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*

Technical Report DCS-TR-487
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854

May 18, 2002

Abstract. *We present PlanetP, a peer-to-peer (P2P) content search and retrieval infrastructure targeting communities wishing to share large sets of text documents. P2P computing is an attractive model for information sharing between* ad hoc *groups of users because of its low cost of entry and explicit model for resource scaling. As communities grow, however, a key challenge becomes finding relevant information. To address this challenge, our design centers around indexing, content search, and retrieval rather than scalable name-based object location, which has been the focus of recent P2P systems. PlanetP takes the novel approach of replicating the global directory and a compact summary index at every peer using gossiping. PlanetP then leverages this information to approximate a state-of-the-art document ranking algorithm to help users locate relevant information within the large communal data set. Using a prototype implementation together with simulation, we show: (i) it is possible to design a gossiping algorithm that reliably maintains a copy of communal state at each peer yet requires only a modest amount of bandwidth, (ii) our content search and retrieval algorithm tracks the performance of the original ranking algorithm very closely, giving P2P communities a search and retrieval algorithm as good as that possible assuming a centralized server, and (iii) PlanetP's gossiping and search and retrieval algorithms both scale well to communities of at least several thousand peers.*

1 Introduction

We present PlanetP, a peer-to-peer (P2P) content search and retrieval infrastructure targeted to communities wishing to share large sets of text documents such as scientific publications, news articles, legal documents, etc. P2P computing, where communal resources are provided directly by members of a community, is an attractive model for information sharing between *ad hoc* groups of users because of its low cost of entry and natural scaling model. Any two users wishing to share information can form a P2P community using their existing computing resources. As individuals join the community, they will bring more resources with them, allowing the community to grow naturally. Measurements of one such community at Rutgers show over 500 users sharing over 6TB of data; open communities such as Gnutella [15] have achieved much greater sizes [29].

The value of an information sharing community is often directly proportional to its size: larger communities provide more information to the individual users and so provide greater value. As communities grow, however, locating information becomes a critical challenge. We designed PlanetP specifically to meet this challenge. Unlike many existing P2P systems that focus on providing an efficient and extremely scalable name-based object location service [24, 23, 30, 27], our design centers around an indexing and content search and retrieval core. This design is motivated by the success of the Internet search engines, which argues that content addressing is an intuitive paradigm for users to manage and access large volumes of information.

By targeting the P2P model, however, PlanetP must face constraints not applicable to the current Internet search engines. These include: (a) there is no centralized administration, management, or coordination, (b) communal resources may fluctuate rapidly and un-

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

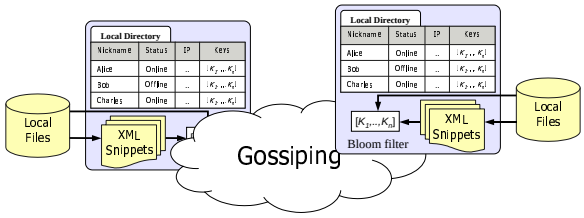


Figure 1: *A PlanetP community is a dynamic set of peers wishing to share a set of documents distributed across the peers' local data stores. Peers continually gossip to help each other maintain a local copy of the global directory for content search and retrieval.*

predictably as the presence of peers online is uncontrolled and unpredictable, and (c) resources are typically fragmented and potentially spread across wide geographic areas. These constraints mean that PlanetP cannot employ the current model practiced by all Internet search engines: crawl the information sources, bring the shared information to a centralized location with massive computing power, and allow clients to query the centralized information repository.

PlanetP instead takes the opposite approach of replicating a compact summary of the entire index at every peer using gossiping [8]. More specifically, all members of a PlanetP community agree to continually gossip about changes in the community to help each other maintain a local copy of a global directory (Figure 1). This directory contains the names and addresses of all current members, as well as a Bloom filter [1] per member that summarizes the set of terms contained in the documents being shared by that member. Each member uses its copy of the directory to query against and retrieve matching documents from the collective information store of the community.

Additionally, peers can optionally choose to implement an information brokerage service based on consistent hashing [20] to optimize the location of new or rapidly changing data. For example, when a document is first introduced to the community, the publishing peer may wish to publicize this fact using the information brokerage service so that other peers can find the document without waiting until a new Bloom filter has been generated and gossiped everywhere. Note, however, that unlike systems such as Chord [30], PlanetP's information brokerage service only serves to optimize performance. *All of PlanetP's services critically depend on gossiping, not the information brokerage service.* We made this explicit design decision because gossiping is well suited to the P2P environment, where peers may come and go freely; gossiping is never deterred by the abrupt leaving or absence of any subset of peers.

Finally, to help users better navigate large sets of documents, PlanetP implements a search and retrieval engine that is based on a state-of-the-art text ranking algorithm. A naive implementation of this ranking algorithm would require each peer to have the inverted index of the entire community, which is costly both in terms of bandwidth and storage. Instead, we show how this algorithm can be approximated given PlanetP's Bloom filter summaries of peers' inverted indexes gathered at each peer.

In designing and prototyping PlanetP, we make the following contributions: (1) We show how a content search and retrieval engine approximating a state-of-the-art text ranking algorithm can be built in the specific context of P2P computing, (2) We show that gossiping is an appropriate mechanism for replicating information across a P2P community, (3) Using five benchmark collections from Smart [3] and TREC [19], we show that our search and retrieval algorithm matches the performance of the original ranking algorithm, despite the accuracy that it gives up by using only a compact summary of each individual's inverted index. (4) We show that PlanetP's gossiping algorithm and content search and retrieval algorithm both scale well to communities of at least several thousand peers[1].

2 Local Data Stores and Bloom Filters

PlanetP maintains a local data store at each peer of an information sharing community. PlanetP assumes that the basic unit of storage is an XML document, allowing it to index arbitrary data for search and retrieval, regardless of the applications used to create and manipulate the data. A peer *publishes* an XML document to PlanetP when it wishes to share the document. Each published XML document contains text and possibly links (XPointers) to external files. PlanetP indexes any text

[1]While much of our simulation data suggest that PlanetP could scale well beyond several thousand peers, there are two concerns. First, if we admit users with only modem-speed connectivity, it is difficult for them to download the index summary and global directory when they first join the community. Second, simultaneously joining of a large number of new members with large indexes to share can require large aggregate gossiping network volume.

in a published XML document[2] as well as linked external files if they are of a known type (e.g., postscript, PDF, or text), thus providing backward compatibility for peers to share and search for non-XML documents. Each published XML document is stored in the local data store of the publishing peer; external files are not stored by PlanetP.

PlanetP stores the terms extracted from published documents in a local inverted index. PlanetP summarizes the inverted index of each peer using a Bloom filter [1] and, as shall be seen below, diffuses this summary across the entire community to support communal content search and retrieval. 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 inverted index. The filter is computed by obtaining n indices for each term in the set, typically via n different hashing functions, and setting the bit at each index to 1. Then, given a Bloom filter, we can ask, is some term x a member of the set by computing n indices for x and checking whether those bits are 1. Bloom filters can give *false positive* but never *false negative*. Thus, given a set of Bloom filters summarizing peers' inverted indexes, we can easily compute the subset of peers that may contain documents relevant to a given query.

We choose to use Bloom filters because they provide four important advantages: (1) The Bloom filter is a relatively efficient summary mechanism; for example, to support a false positive rate of less than 5% when using two hash functions, we would need 1.9 KB to summarize 1000 terms and 34 KB for 50,000 terms. (2) Previous studies of file systems have shown that a majority of files change very slowly [26, 10]. If P2P information collections display the same characteristic, then, using Bloom filters, PlanetP will place very little load on the community for searches against this bulk of slowly changing data. (3) Peers can independently trade-off accuracy for storage. For example, a peer a may choose to combine the filters of several peers 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).

[2]Currently, XML tags are indexed simply as normal terms. We will extend PlanetP to make use of the structure provided by XML tags in the near future.

(4) A peer can know that documents relevant to a query might exist on peers that are currently off-line. Thus, instead of missing these documents as in current systems, the searching peer could arrange to rendezvous with the off-line peers when they reconnect to obtain the needed information.

3 Gossiping

At the heart of PlanetP is its gossiping algorithm: PlanetP uses gossiping to replicate a global directory that includes the list of peers, their IP addresses, and their Bloom filters at each peer in the community. 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.

PlanetP's gossiping algorithm is a novel combination of *rumor mongering* and *anti-entropy* as previously introduced by Demers et al. [8] together with a *partial anti-entropy* algorithm that we found improved performance significantly for dynamic P2P environments. Demers et al.'s algorithm works as follows. Suppose that a peer x learns of a change to the directory (e.g., it just updated its Bloom filter). Every T_g seconds, x randomly chooses a target peer y believed to be currently on-line and attempts to tell y of the change. If y has not heard of this change, it records the new information and then itself attempts to spread the rumor as x is doing. If x contacts n peers in a row that already knows about the change, it stops spreading the rumor. Because this rumoring process can leave a residual set of peers that do not hear about a rumor before it dies out, every so often, each peer performs an anti-entropy operation instead of rumoring. For example, in our implementation of this algorithm, every tenth round of rumoring (or if

there's currently no new information to be rumored), a peer x would send an anti-entropy message instead of a rumor. The anti-entropy message asks the target y to send a summary of its entire directory to x. When x gets y's summary, x parses it to see whether y has more updated information. If so, then x asks y for the needed information. This combination of push rumoring and pull anti-entropy helps to reliably spread new information everywhere.

In a dynamic P2P environment, however, where rumors may arrive often because of peers' leaving and coming, the time required to spread a particular rumor everywhere can become highly variable even with the above combined algorithm. This is because the high arrival rate of new rumors forces the rate of anti-entropy to the minimum: every tenth round. If a peer is unlucky enough to contact another peer that is also missing a particular rumor, then it may be several tens of rounds before the rumor reaches everyone. To fix this, we can increase the frequency of performing anti-entropy, say to every other round or every fifth round. Unfortunately, anti-entropy is much more expensive than rumoring because a summary of the entire directory must be sent. Thus, we would be expending much more bandwidth, reducing the efficiency of gossiping.

Instead, we decided to extend each rumoring round with a partial anti-entropy exchange, which works as follows. When x sends a rumor message 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. x can then check to see whether it is missing something that y recently learned about and pull that information from y. This partial anti-entropy requires only one extra message *in the case that y does know something that x does not*. Further, the amount of data piggybacked on y's message is very small, in order of tens of bytes. We have found the inclusion of this partial anti-entropy step to significantly reduced the variation in the time required to inform the entire community about a particular new piece of information as well as requiring much less bandwidth than if we performed anti-entropy more often.

Finally, PlanetP currently uses a base gossiping interval T_g of 30 seconds to accommodate peers with relatively slow communication links. PlanetP dynamically adjusts this base interval to reduce bandwidth usage when the system has reached a stable configuration. When a peer x does not have any rumor to spread, it maintains a count p of the number of times it has contacted a peer that has the same directory as it does [8]. Whenever this count reaches a *gossip-less threshold*, currently set to 2, x increases its gossiping interval by a *slow down constant*, currently set to 5 seconds[3]. After increasing its gossiping interval, x resets p to zero and continues. In this manner, the gossip interval can gradually increase to a maximum of 2 minutes. On the other hand, whenever x receives a rumor message or finds a new piece of information through anti-entropy, it immediately resets its gossiping interval to the default in order to efficiently diffuse this new information.

Dynamically adapting the gossiping interval has two advantages. First, we do not need to define a termination condition given the probabilistic nature of the algorithm. Second, when global consistency has been achieved, the bandwidth use is negligible after a short time.

4 Information Brokerage

While gossiping is an elegant solution for maintaining a copy of the global directory at each peer, enabling peers to accurately search the entire communal data store, it does have two disadvantages: (1) new or rapidly changing information spreads slowly (e.g., around 10 minutes in communities of several thousands peers with our current settings), and (2) information is always diffuse everywhere, even if it is relevant to only a small subset of the community. To address these limitations, peers in PlanetP can also choose to implement an information brokerage service that uses consistent hashing [20] to publish and locate information. As already mentioned, however, this service is an optimization rather than a necessary part of PlanetP. In particular, this service makes no guarantee as to the safety of information published to it. If a member leaves abruptly without passing on its portion of the published data, that data will be lost. (We show how this unreliable service can be a useful optimization when discussing PlanetFS, a semantic file system that we are building using PlanetP, in Section 6.)

PlanetP's information brokerage service works as follows. Information is published to the brokerage ser-

[3]The various constants/parameters we use were found to work well in our current simulation but can be tuned as needed for any particular community.

vice as an XML snippet with a set of associated keys (terms) and a discard time. The network of brokers use consistent hashing [20] to partition the key space among them. In particular, each active member chooses a unique broker ID from a predetermined range (0 to $maxID$). Then, all members arrange themselves into a ring using their IDs. To map a key to a broker, we compute the hash H of the key. Then, we send the snippet and key to the broker whose ID makes it the least successor to $H \ mod \ maxID$ on the ring. The snippet is discarded after its discard time expires.

The real complexity of implementing this service lies in handling the dynamic joining and leaving of members. Because of space constraints, however, and the fact that this service is not central to this paper, we refer the interested reader to a longer technical report [7] for the details of this part of the implementation.

5 Content Search and Retrieval

PlanetP supports two types of searches: an *exhaustive* search that we expect to mostly be used by applications written on top of PlanetP, and a selective search more suitable to user-initiated searches that uses the vector space ranking model to choose the subset of documents most relevant to a query.

5.1 Exhaustive Search

To exhaustively search a PlanetP data store, an application poses a query represented as a conjunction of keys separated by white spaces. When presented with this query, PlanetP searches the Bloom filters in its local directory to obtain a list of candidate peers that may have documents matching the query. Then, PlanetP contacts these candidate peers as well as the appropriate brokers to retrieve matching XML documents. Once all contacted peers have replied, the set of retrieved documents is returned to the caller.

Persistent Queries. PlanetP also supports *persistent* queries for exhaustive searches. Persistent queries allow peers to specify interest in new information entering the system without having to constantly poll as well as providing a way for applications to implement traditional distributed mechanisms like condition variables, publish/subscribe communication, tuple spaces, etc. When posting a persistent query, the poster provides an object that will be invoked whenever a new matching snippet is found, either when a new Bloom filter is received or a new snippet is published to the brokers.

5.2 Vector Space Ranking

To provide a more selective search that would help users better navigate large sets of documents, we have implemented a distributed ranking algorithm based on the vector space model [31], a state-of-the-art text-based ranking algorithm. In this section, we first give a brief description of the vector space ranking model, then discuss the changes that we have introduced to adapt this algorithm to PlanetP.

Background. 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:

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

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 [28]. In our work, we adopt the following system of equations as suggested by Witten et al. [31]:

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

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

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

$$Sim(Q, D) = \frac{\sum_{t \epsilon Q} w_{D,t} \times IDF_t}{\sqrt{|D|}} \quad (2)$$

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

PlanetP. We cannot implement the above relevance ranking directly in PlanetP because we do not have all the necessary information. Instead, we approximate this function by breaking the ranking problem into two sub-problems: (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 the Bloom filters collected at each peer: N is the number of Bloom filters, N_t is the number of hits for term t against these Bloom filters.

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

$$R_i(Q) = \sum_{t \epsilon Q \wedge t \epsilon BF_i} IPF_t \quad (3)$$

which is simply a weighted sum over all query terms contained by peer i, weighted by how useful that term is to differentiate between peers; t is a term, Q is the query, and BF_i is the set of terms represented by the Bloom filter of peer i, and R_i is the resulting relevance of peer i 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. Thus, once we have established a relevance ordering of peers for a query, we must then decide how many of them to contact. To address this problem, we first assume that the user specifies an upper limit k on the number of documents that should be returned in response to a query. Then, a simple solution to the selection problem would be to contact the peers one by one, in the order of their relevance ranking, until we have retrieved k documents.

Unfortunately, this obvious approach leads to terrible retrieval performance [6]. To address this problem, we introduce the following heuristic for adaptively determining a stopping point. Given a relevance ordering of peers, contact them one-by-one from top to bottom. Maintain a relevance ordering of the documents returned using equation 2 with IPF_t substituted for IDF_t.

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 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 7), we propose the following function for p

$$p = \left\lfloor 2 + \frac{N}{300} \right\rfloor + 2 \left\lfloor \frac{k}{50} \right\rfloor \quad (4)$$

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.

6 PFS: An Example Application

We have implemented PFS, a personal semantic file system, to show how the various components of PlanetP work together to provide a useful infrastructure for information sharing. PFS provides similar functionality to the semantic file system defined by Gifford et al. [14]. PFS's novelty lies in the fact that it supports content querying across a dynamic community of users without requiring centralized indexing. PFS does not manage storage directly. Files are stored using the local file system of each peer; files to be shared with the community are *published* to PFS, which then uses PlanetP to make it possible for the entire community to search for these files based on its content.

Each user uses PFS to share documents and to create a personal semantic namespace over the set of shared documents. Currently, each namespace is private to a single user. Like the semantic file system, a directory is created in PFS whenever the user poses a query. PFS creates links to files that match the query in the resulting directory. If a file is created or modified such that it matches some query, PFS will update the directory to have a link to this file. Building a query-based subdirectory is equivalent to refining the query of the containing directory.

PFS is comprised of three components: Explorer, File Server, and PFS Core. The Explorer provides a GUI interface to the user. The File Server is a very simple web server that provides two functions: (a) return a URL when given a local pathname, (b) return the content of the appropriate file in response to a GET operation.

When the user publishes a file, PFS (Core) obtains a URL for that file from the File Server. PFS then embeds this URL and a pointer to the file in a XML snippet and publishes it to PlanetP, which automatically indexes the file. PFS also asks PlanetP to publish the XML snippet on the information brokerage service using the 10% most frequently appearing terms in the file with a discard time of 10 minutes. All terms are automatically summarized in the Bloom filter. This dual publication allow peers to find a file in a very short time after publication if they are searching for one of the key that appears most often in the file. For other keys, the newly published file can only be found when the new Bloom filter has been computed and diffused throughout the system.

When the user creates a directory, PFS poses its name as a persistent exhaustive query to PlanetP. File names are extracted from snippets returned by PlanetP and entered in to the directory. PFS automatically updates directories for addition via PlanetP's persistent query upcalls. Updates for removal—that is, when a file is deleted by its owner or modified to no longer map to the directory's query—is more difficult. Whenever the user opens a directory, PFS checks the last time that the directory was updated. If this time is greater than a fixed threshold, PFS reruns the entire query to get rid of stale files.

Given PlanetP, we were able to implement PFS in less than two weeks, with much of the first week given to designing PFS graphical interface.

7 PlanetP Performance

We now turn to evaluating PlanetP's performance, concentrating on PlanetP's gossiping algorithm and content search and retrieval engine. We refer the interested reader to [7] for the performance of the information brokerage service. We begin by running a number of micro benchmarks against our prototype implementation to give an idea of the cost of basic PlanetP operations; the prototype is written entirely in Java and currently stands at around 7000 lines of code. Then, we study the scalability of PlanetP's gossiping algorithm, both in terms of the time required to distribute information as well as the required bandwidth. Finally, we study the effectiveness of PlanetP's content search and retrieval algorithm using a number of benchmark data collections.

7.1 Micro Benchmarks

We start by measuring the costs of PlanetP's basic operations: the manipulation of Bloom filters and managing the inverted index of the local data store at each peer. Table 1 lists these operations and their costs 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. We observe that while we have not optimized our implementation at all—for ease of implementation, most data structures were implemented using the Java Collections Framework—the costs after JIT compilation is quite reasonable. For example, it takes only

Operation	Cost before JIT (ms)	Cost after JIT (ms)
Bloom filter insertion	17 + (0.111 * no. keys)	4 + (0.011 * no. keys)
Bloom filter search	0.107 * no. keys	0.010 * no. keys
Bloom filter compress	411 + (0.016 * no. keys in filter)	21 + (0.001 * no. keys in filter)
Bloom filter decompress	0.028 * no. keys in filter	0.005 * no. keys in filter
Insertion into inverted index	5 + (0.137 * no. keys)	14 + (0.024 * no. keys)
Search inverted index	0.024 + (0.001 * no. keys)	0.002 + (0.0001 * no. keys)

Table 1: *Costs of PlanetP's basic operations. Each cost is presented as a fixed overhead plus a marginal per key overhead or just the latter; for example, the cost for inserting n keys into a Bloom filter is $4ms + 0.011n$.*

Operation	Cost (ms)
CPU gossiping time	5ms
Base gossiping interval	30sec
Max gossiping interval	60sec
Network BW	56Kb/s to 45Mb/s
Message header size	3 bytes
1000 keys BF	3000 bytes
20000 keys BF	16000 bytes
BF summary	6 bytes
Peer summary	48 bytes

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

about 1/2 a second to create a Bloom filter for 50,000 terms. It takes only 50 ms to search for a query with five terms across 1000 Bloom filters.

We currently compress Bloom filters to reduce the gossiping bandwidth because we are using constant size (50 KB) Bloom filters for ease of implementation. The chosen size let us summarize up to 50,000 terms with less than 5% error. Our compression scheme is a run-length compression that uses Golomb codes [16] to encode runs, which outperforms gzip in our specific context. Table 1 shows that decompression, which occur much more frequently, is reasonably efficient in time. In the future, we will almost certainly move to variable size filters and so the compression may no longer be needed.

7.2 Gossiping

We have built a simulator to assess the reliability and scalability of PlanetP's gossiping algorithm. We used measurements of our prototype to parameterize the simulator; Table 2 list 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.

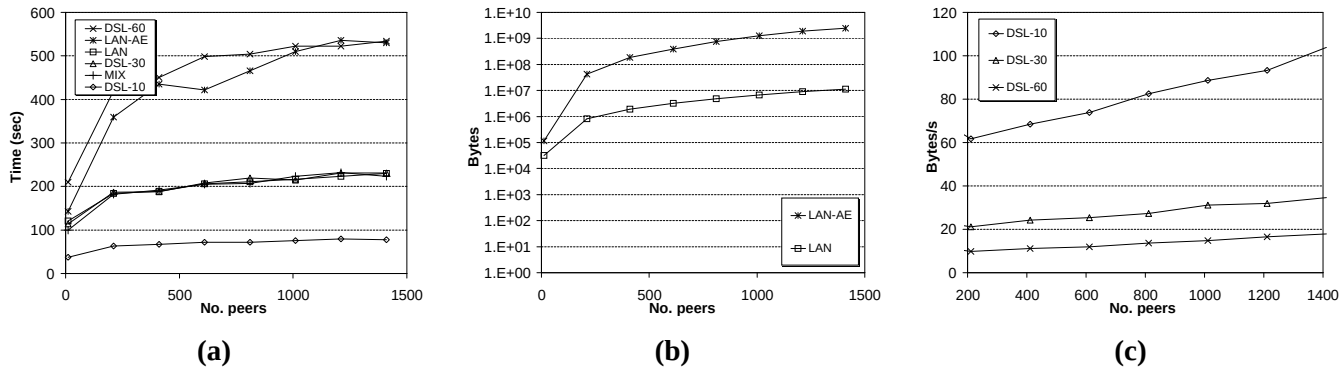
Propagating new information. We begin by studying the time required to gossip a new Bloom filter summarizing 1000 terms throughout stable communities of various sizes. We are using 1000 words because PlanetP sends diffs of the Bloom filters to save bandwidth; thus, this scenario simulates the addition of 1000 new terms to some peer's inverted index. 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.

Figure 2(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 [18], Bayou [9] and Deno [22]. In fact, our first gossiping algorithm was a push/pull anti-entropy based on these previous works. We quickly realized, however, that the required bandwidth grew rapidly with community size and so moved more toward rumor mongering.

DSL-10,30,60 Peers are connected by 512 Kbps links.



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

Figure 2: *(a) Time, (b) aggregated network volume, and (c) average per-peer bandwidth required to propagate a single Bloom filter containing 1000 keys everywhere vs. community size.*

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. [29], 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 2(b) shows the aggregate network volume used to propagate the new piece of information throughout the community. Figure 2(c) 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 a log function of community size, 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 230 for a community with 1500 peers. For DSL-30, we have continued the simulation for community sizes of up to 5000. At 5000, the propagation time is still only 250 seconds. (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 keys only requires a total of 11 MB to be sent, leading to a per-peer average bandwidth requirement of less than 40 B/s 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. 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 new members join an established community. We perform a slightly different experiment than the previous one; performing the same experiment would lead to very similar conclusions except that the required network volume (and so bandwidth) would be somewhat higher if we assume that a new member would be propagating a Bloom filter representing more than 1000 keys. Instead, 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 keys with the rest of the community through their Bloom filters.

Figure 3 plots the time to reach consistency vs. the

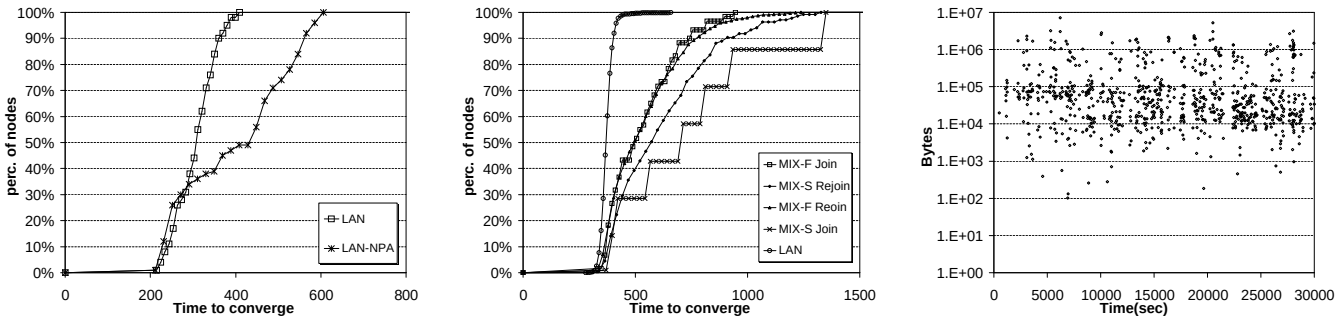


Figure 4: *Time required to complete gossiping about dynamic events for (a) a set of peers joining the community, and (b) normal operation of a dynamic community, with peers coming and going; LAN-NPA is our gossiping algorithm without the partial anti-entropy component; Join refers to when a peer arrives back online, wishing to share 1000 new keys; Rejoin refers to when a peer arrives back online with no new information to share. (c) Aggregated gossiping bandwidth per second for (b).*

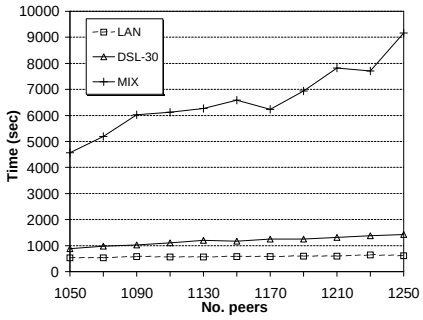


Figure 3: *Time required for $x - 1000$ peers to simultaneously join the community of 1000 stable online peers, each wishing to share 20000 keys.*

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 keys from the existing community. Also, having 250 members join at once means that 250 Bloom filters representing 5 million keys 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. In particular, when they first join, the time to download the entire directory would 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 for 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 others and peers with slower connectivity to preferentially gossip with each others. 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 guy (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 how well PlanetP's gossiping performs in a dynamic community of rejoining and leaving peers. 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 online peers; 100 peers join the community according to a Poisson process with an average inter-arrival rate of once every 90 seconds. Figure 4(a) plots the cumulative percentage of events against the convergence time (how long before an arrival event is known to everyone in the 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 total membership of the community is 1000 members. 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. 5% of the time, when a peer rejoins the online community, it has 1000 new keys. These parameters were again based roughly on measurements reported by Saroiu et al. [29] (except for the number of new keys being shared occasionally) and are meant to be representative of real communities.

Figure 4(b) plots the cumulative percentage of events against the convergence time. We observe that with sufficient bandwidth, convergence time is very tight

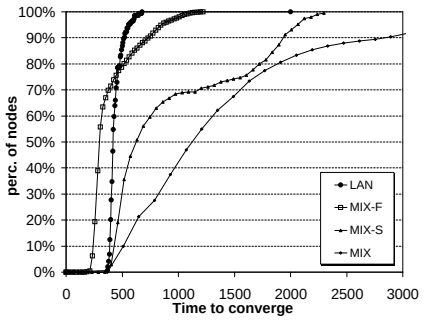


Figure 5: *Convergence time for a dynamic community with 2000 members.*

around 400 seconds. For the MIX community, our bandwidth aware gossiping algorithm allows peers with fast connections to learn about events much more expeditiously than peers with slow connections. Even for peers with fast connections, however, the convergence times are much more variable. This is because fast peers sometime still have to talk to slow peers and so their progress may be impeded. When looking closer into the fast nodes convergence time, we found that up 70% of the nodes were able to see all other fast nodes as quickly as in the LAN case.

Figure 4(c) plots the aggregate bandwidth against time. This graph shows that the normal operation of a community requires very little bandwidth, ranging from between 100 KB/s to 1 MB/s across the entire community.

Finally, Figure 5 plots the cumulative percentage of events against convergence time for a dynamic community of 2000 peers. LAN and MIX are as before. MIX-F gives convergence time for joins and rejoins of fast peers with the convergence condition being that only fast peers need to learn about the event. MIX-S gives convergence time for joins and rejoins of slow peers with the same convergence condition. Observe that our bandwidth aware gossiping algorithm allow fast peers to learn about new events quite efficiently. At the same time, it does not harm the slow peers beyond the fact that they will be slow anyway because of their bandwidth limitations.

Summary. Results across the above set of experiments suggest that gossiping works reliably and scales well to the range of several thousand peers. The two concerns for scaling beyond this level is the time required to download the entire set of Bloom filters when

a peer first joins a community if that peer is bandwidth limited, and the aggregate bandwidth required for new members to join the community.

7.3 Search efficiency

We now turn to assessing the performance of PlanetP's search and retrieval engine. We measure performance using two accepted metrics, *recall* (R) and *precision* (P), which 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 and not a single irrelevant one. If we did this, we would obtain a 100% recall and 100% precision. Also ideally, we would want to contact as few peers as possible to achieve 100% recall and 100% precision.

We assess PlanetP's performance by comparing its achieved recall and precision against the original TFxIDF algorithm. If we can match the TFxIDF's performance, then we can be confident that PlanetP provides state-of-the-art search and retrieval capabilities[4], despite the accuracy that it gives up by gossiping Bloom filters rather than the entire inverted index.

We use five collections of documents (and associated queries and human relevance ranking) to measure PlanetP's performance; Table 3 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 are comprised of small fragments of text and summaries and so are relatively small in size. The last collection, AP89, was extracted from the TREC collection [19] and includes full articles from Associated Press published in 1989.

To measure PlanetP's recall and precision on the above collections, we built a simulator that first distributes documents across a set of virtual peers and then runs and evaluates different search and retrieval algorithms. The distribution of documents on our simulation follows a Weibull function, which is motivated by observing current P2P file-sharing communities. (In [6], we also study a uniform distribution and show that PlanetP does equally well although it has to contact more peers as documents are more spread out in the community.) To compare PlanetP with TFxIDF, 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 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, TFxIDF and TFxIPF, the 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 conflate words to their root (e.g. **running** becomes **run**).

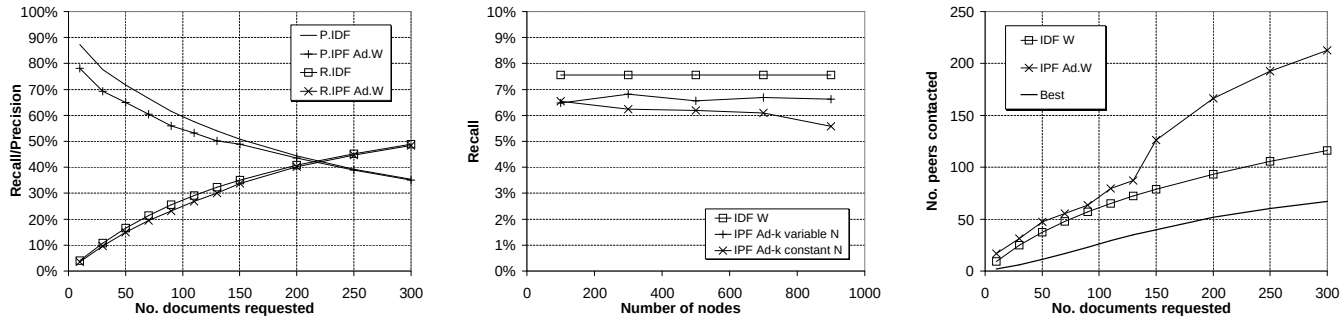
Figure 6(a) plots TFxIDF's and PlanetP's average recall and precision over all provided queries as functions of k for the AP89 collection. We only show results for this collection to save space; these results are representative for all collections. We refer the reader to our web site, http://www.panic-lab.rutgers.edu/, for results for all collections. Figure 6(b) plots PlanetP's recall against community size for a constant k of 20. Finally, Figure 6(c) plots the number of peers contacted against k.

We make several observations. First, using TFxIPF and our adaptive stopping condition, PlanetP tracks the performance of TFxIDF closely. It performs slightly worse than TFxIDF for $k < 150$ but catches up for larger k's. In fact, for some of the collections, TFxIPF slightly outperforms TFxIDF at large k's. While the performance difference is negligible, it is interesting to consider how TFxIPF can outperform TFxIDF; this is possible since TFxIDF is not always correct. In this case, TFxIPF is finding lower ranked documents that were determined to be relevant to queries, while some of the highly ranked documents returned by TFxIDF, but not TFxIPF, were not relevant.

[4] when only using the textual content of documents, as compared to link analysis as is done by Google and other web search engines [2]

Trace	Queries	Documents	Number of words	Collection size (MBs)
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 3: *Characteristic of the collections used to evaluate PlanetP search and retrieval capabilities.*



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

Figure 6: *(a) Average recall (R) and precision (P) for the AP89 collection distributed among 400 peers. IDF is TFxIDF. IPF Ad.W is TFxIPF with the adaptive stopping heuristic on the Weibull distribution of documents. (b) PlanetP's recall as a function of community size for a fix k of 20. (c) Number of peers contacted when requesting different numbers of documents (k). IPF Ad.W is TFxIPF with the adaptive stopping heuristic. Best is the minimum number of nodes that can be contacted to retrieve k documents using the relevance judgments.*

Second, PlanetP scales well, maintaining a constant recall and precision for communities of up to 1000 peers (not shown here to save space). We have not study scalability beyond that point because the traces are not sufficiently large.

Finally, PlanetP's adaptive stopping heuristic is critical to its performance. As k and/or the number of peers grow, PlanetP must contact more peers to achieve good recall and precision. The adaptive stopping heuristic allows PlanetP to do this efficiently. The fact that PlanetP starts to contact many more peers at $k > 150$, see Figure 6(c), may be an indication that the linear dependency of our stopping heuristic on k may be too aggressive. PlanetP probably does not need to contact this many peers as, at this point, PlanetP's recall catches up to TFxIDF and in fact, outperforms it slightly for some of the other collections.

8 Related Work

While current P2P systems such as Napster [24], Gnutella [15], and KaZaA [21] have been tremendously successful for music and video sharing communities, their search engines have been frustratingly limited. Our goal for PlanetP is to increase the power with which users can locate information in P2P communities. Also, we have focused more tightly on text-based information, which is more appropriate for collections of scientific documents, legal documents, inventory databases, etc.

In contrast to existing systems, recent research efforts in P2P seek to provide the illusion of having a global hash table shared by all members of the community. Frameworks like Tapestry [32], Pastry [27], Chord [30] and CAN [25] 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 from the community. Also, the entire community collaborate to spread information about what each peer has to share, instead of putting the publishing burden entirely on the sharing peer. Second, we have focused on content search and retrieval, attempting to provide a similar service to web search engines, which none of these systems have explored.

In addition to P2P systems, PlanetP also resembles previous work done on tuple spaces [13] and publisher-subscriber models [11]. A discussion of this body of related work, however, is beyond the scope of this paper. More recently system like Herald [4] have started to study self organizing solutions in environments similar to ours. They have proposed building a publish/subscribe system by using replicated servers on P2P networks.

More related to PlanetP's information retrieval goals, Cori [5] and Gloss [17] address the problems of database selection and ranking fusion on distributed collections. Recent studies done by French et al. [12] show that both scale well to 900 nodes. Although they are based on different ranking techniques, the two rely on similar collection statistics. In both cases the amount of information used to rank nodes is significantly smaller than having a global inverted index. Gloss needs only 2% of the space used by a global index. Both Gloss and Cori assume the existence of a server (or a hierarchy of servers) that will be available for users to decide which collections to contact. In PlanetP we want to empower peers to work autonomously and therefore we distribute Bloom filters widely so they can answer queries even on the presence of network and node failures.

9 Conclusions

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 communal 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 decision to support an effective P2P search engine. 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. A naive implementation of TFxIDF would require each peer in a community to have access to the inverted index of the entire community. Instead, we show how TFxIDF can be approximated given a compact summary (the Bloom filter) of each peer's inverted index.

We make four contributions: (1) We show how a content search and retrieval engine approximating a state-of-the-art text ranking algorithm can be built in the specific context of P2P computing, (2) We show that gossiping is an appropriate mechanism for replicating information across a P2P community, (3) We show that our search and retrieval algorithm matches the performance of TFxIDF, giving P2P communities a search and retrieval algorithm as good as that possible assuming a centralized server. (4) We show that PlanetP's gossiping algorithm and content search and retrieval algorithm both scale well to communities of at least several thousand peers.

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] L. Cabrera, M. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.

[5] 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.

[6] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. Technical Report DCS-TR-483, Department of Computer Science, Rutgers University, Apr. 2002.

[7] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Infrastructure Support for P2P Information Sharing. Technical Report DCS-TR-465, Department of Computer Science, Rutgers University, Nov. 2001.

[8] 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.

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

[10] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[11] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. Technical Report DSC ID:2000104, EPFL, 2001.

[12] 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.

[13] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[14] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr. Semantic File Systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991.

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

[16] S. W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12(3):399–401, 1966.

[17] 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.

[18] 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.

[19] 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.

[20] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, 1997.

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

[22] P. Keleher and U. Cetintemel. Consistency management in deno. To appear in The Journal on Special Topics in Mobile Networking and Applications (MONET).

[23] 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.

[24] Napster. http://www.napster.com.

[25] 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.

[26] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In Proceedings of the 2000 USENIX Annual Technical Conference, June 2000.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.