The Persistence of Memory in Freenet

Theodore W. Hong
Imperial College London
t.hong@doc.ic.ac.uk

Ian Clarke
The Freenet Project
ian@freenetproject.org

Abstract

Most peer-to-peer systems are either unstructured file sharing networks that offer no data availability guarantees or tightly-structured file storage networks that do. The Freenet network occupies a middle ground in providing a loosely-structured file storage service but not guaranteeing data persistence absolutely. We present simulation results showing that in practice not that many files ever do get dropped from the system, even with very small datastores. Surprisingly, medium popularity files are more vulnerable than the least popular ones.

1 Introduction

Despite their relative lack of sophistication, peer-to-peer file sharing networks like KaZaA have enjoyed a tremendous amount of success, with millions of users and file transfers every day. In these systems, files have no predefined "home" nodes, but are stored on the computers of those who initially publish them and those who subsequently pull them through requests. No attempt is made to guarantee data availability: participants can and regularly do take their nodes offline without notice, making their data unavailable to the network until the nodes rejoin. Still, users appear willing to put up with some level of unreliability and are relatively unconcerned if they can't always get what they want.

These unstructured file *sharing* networks can be contrasted with the recent generation of tightly-structured file *storage* networks like Chord[5], OceanStore[8], CAN[7], and Pastry[9], in which files are pushed to nodes for storage. Every file inserted is assigned a unique key that maps to some preferred node or set of nodes to be stored on, according to a global distributed hash table structure. Since specific nodes are assigned responsibility for particular files, they can track replicas and ensure their availability.

Freenet[3] occupies a middle ground between the two groups in that it is a file storage network in the sense just described, yet does not offer deterministic guarantees since it is loosely rather than tightly structured. Files do not have fixed homes in the network but migrate towards clusters through a local self-organizing process. This characteristic enhances security because malicious nodes cannot manipulate the network to gain responsibility for (and thus control over) files that they wish to attack. However, the price is that Freenet cannot always guarantee the persistence of data. Since nodes are not explicitly assigned responsibility for files, they may inadvertently delete the last copy of a file without realizing it.

How tolerable is Freenet's level of lossiness?

A common criticism is that the network might come to be dominated by the most popular files and never keep any unpopular ones. We conducted a set of simulation experiments that attempted to quantify the extent to which files are actually dropped in practice, and discovered that only a relatively low proportion of files are dropped even with very small datastores. As expected, the number of copies of files follows the popularity distribution, but surprisingly, medium popularity files turn out to be more vulnerable to being dropped than the least popular ones.

The rest of the paper is organized as follows. Section 2 gives a quick overview of Freenet's data storage mechanisms, while Section 3 provides some explanation of why the system is lossy as a consequence of its goals. In Section 4, we present details of our simulation measurements on data persistence. Section 5 reviews related work and Section 6 concludes and offers some directions for future work.

2 Data storage in Freenet

Here we briefly summarize how Freenet handles data storage; for full details, see [3].

Files are published to the network by sending an insert message to an initial node with some specified time-to-live limit, typically around 15–25. This message is forwarded from node to node until the time-to-live expires, at which time the file is stored on all of the nodes reached by the insert message. Hence, the time-to-live determines the number of initial copies of the file. Requests are similarly carried out by forwarding request messages through the network until either the data is found or the time-to-live expires. If the data is successfully found, it is passed back through the network and stored on all of the intermediate nodes on the way back to the original requester, thus creating additional copies to speed up subsequent requests and distribute load. The number of copies created is not easy to assess beforehand but depends on the distance the request had to travel before encountering the data—somewhere between 1 and the time-to-live.

Nodes have a user-specified datastore capacity, which has a default value of 256 megabytes but is often increased to the gigabyte range. Storage is managed as a least-recently used cache in which data items are kept sorted by decreasing time of most recent access. When a new file arrives which would cause the datastore to exceed the designated size, the least-recently used files are evicted in order until there is room.

3 Why Freenet is lossy

Under this cache strategy, it is possible that files that are rarely requested might be evicted from all the nodes on which they are stored and thus disappear from the network. Though undesirable, this lossy data model is a consequence of Freenet's open access policy.

One of Freenet's primary rationales is to provide a safe haven for sensitive information that might face attack from powerful opponents. For example, Diebold Election Systems has reacted strongly to the distribution of internal memos discussing vulnerabilities in its vote-counting software (accessible at freenet:ssk@ijROtU5Ksue7Q5N4UZefFioV76EPAgM/1). A nervous insider exposing such information would want to be able to simply "post and run." We don't want to force her to jump through hoops like running a node providing storage, making a payment, or establishing a reputation before permitting her to publish.

As a result, however, spam flooding attacks become a concern: if Deep Throat can insert, so can Joe Spammer. A computer can't tell the difference. The ability to drop files is necessary in order to prevent the network from being filled up with junk. One possible alternative to LRU might be to create some type of rating system for files, but distributed ratings are hard to implement well and might themselves be vulnerable to meta-spamming.

To what extent are files actually dropped, however? The next section presents some simulation experiments that attempt to quantify Freenet's lossiness in practice.

4 Persistence measurements

In earlier work[3, 6], we implemented a simulator to evaluate the request performance of the Freenet network. Using a similar methodology, we now consider its performance in terms of data persistence. The number of copies of a file and its likelihood of being dropped may be dependent on many factors, including request popularity, size, and age. In any network of fixed capacity, the continued insertion of new data will of course eventually cause some data to be dropped. Therefore, in order to compare Freenet against a baseline "no-dropping" scenario, we simulated a network that grew while we inserted data, with the rate of increase in capacity equal to the average rate of insertion of new data.

Specifically, the test network was initialized to a set of 20 nodes, each having a datastore size of 50 files and a routing table size of 250 addresses. The datastores were initialized to be empty, and the routing tables were initialized to connect the network in a regular ring-lattice topology in which each node had routing entries for its four nearest neighbors.

The small datastore size was chosen to highlight the possible effects of dropping data. Real datastores can easily hold thousands of files, so this simulation should be regarded as an upper bound on the amount of data dropped.

All files were assumed to be the same size—although real files vary in size, Freenet splits large files into standard blocks which are treated independently at the network level. The only difference is that the simulation ignores the fact that blocks belonging to the same file will tend to be inserted or requested at the same time.

Inserts of random keys were sent to random nodes in the network, interspersed randomly with requests for random keys known to have been previously inserted, using a time-to-live of 20 for both. To model the varying popularity of different files, keys were drawn from a Zipf (power-law) distribution. Empirical evidence suggests that the popularity of data in the real world follows a highly-skewed distribution of this type[1].

Every five timesteps, a new node was added to the network by simulating a node announcement message with time-to-live of 10 sent from it to a random existing node. On average, there will be 2.5 inserts * time-to-live 20 = 50 inserted copies of files between additions, and each added node has capacity 50. Therefore, there should be a rough balance between inserts and capacity growth. The 2.5 requests between node additions will also generate additional copies of files, tilting the balance somewhat towards overfilling the network. Simulations were run until the network reached a size of 10,000 nodes, and results from ten runs were averaged together.

As a baseline comparison, a network that

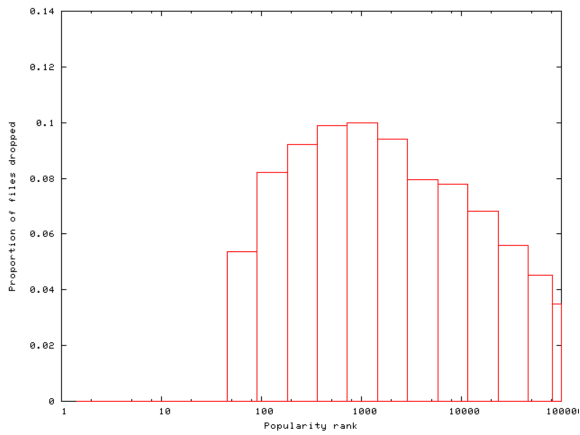

Figure 1: Proportion of files dropped versus popularity.

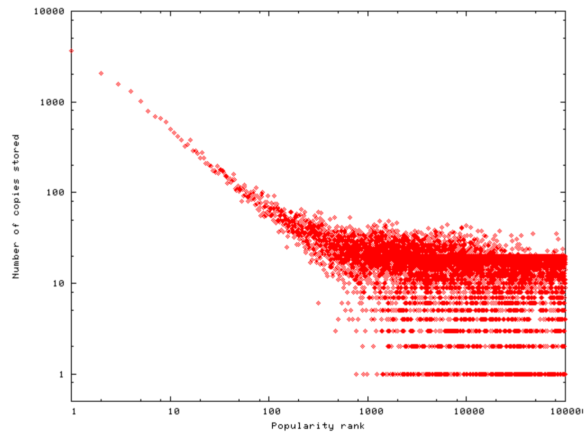

Figure 2: Number of file copies stored versus popularity.

never drops data might simply store 20 copies of each inserted file in perpetuity. Freenet, however, will reallocate space to adapt the number of copies to the number of requests, increasing the number of copies of popular files and decreasing the number of copies of unpopular files. Although this reallocation keeps nodes from being overwhelmed by a flood of requests for a suddenly-popular file, the danger is that unpopular files might have their copies reduced to zero and disappear from the network altogether.

Figure 1 shows the proportion of files that are dropped from the network versus their popularity rank. Key 1 is the most popular and key 100,000 the least. Since popularity in the Zipf distribution drops off sharply, the data have been exponentially binned.

Even with these extremely small datastores, over 90% of files stay in the system. As expected, the most popular files are never dropped. Surprisingly, however, the least popular files are not the ones that are most vulnerable. The files that are dropped most often are the ones in the middle of the popularity distribution. We are not entirely sure why this should be the case, but are investigating further.

As might be expected, the number of copies of a file at the end of the simulation is strongly correlated to its popularity and follows the same Zipf distribution, as shown in Figure 2. At the head of the distribution, the most popular file is stored on about a third of all nodes. The tail of the distribution breaks down somewhat, with a strong cluster around 20, which is the number of copies of a newly-inserted file in our simulation.

Finally, we consider whether a file's age has any relationship to its likelihood of being dropped. Figure 3 shows the distribution of ages (in terms of timesteps since insertion) of all the files dropped during the course of the simulation. When plotted on a logarithmic scale, the ages are approximately exponentially distributed. This means that a file's probability of being dropped is roughly independent of age, since in an exponential distribution, the probability of surviving from age a to age $a+1$ is constant for all a.

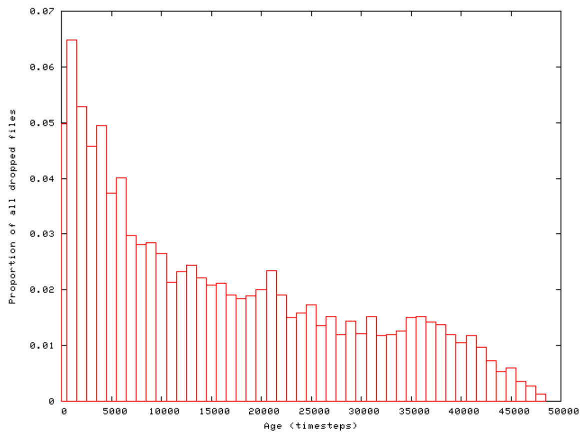

Figure 3: Age distribution of dropped files.

5 Related work

There has been much work done examining request pathlengths and search strategies in peer-to-peer networks, for example [2], but less on data persistence and caching strategies. Zhang[10] describes a modification to Freenet's caching to improve its routing by reinforcing a small-world topology. Cohen and Shenker[4] analyzed a range of replication strategies and discovered the interesting result that a square-root allocation is optimal under certain conditions.

6 Conclusion

In conclusion, we performed simulations to examine the extent to which files are actually dropped in Freenet. Even with very small simulated datastores, more than 90% of files did stay in the network. As might be expected, the number of copies stored was strongly correlated to popularity, although surprisingly the medium-popularity files were the most vulnerable rather than the least popular ones. Age did not affect the likelihood of dropping a file.

One direction for future work is to examine the effect of different caching strategies on data persistence and query performance. We would also like to perform persistence measurements on the live network, although it is difficult to obtain results because of the effects of bugs and nodes running many different versions of the software.

References

[1] L.A. Adamic and B.A. Huberman, "Zipf's law and the Internet," *Glottometrics* **3**, 143 (2002).

[2] L.A. Adamic, R.M. Lukose, A.R. Puniyani, B.A. Huberman, "Search in power-law networks," *Physical Review E* **64**(4), 046135 (2001).

[3] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: a distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies* (2001).

[4] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," ACM SIGCOMM 2002.

[5] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," ACM SOSP 2001.

[6] T. Hong, "Performance," in *Peer-to-Peer,* O'Reilly (2001).

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," ACM SIGCOMM 2001.

[8] S. Rhea *et al.*, "Maintenance-free global data storage," *IEEE Internet Computing* **5**(5), 40–49 (2001).

[9] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," SOSP 2001.

[10] H. Zhang, A. Goel, and R. Govindan, "Using the small-world model to improve Freenet performance," IEEE Infocom 2002.