PlanetP: Using Gossiping and Random Replication to Support Reliable Peer-to-Peer Content Search and Retrieval*

Francisco Matias Cuenca-Acuna, Richard P. Martin, Thu D. Nguyen
{*mcuenca, rmartin, tdnguyen*}*@cs.rutgers.edu*

Technical Report DCS-TR-494
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854

July 9, 2002

Abstract. *We introduce the PlanetP system, which explores the construction of a reliable peer-to-peer (P2P) content search and retrieval service using randomly circulated global state between peers of an unstructured community. Our work represents a novel alternative approach to recent P2P systems that focus on enabling very large-scale name-based object location using sophisticated distributed data structures. We show that our simpler approach scales to several thousand peers (ultimately targeting the regime of about ten thousand) and converges in several minutes using only modest bandwidth while still maintaining reliable search, ranking, and retrieval similar to an Internet search engine. Unlike current search engines or other P2P systems, however, PlanetP does not require centralized directories or management, nor builds a complex distributed data structure. PlanetP achieves its goals using three major components. First, peers collaborate to maintain local copies of the global membership directory along with compact summaries of shared content using a randomized gossiping algorithm. Second, peers implements a per query, text-based ranking algorithm to help users ignore irrelevant documents. Finally, peers collaborate to replicate unpopular content—popular content is naturally highly replicated via hoarding—using Reed-Solomon erasure coding to increase the probability of successful retrievals.*

1 Introduction

Recent peer-to-peer (P2P) systems often focus on sharing information across very large communities. To meet their scaling targets of millions, or even billions, of users [8], these systems construct and maintain large *distributed data structures*, such as hash tables, across the entire community to support *name-based object location* [9, 15, 12, 14]. A significant challenge for these systems is maintaining the correctness and consistency of their global data structures as peers independently and unpredictably join and leave the system. Further, these mapping schemes must load balance across peers with potentially widely different resources.

In this paper, we propose a novel alternative approach: maintain global state by using gossiping and random replication for *unstructured* communities of several thousand, perhaps up to ten thousand, peers. In addition, we show how to leverage the same infrastructure to provide a powerful and reliable *content search and retrieval* engine to enable a content-addressable model similar to current Internet search engines[1]. We believe that this scale will be applicable to many information sharing communities such as groups of researchers sharing publications, doctors sharing medical analyses, and students sharing music and videos. Further, the success of the Internet search engines argue that content addressing is a natural and powerful model for locating information in large data

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

[1] Gnutella [6] is an example of a current P2P system that is also unstructured. However, systems such as Gnutella do not scale well [7] because they flood on each query; in addition they do not support content addressing. While the current Gnutella community exceeds our target size of ten thousand users, its success critically depends on a pattern of music and video sharing where a small subset of content is very popular and so is highly replicated. This allows Gnutella to successfully locate information while employing its inefficient flood-based search algorithm.

stores.

We realize this approach in a prototype system called PlanetP. We show it to be a simple, yet powerful system for sharing information. PlanetP is simple because each peer must only agree to perform a periodic, randomized, gossiping-style of exchange with other peers, rather than collaborate to correctly and consistently maintain a global data structure. In addition, each peer must also agree to provide some excess local storage for replication of data to enable high-availability. PlanetP is powerful because it maintains a view similar to a search engine: a global, coherent, and content-ranked data store with high degree of availability without depending on centralized resources and the online presence of specific peers.

To address the problem of locating and ranking documents, PlanetP replicates both a global membership directory and a compact summary of each peer's shared content at every peer using a randomized gossiping algorithm [5]. The directory contains the names and addresses of all current members. The content summary, or index, takes the form of a Bloom filter [1] per peer that summarizes the set of terms contained in the documents being shared by that peer. To keep both the directory and index updated and minimally consistent across the community, all members agree to continually gossip about changes in the community. We explicitly chose gossiping because the randomness built into gossiping is well suited to the P2P environment, where peers may come and go freely. Further, gossiping is never deterred by the abrupt leaving or absence of any subset of peers.

To aid the user in locating relevant information, we have implemented a ranking algorithm that approximates a state-of-the-art text-based ranking algorithm [2]. We do not discuss the algorithm further here as the focus of this paper is on gossiping and our intended approach to replication for reliable retrieval. However, the ranking algorithm affects the experiments that we use to evaluate the effectiveness of gossiping and replication.

Finally, to address availability, PlanetP attempts to maintain at least one copy of each shared file online at all times. In the future, we may relax this constraint somewhat; currently, however, this provides a succinct goal to work toward as we explore our proposed approach to replication. Occasionally, each peer randomly chooses a file from its local store and estimates its availability. A file may already be highly available because of hoarding—peers tend to download popular files to their local stores—or because some other peer has actively replicated the file. In this case, the peer does nothing. Otherwise, the peer actively replicates the file by using a variation of the Reed Solomon erasure coding to compute k fragments of a file and pushes these fragments to peers with sufficient storage. The use of this coding brings two advantages. First, less space is required than complete file replication for the same availability. Second, nodes can decide whether to actively add fragments based solely on the number of fragments available rather than precisely which fragments are available. This latter property is critical in that it maintains our *unstructured* approach.

In the remainder of the paper, we first discuss how information is shared and indexed in PlanetP (Section 2). Then, we describe our gossiping algorithm and evaluate its performance (Section 3). Finally, we outline our approach to randomized replication to support reliable retrieval of information and present remaining open questions (Section 4).

2 Data Stores and Bloom Filters

At each peer of an information sharing community, PlanetP (i) maintains a local data store of shared files, (ii) indexes the text of each file added to the local data store and maintains a word-to-document inverted index to support content search and retrieval, and (iii) summarizes the inverted index of each peer using a Bloom filter [1]. Bloom filters can give a *false positive* for a lookup but never *false negative*. Thus, given a set of Bloom filters summarizing peers' inverted indexes, we can probabilistically compute the subset of peers containing documents relevant to a given query. We choose to use Bloom filters because they provide a number of distinct benefits; we refer the readers to [3] if they are interested in this discussion.

3 Locating Relevant Information

As already mentioned, PlanetP's approach to supporting the reliable location of relevant information is to replicate a global membership directory and Bloom filter summaries of peers' indexes at each peer using gossiping. Each peer then uses this information and a distributed ranking algorithm to independently

locate and retrieve information relevant to particular queries. In this section, we first describe PlanetP's gossiping algorithm. Then, we evaluate the reliability of this gossiping algorithm to spread new information everywhere in a timely manner. We also briefly examine the bandwidth necessary to support gossiping. We have already shown that, given an accurate view of the global directory, our distributed ranking algorithm can successfully help users to pinpoint documents highly relevant to particular queries [2].

Gossiping. Information replicated at each peer includes a list of all peers, their IP addresses, their Bloom filters, and whether they are currently online. Events that change the directory and so require gossiping include the joining of a new member, the rejoin of a previously offline member, and a change in a Bloom filter. Currently, we do not gossip the leaving (temporary or permanent) of a peer although this may become necessary to improve our replication approach.

PlanetP's gossiping algorithm is a combination of *rumor mongering* and *anti-entropy* as previously introduced by Demers et al. [5] together with a *partial anti-entropy* algorithm that we found improved performance significantly for dynamic P2P environments. This 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 the above 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 (say, every tenth rumoring operation becomes an anti-entropy operation or if there's currently no new information to be rumored). An anti-entropy message from x to y asks 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.

Finally, we observe that in a dynamic P2P environment, where rumors may arrive often because of peers' leaving and coming, the time required to spread a particular rumor everywhere can become highly variable. 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 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. We refer the reader to [3] for the details of this dynamic adjustment. 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.

Performance. We now turn to evaluating the reliability with which gossiping can spread new information throughout a community. This is critical to our goal of locating relevant information in a reliable and accurate manner. For example, if a peer A does not learn about new content that is being shared by another peer B because changes in B's Bloom filter does not reach A, then a query at A cannot locate this new document, even if it is highly relevant to that query.

We evaluate PlanetP's gossiping performance using a simulator that has been validated against a pro-

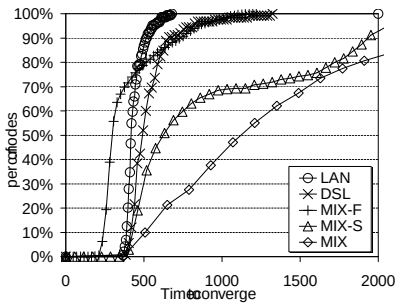

Figure 1: *Convergence time of gossiping in seconds for a dynamic community with 2000 members. LAN represents a community of peers connected by 45 Mbps links. DSL represents a community of peers connected by 512 Kbps links. MIX represents a community of peers connected by a mixture of link speeds. Using measurements of the Gnutella/Napster communities recently reported by Saroiu et al. [13], 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.*

totype implementation. We use measurements of our prototype to parameterize the simulator. We refer the reader to [3] for our simulation parameters together with the bulk of the experiments and their results.

Here, we present the results of only one experiment. In particular, we study how well gossiping performs in a dynamic community of 2000 (re)joining and leaving peers. 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. When a peer that starts out offline first rejoins the community, it has to retrieve 1000 new keys for each member currently online. These parameters were based roughly on measurements reported by Saroiu et al. [13] (except for the number of new keys being shared occasionally) and are meant to be representative of real communities.

Figure 1 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, convergence time is significantly worse. This degradation in performance is due to our requirement that an offline peer obtain 1000 new keys per currently online peer (which models the fact that the peer has been offline for a long time). This translates to the downloading of several MB of Bloom filters, which takes a long time at modem speed.

This degradation points to two necessary adaptations: (1) we must implement a way for new users to acquire Bloom filters slowly over time if we want to accommodate modem connections, and (2) we should alter our gossiping algorithm so that peers with fast connections are not hurt by peers with slow connections. We have designed and implemented the second change [4]. The MIX-F curves give convergence time for joins and rejoins of fast peers (fast being those with connection speeds of at least 512 Kb/s) 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. The above results along with results from a number additional simulation experiments [3] suggest that gossiping works reliably, at least to the range of several thousand peers. Not once did we observe an event that did not propagate successfully throughout the community. We are in the process of evaluating whether gossiping will scale to ten thousand peers.

4 Maintaining Availability

In this section, we describe an approach to random replication that probabilistically ensures having at least one copy of every document always online. Each peer advertises the amount of storage, called the replication store, that it is willing to devote to storing extra copies of documents. Typically, information about the current state of the replication store is piggybacked on rumors. Periodically, every T_r seconds, each peer randomly selects a file in either its data store or its replication store. The peer then checks for the availability of the file using its set of Bloom filters (we assume that file names will be inserted into the Bloom filters along with terms contained in the files). If the availability is sufficient, then it does nothing. If the availability is deemed insufficient, it actively replicates this file. Specifically, the replicating peer fragments the file using a variation

$$A(F) = 1 - P_{Offline}^{k} \sum_{i=n-m+1}^{n} \binom{n}{i} P_{Offline}^{i} (1 - P_{Offline})^{n-i} \quad (1)$$

of the Reed Solomon erasure coding [10], where n fragments are created but only m are required to reconstruct the file. Finally, it pushes the n fragments to peers chosen using hints about available space in the peers' replication store.

When a peer receives a replication request, if there is room in its replication store, then its saves a copy of the forwarded document fragments. If there is insufficient room, it either rejects the replication request or ejects enough fragments to make room for the new fragments. If the peer already has the fragments being pushed to it, then it just drops the replication request.

Over time, we believe this replication algorithm will push the most popular and important documents onto the subset of peers that are online most of the time. Peers that are likely to be online at different times will get copies as well in order to maintain continuity. Given a replication store that is very large compared to the set of documents being shared, then this approach will work [11]. The question becomes what is the necessary ratio of the replication store to the size of the document set and what is the necessary distribution of the replication store across peers with different join/leave behaviors. We intend to explore the answers to these questions over the next several months.

Evaluating files' availability. We assume that files are naturally replicated by peers that are interested in accessing that file when it is offline (or when the original owner is offline). We call these hoarded copies. PlanetP then actively replicates only files that are not highly hoarded. We are exploring two scenarios for evaluating a file's availability. The first assumes that there is a very slowly changing community of users. Each user advertises their average online and offline times as well as the time they started the current online session. The idea is to minimize the probability of having all the hoarded replicas and $n - m + 1$ fragments of a file being offline at the same time. Therefore availability is estimated using equation 1 where F is the file, $A(F)$ is the estimated availability of F, $P_{Offline}$ is the *average* probability of a peer being offline, k is the number of hoarded copies of F, n is the number of existing fragments of F, and m is the number of fragments required to reconstruct F. To keep the equation simple, we have simply used the average probability of peers being online and offline. For better accuracy in a real system, we may need to account for peers with widely differing behaviors, complicating the above equation significantly. Also, we will likely have to modify the above algorithm to account for the fact that offline peers may leave the community and never come back.

The second, more pessimistic, approach to estimate availability assumes only online nodes can keep a file available. In this scheme, PlanetP tries to minimize the chances of having all the online nodes that store a file going offline at the same time. Availability for a file F is estimated using equation 1 where $P_{Offline}(k)$ is replaced by $P_{GoingDown}(k)$. In addition, note that in this scenario, having accurate information about nodes currently online is critical. Thus, we may have to extend PlanetP to gossip leave events as well as (re)join events.

Choosing replication nodes. Peers keep space hints that track the amount of available storage at other peers. When fragments are distributed, a best-fit policy is used to select the target node. Space hints are obtained when storing fragments at other nodes or when a node denies replication request. If no hints are available, then targets are selected at random.

Fragmentation. We use a variation of the Reed Solomon (RS) erasure coding for fragmenting a file for replication. In a standard application of a code such as RS, one creates n fragments but only m distinct fragments ($m < n$) are needed to reconstruct the file [10]. The disadvantage of directly applying such a coding scheme is that to enhance the availability of a file, a peer would have to know exactly which of the n fragments are currently missing. Our use of the Reed Solomon code, where n is not a *hard* parameter, allows us to make the following useful extension. When a peer determines that it needs to increase the number of fragments of a file to increase its availability, it simply generates k additional *random* fragments.

The fundamental idea behind RS is that a polynomial of degree $m - 1$ in a Galois field $GF(2^w)$

is uniquely defined by any m points in the field. In order to create an erasure code for the blocks $D_1, ..., D_m$ of a file, we need a polynomial p such that $p(t_1) = D_1, ..., p(t_m) = D_m$. In addition, we want to create $n - m$ extra fragments $p(t_{m+1}) = C_1, ..., p(t_n) = C_{n-m}$ such that we can reconstruct the polynomial from any m tuples of the form $(t_1, D_1), ..., (t_m, D_m), (t_{m+1}, C_1), ..., (t_n, C_{n-m})$.

A typical RS implementation will just use a fixed base $(t_1, ..., t_n)$ for a given (m,n) code. Our extension consists of picking these points randomly from $[0, 2^w]$. Since we can pick a w, we can make the chance of having duplicates fragments as small as needed.

Open questions. While our approach is promising, a number of open questions remain. What replacement algorithm should peers use to maintain their replication store? Do we need to actively garbage collect the replication store? Is the system stable? Will we need some sort of backing-off component if peers start to thrash the replication store? How does the fragmented replication scheme fit with our ranking algorithm when no hoarded (complete) copy of a file is available online?

5 Related Work

Most recent P2P systems [15, 12, 14, 9] have proposed imposing different structures over P2P communities to provide scalable name-based object location. In this paper, we present an unstructured approach that uses randomized gossiping. While less scalable, our approach does not require any maintenance of global structures and is robust to nodes signing off abruptly. Further, we focus on content search and retrieval instead of object location. This focus fits well with our gossiping infrastructure because individual nodes do not have to shoulder the entire burden of publishing large numbers of keywords; rather, the entire community helps out via gossiping.

Other systems like OceanStore [8] have also used erasure codes to improve availability. In contrast to PlanetP, however, OceanStore can always rely on an OceanStore Service Provider to regenerate a missing fragment if their distributed repair algorithm needs it. To maintain PlanetP's unstructured approach, we extend the basic Reed Solomon erasure code to allow peers to increase the availability of files by generating some number of random fragments.

Mnemosyne [11] is a P2P steganographic storage system. While their main focus is on concealing the existence of files, they also address the problem of placing fragments randomly on a Tapestry network. Their simulations show that to achieve a 99.999% availability they need 10 times more disk space than the amount actually store. Since PlanetP is not concerned with hiding files' existence, we believe we will be able to improve this ratio.

6 Conclusions and On-Going Work

P2P computing is a potentially powerful model for information sharing between *ad hoc* communities of users. As P2P communities grow in size, however, reliably locating information distributed across the community becomes challenging, especially when peers join and leave the online community in an uncontrolled and unpredictable manner. In this paper, we have presented the approaches taken in PlanetP to provide a reliable content search and retrieval engine in the P2P environment.

In PlanetP, we break the problem down to two phases: (1) reliably locating peers that may contain information relevant to a query, and (2) ensuring that at least one copy of each document is always available on some online peer to support reliable retrieval of relevant information. Our approach then uses gossiping to maintain a copy of the global directory and summaries of peers' indexes on each peer, and to use this information to support a search algorithm that employs a text-based ranking scheme to find relevant information. We have worked out the gossiping algorithm in some detail and preliminary simulated results show that our algorithm can reliably diffuse information throughout the community so that no peer's local picture of the global community becomes too out of date.

Our proposed approach to address the second phase is to explore a randomized replication approach that, at heart, is not too different from gossiping. We have just formulated this approach and are in the process of designing the algorithm and exploring the design space. We believe that this approach can work but need to validate this belief. Thus, once we have a concrete design and implementation of both gossiping and random replication, we intend to extend our

current simulator to simulate searches in a live community, and measure the efficacy of our infrastructure to deliver accurate search and retrieval capabilities.

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] F. M. Cuenca-Acuna and T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc P2P Communities. In *Proceedings of the International Workshop on Peer-to-Peer computing (co-located with Networking 2002)*, May 2002.

[3] 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.

[4] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable P2P Information Sharing Systems. Technical Report DCS-TR-487, Department of Computer Science, Rutgers University, May 2002.

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

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

[7] J. Guterman. Gnutella to the rescue? not so fast, napster friends. http://gnutella.wego.com, Sept. 2000.

[8] 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.

[9] 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.

[10] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2):24–36, Apr. 1997.

[11] T. Roscoe and S. Hand. Transaction-based charging in mnemosyne: a peer-to-peer steganographic storage system. In *Proceedings of the International Workshop on Peer-to-Peer computing (co-located with Networking 2002)*, May 2002.

[12] 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.

[13] 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.

[14] 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.

[15] 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.