Kubiawicz, 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.