Friendstore: cooperative online backup using trusted nodes

Dinh Nguyen Tran, Frank Chiang, Jinyang Li
New York University

1 Introduction

Today, it is common for users to own more than tens of gigabytes of digital pictures, videos, experimental traces, etc. Although many users already back up such data on a cheap second disk, it is desirable to also seek off-site redundancies so that important data can survive threats such as natural disasters and operator mistakes. Commercial online backup service is expensive [1, 11]. An alternative solution is to use a peer-to-peer storage system. However, existing cooperative backup systems are plagued by two long-standing problems [3, 4, 9, 19, 27]: enforcing minimal availability from participating nodes, and ensuring that nodes storing others' backup data will not deny restore service in times of need.

This paper presents Friendstore, a cooperative backup system that differs from previous proposals in one key aspect: each node only stores its backup data on a subset of peer nodes chosen by its user. In practice, each user trusts nodes belonging to her friends or colleagues. By storing data on trusted nodes only, Friendstore offers a non-technical solution to both the availability and denial-of-service problems: users enter "storage contracts" with their friends via real world negotiations. Such contracts are reliable because social relationships are at stake. Each user only stores data with her friends instead of friends-of-friends because we do not believe non-direct social relationships can enforce such contracts reliably.

Although Friendstore's architecture is conceptually simple, a number of technical challenges remain in order to provide reliable long term storage with the highest possible capacity. The capacity of Friendstore is limited by two types of resources: wide area bandwidth and the available disk space contributed by participating nodes. Bandwidth is a limiting resource because nodes must re-copy backup data lost due to failed disks. To prevent a node from storing more data than it can reliably maintain, we propose to let each node calculate its maintainable capacity based on its upload bandwidth and limit the amount of backup data it stores in the system accordingly.

The system's capacity may also be limited by the available disk space. We propose to trade off bandwidth for disk space by storing coded data in situations when disk space, instead of bandwidth, is the more limiting resource. Our scheme, XOR(1,2), doubles the amount of backup information stored at a node at the cost of transferring twice the amount of data during restore in order to decode the original data.

The technical challenges addressed in this paper, namely, calculating maintainable capacity and trading off bandwidth for storage, are not unique to Friendstore but present in all replicated storage systems. However, the targeted deployment environment of Friendstore makes addressing these challenges a pressing need. Friendstore runs on nodes with a wide range of bandwidth and available disk space. Some nodes may be limited by their upload bandwidth, hence they must refrain from storing more data than the maintainable capacity. Other nodes may be limited by the available disk space. As each Friendstore node only has a few choices to store data, it is attractive to store more information in the limited disk space using coding.

The paper is organized as follows: Section 2 discusses the underlying trust model that has inspired Friendstore's architecture. We proceed to present Friendstore's overall design (Section 3), how a node calculates maintainable capacity (Section 4) and how it trades off bandwidth for storage (Section 5). In Section 6, we evaluate the long term reliability of Friendstore using trace-driven simulations and share lessons from our early software deployment.

2 Trust Model

The viability of all cooperative backup systems depends on the majority of participants cooperating, hence the name. Unfortunately, no technical solution can ensure that nodes always cooperate. For example, a node storing others' data can faithfully adhere to a system's protocol for a long time but decide to maliciously deny service when it is asked to help others restore. Therefore, the best a system can do is to ensure that our assumptions about how "well behaved" nodes act are highly likely to hold in practice. Systems do so by pruning the set of trustworthy nodes to eliminate misfits and creating disincentives to violating assumptions in the first place. For example, a node can frequently check that others are faithfully storing its data and remove any node that fails periodic checks [3, 9, 19] from the system. A disincentive to misbehavior could be punishments in the form of deletion of

the offending node's backup data [9] or expulsion from the system by a central authority [3]. Both of these approaches have drawbacks: pruning mechanisms based on system-level health probes can be imprecise (e.g. it is difficult to distinguish a node who has just suffered from a hard disk crash from one that purposefully deleted others' data). Inflexible disincentives (e.g. deletion of an expelled node's data) could cause the system to be unnecessarily fragile: it is easy to imagine an incident such as unexpected software crashes or temporary network congestion leading to an escalating spiral of punishments.

Friendstore leverages information in the social relationships among users to select trustworthy nodes and provide strong disincentives against non-cooperative behavior. Each user chooses a set of trusted nodes that she believes will participate in the system over the long term and have some minimal availability according to her social relationships. A Friendstore node only stores backup data on those trusted nodes. By exploiting real-world relationships in this way, Friendstore is able to use simple and lightweight technical mechanisms to provide a reasonable expectation that nodes will behave as expected. Friendstore checks for the presence of remote backup data infrequently and uses a long timeout to mask transient node failures knowing that the unresponsiveness of a trusted neighbor is more likely due to its user's vacation than an act of malice. Also, disincentives in this system carry more weight since they stem from possible disruption of the social relationships: violation of trust results in the resentments of one's friend which we believe most users want to avoid. Friendstore defers punishments for a misbehaving node such as deleting of its backup data to individual users who are free to use their own retribution policies based on more complete and accurate information.

3 Design Overview

Friendstore consists of a collection of nodes administered by different users. Each node runs an identical copy of the software and communicates with a subset of other nodes over the wide area network. The software running on a node has two roles: backing up a node's local data and helping others store their backups. We refer to a node as an *owner* when it is performing activities involving its own data and a *helper* when it is acting to help others. Each node is named and authenticated by its public key and a user chooses a subset of helpers for storing data by configuring her node with the public keys of her friends' nodes.

An online backup system undertakes a number of activities: to store local data on remote helpers (backup), to periodically check that remote copies of its backup data are still intact and create new ones if not (verify and repair), and to retrieve remote backups following a disk crash (restore). We describe how Friendstore performs each task in turn.

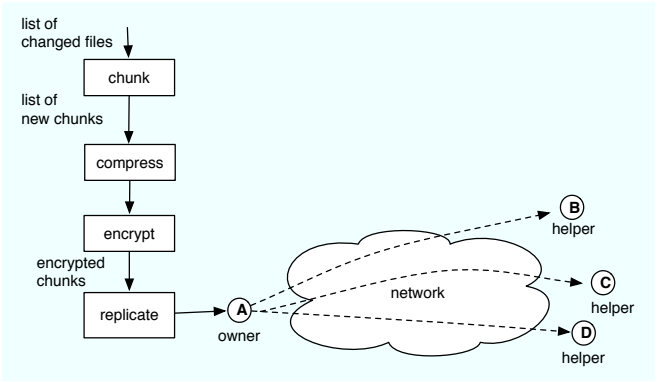

Figure 1: The sequence of steps an owner takes for preparing a collection of files for backup on remote helpers.

Backup An owner prepares a collection of files for backup in a sequence of steps shown in Figure 1. The owner processes the files by chunking large files into smaller pieces, compressing and encrypting individual pieces using symmetric encryption. Finally, it uploads r copies of its encrypted chunks on distinct helpers. We use the term *replica* to refer to these encrypted chunks stored at helpers. In our prototype, r is set to be two. We prefer replication over erasure coding because efficient erasure codes require more helpers than typically available for many owners. Owners do not modify replicas at helpers once created, but can explicitly delete them. To garbage collect data, helpers also expire replicas after a default three month period.

Friendstore discourages one common form of free-riding, namely, selfish helpers attempting to store less data for others than required. To detect such selfish behavior, each node keeps track of how much backup data its owner has stored on each of the helpers vs. how much data that helper's corresponding node has stored on itself. An owner always prefers storing data at the helper who owes it the most storage and if denied storage by such a helper, the owner reports such incidence to its user for punishments or further investigations.

Verify and Repair Each owner periodically checks the health of its remote replicas by requesting its helpers to compute and return the hash of a randomly chosen replica starting at a random offset. By comparing a helper's hash value with that computed from its local data, an owner can detect replica corruption and re-send the corrupted replica quickly. When an owner fails to contact a helper for verification after a timeout period, it re-sends all replicas stored on the unresponsive helper to another helper. Since users explicitly choose helpers that agree to participate over the long term, we believe most failures are due to transient node offline events as opposed to permanent departures. Thus, Friendstore uses a large timeout threshold of 200 hours to mask most transient failures.

Restore Restoring data after an owner's disk crash is straightforward. However, since the owner might lose all per-

sistent data after a disk crash, it must store its private key used to encrypt the replicas offline. Friendstore uses a separate service to help an owner remember the identities of its helpers. During restore, a helper uploads the owner's data as well as its own lost replicas previously stored on that node. We must point out that a helper has no real incentives to help an owner restore. The reason we believe it is likely to do so in practice comes from the real world social relationship between users.

4 Calculate the maintainable capacity

If backup data is to be stored reliably, it must be re-copied by owners as disks fail. The rate at which an owner can upload data determines the amount of data it can reliably store on remote helpers. This amount could be much less than the disk space available at helpers. To ensure the reliability of backup, we do not not want to store more data than can be maintained over the long term. Therefore, we propose to let each owner calculate its maintainable storage capacity (s_{max}) based on its upload bandwidth and use this estimate to limit how much data it attempts to store on helpers. Similarly, we calculate the maintainable capacity for each helper (d_{max}) and use the estimate to limit the amount of data it contributes to other owners. For simplicity, we assume that a node's download bandwidth is larger than its upload bandwidth. Similar arguments apply when a node's download bandwidth is the more limiting resource.

Intuitively, the reliability of replicated data is affected by the amount of bandwidth required to recover from permanent disk failures relative to the amount of available bandwidth at each node [8]. When an owner stores s units of backup data on remote helpers, it must consume $\lambda_f \cdot 2 \cdot s$ units of bandwidth to re-copy 2 replicas per unit of data when disks fail at rate λ_f. Likewise, when a helper stores d units of replicas for others, it needs to upload one out of every two copies to help owners restore, consuming $\lambda_f \cdot \frac{d}{2}$ units of bandwidth. Since a node acts both as an owner and a helper, the total bandwidth required to recover from permanent failures is: $\lambda_f \cdot 2s + \lambda_f \cdot \frac{d}{2}$. Our simulation results show that when the required recovery bandwidth does not exceed one tenth of the actual available bandwidth, there is little data loss ($< 0.15\%$) over a five year period. Therefore, we can calculate the maintainable storage capacity (s_{max}, d_{max}) as follows:

$$\lambda_f \cdot 2 \cdot s_{max} + \lambda_f \cdot \frac{d_{max}}{2} = \frac{1}{10} \cdot B \cdot A \qquad (1)$$

In Equation(1), a node's available upload bandwidth is measured by its upload link speed (B) scaled by the fraction of time it remains online (A).

As each unit of backup data is replicated twice, helpers must store twice the amount of total backup data. Substituting $d_{max} = 2s_{max}$ into (1), we obtain $s_{max} = \frac{B \cdot A}{30 \cdot \lambda_f}$. To calculate the actual value of s_{max}, an owner uses its measured upload bandwidth, node availability and an approximation of the permanent disk failure rate (e.g. $\lambda_f \approx 1/3$ years if we approximate the average disk lifetime to be 3 years). If a user only allocates a fraction of her uplink capacity for use by Friendstore, s_{max} will be calculated using the throttled bandwidth. Each owner refrains from storing more than s_{max} units of backup data on remote helpers and each helper does not store more than $d_{max} = 2s_{max}$ units of data for other owners.

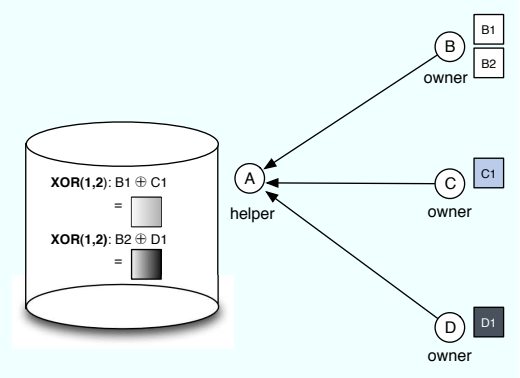

Figure 2: Helper A uses XOR(1,2) to store coded replicas belonging to different owners.

5 Store more information with coding

Disk space, instead of upload bandwidth, can also be the limiting resource in many circumstances. For example, when operating on a college campus network, a helper can reliably store up to $d_{max} = 754$GB data for other owners with 1Mbps upload bandwidth. But in reality, its idle disk space can be far less than d_{max}. Since each Friendstore owner only has a few trusted helpers to store data, it is important to utilize helpers' available disk space efficiently. We introduce a coding scheme, called XOR(1,2), to let a helper simultaneously provide redundancy for multiple owners by storing coded replicas. The actual coding mechanism is not new and has been explored in RAID systems [23]. However, Friendstore presents a novel application for coding to enable a helper to trade off bandwidth for storage when the available disk space is the limiting resource.

Figure 2 illustrates an example in which helper A must store replicas for owners B, C, D. Instead of storing backup data from them separately ($B1, B2, C1, D1$), helper A can store $B1 \oplus C1$ and $B2 \oplus D1$, consuming two units of space as opposed to 4. To restore $B1$, helper A needs to fetch the original replica ($C1$) from owner C in order to decode $B1$, i.e. $B1 = C1 \oplus (B1 \oplus C1)$. As this example shows, XOR(1,2) allows helper A to utilize the original information stored at owner C to recover data belonging to a different owner (B) but it also consumes additional bandwidth during restore.

Since coding trades off bandwidth for storage, we must be careful not to apply XOR(1,2) in situations when the

capacity is limited by bandwidth. As Figure 2 shows, with coding, an owner C must upload its replica again in response to owner B's failure as well as helper A's failure. Therefore, we must update Equation (1) to reflect the additional replica transfer by an owner when XOR(1,2) is in use: $\lambda_f \cdot (2+1) \cdot s'_{max} + \frac{d'_{max}}{2} = \frac{1}{10} \cdot B \cdot A$, resulting in $s'_{max} = \frac{B \cdot A}{40 \cdot \lambda_f}$. Each owner uses the new estimate (s'_{max}) to constrain the amount of data it attempts to store on remote helpers while allowing XOR(1,2). Similarly, a helper uses d'_{max} to limit the amount of information it stores as coded replicas for other owners. Furthermore, a helper does not code those replicas for which their owners have not explicitly permitted coding. By allowing coding, an owner agrees to undertake extra work to upload the original encrypted replicas to the helper again during data restore by another owner. Therefore, an owner should permit coding only for replicas that correspond to immutable data such as media files. We rely on the normal verification process to detect replica loss due to unexpected changes in original files. An owner has incentive to allow coding because doing so enables it to store more data at a helper than otherwise possible. Unfortunately, coding also causes an owner's restore process to depend on another owner that might not be directly trusted by its user. This is a tradeoff that Friendstore leaves to individual users.

Storing coded blocks complicates the normal verification process because a helper is unable to calculate the requested hash value of the original replica. To enable verification, we use a homomorphic collision resistant hash function with the property: $h_G(x+y) = h_G(x)h_G(y)$ where $G(p,q,g)$ specifies the hash function parameters [17]. To apply this homomorphic hash function to verify coded replicas, we change the XOR operator in XOR(1,2) to be the addition operation over Z_q (q is a large prime). We illustrate the new verification protocol using Figure 2 as an example. When helper A is asked by owner B to produce the hash for replica $B1$, it first requests the hash for $h_G(\bar{C}1)$ from owner C, where $\bar{C}1$ is the complement of $C1$ in Z_q and returns to owner B the requested hash value by computing $h_G(B1) = h_G(B1 + C1)h_G(\bar{C}1)$.

6 Evaluation

This section examines Friendstore's performance in terms of storage utilization and long term reliability using trace-driven simulations. In addition, we also present statistics from a pilot prototype deployment on 21 nodes over a period of two months and share our lessons learnt from running Friendstore in practice.

Storage utilization One concern with Friendstore is that its storage utilization might be low: a node might find out that all of its helpers are full even though available disk space exists elsewhere in the system. We show that Friendstore achieves good utilization when operating under typical social relationships. We use a crawled 2363-node Orkut network as the simulated social graph so each owner stores data on helpers belonging to its Orkut neighbors. Since each helper contributes the same amount of disk space as its corresponding owner tries to consume, a homogeneous storage system (e.g. DHash [10], Pastry [28],OpenDHT [26]) would be able to achieve perfect utilization. In comparison, Friendstore's utilization is less (87%). We find that most "wasted" storage space resides on nodes with very low degrees. As our crawled topology is only a subgraph of the Orkut network, 23% nodes have less than 3 crawled neighbors while less than 5% Orkut nodes have degrees smaller than five in the full graph. We vary the minimum node degree by adding new links to the subgraph while preserving the original subgraph's clustering coefficient of 0.23 using the method proposed in [32]. Figure 3 shows that space utilization increases quickly to reach more than 95% when each owner has more than 5 helpers, suggesting Friendstore's utilization is likely to be high in practice.

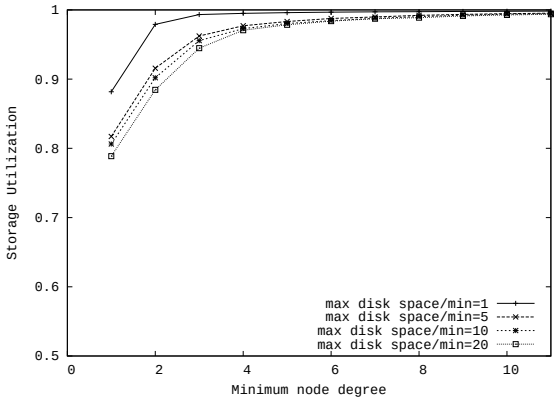

Figure 3: Storage utilization as a function of minimum node degree in the Orkut graph. Utilization remains high even when there is significant variance among contributed disk space by different helpers (as shown by the ratio between the maximum and minimum contributed space.

Long term data durability We use the FARSITE trace [7] which monitors the availability of corporate desktop machines to simulate transient node offline events. The median FARSITE node availability is 81%. Since the trace only covers 840 hours, we sample one random node's up-down sequence from the trace every 840 hours over five simulated years. We generate disk failures using a Weibull distribution to approximate an average disk lifetime of 3 years [30]. Whenever a node suffers a disk failure, we delete all its data. The failed node rejoins the system six days later and its owner attempts to restore from helpers immediately.

Figure 4 shows the fraction of data lost at the end of 5 years as a function of the amount of data each owner stores in Friendstore in the beginning of the experiments. If nodes do not backup at all, 81.2% data will be lost after 5 years. The loss rate increases as each owner stores more backup

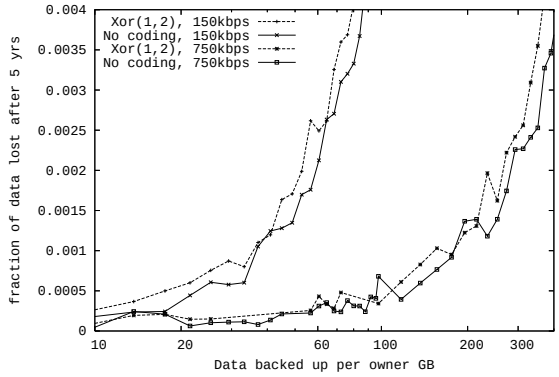

Figure 4: The fraction of data lost after they are first backed up in the system five years ago as a function of the amount backup data stored in the system by each owner. Experiments are done with different upload bandwidth, with and without coding.

Number of users	17
Number of nodes	21
Maximum nodes per user	3
Fraction of time online	75.3% (28.6%, 98.6%)
Max consecutive hours online	175 hours (53, 692)
Max consecutive hours offline	53 hours (13, 120)
Upload link bandwidth	624 kbps (211, 3744)
Number of neighbors per node	3 (1, 7)
Total amount of data backed up	578MB (275, 3077)

Table 1: Two months deployment statistics of Friendstore from 08/01/2007 to 10/01/2007. All statistics are shown by the median number followed by 20- and 80-percentile in parenthesis.

data at helpers because it cannot promptly re-copy replicas lost due to disk failures with limited upload bandwidth. According to Equation (1), $s_{max} = \frac{B \cdot A}{30 \cdot \lambda_f} = 48$GB for owners with 150Kbps upload bandwidth and 81% availability. $s'_{max} = 36$GB if coding is used. As we can see from Figure 4, when each owner stores no more than s_{max}, the probability of it losing data after five years is very low $< 0.15\%$. When an owner's upload bandwidth increases to 750Kbps, s_{max} increases to 240 GB. We have also simulated cases when an owner does not attempt to limit the amount of data it backs up according to s_{max}. Instead, it simply stores more backup data whenever its upload link becomes idle. With such a greedy strategy, we found that 7.98% of data initially backed up five years ago is lost at the end of the experiments.

Deployment Lessons Friendstore is fully implemented in Java and runs on a variety of OSes. We deployed the first version of the software from August to October 2007 in a small scale deployment involving 17 users and 21 nodes. The 21 nodes are of a mixture of university desktops, home desktops and laptop nodes running Windows, Mac and Linux. Table 1 summarizes various statistics from the deployment. Users have configured a wide range of upload bandwidth throttle for Friendstore. We are encouraged to find that the median upload bandwidth usable by Friendstore is quite high (624Kbps) and that nodes are fairly available (median availability is 75%). This suggests that Friendstore's maintainable storage capacity is likely to be high in practice.

The pilot deployment has revealed a number of practical issues for which our early design and prototype lacked good solutions:

- The deployed software displays a warning sign for users whenever a helper cannot be reached during the past five days. We intended for a user to contact her friend to fix the problem when noticing these warnings. Instead, our users often just ignored warnings altogether. The software could be more useful if it can automatically identify the source of the problem and email the responsible user to suggest a fix.

- Our deployed software used existing social relationships collected by Google Talk and Facebook to help users configure trusted nodes. We are surprised to find out that many users do not have accounts with either of the popular services. This suggests that we will have to provide our own social relationship registration service for future deployments.

- Some user owns a few machines and would like to express separate backup policies for each of them. For example, she might want Friendstore to backup her laptop's data on the desktop and not the other way around. Furthermore, a number of users administer a large pool of machines. Since the deployed software lacks the notion of a "group", it is difficult for these users to configure and administer Friendstore on a large collection of machines easily.

- Many users prefer storing certain subsets of files without encryption at trusted nodes so their friends can browse and view the stored files. This suggests that there is potential synergy between backup and file sharing since both might be able to use Friendstore as a generic replicated storage infrastructure.

The Friendstore software is currently undergoing its second major revision to address pitfalls observed in the deployment and to support users behind NATs.

7 Related work

Many researchers have exploited the use of social relationships for a variety of applications: for example, digital preservation (LOCKSS [20]), file sharing (Maze [33] and Turtle [24]), email (Re [13]), web search (Peerspective [22]). Many online reputation systems also use social networks to improve their accuracy [15, 21, 29]. Friendstore offers a novel use of social relationships, namely, to help users choose a set

of trusted nodes for reliable storage. Such user-specified trust relationships resemble those in SPKI/SDSI [12] and PGP certification chain. However, Friendstore's notion of trust is different from that in certification systems. CrashPlan [2] is a recently released commercial software that allow users to backup data on friends' machines. Friendstore shares a similar structure but addresses two technical challenges: ensuring a node does not store more data than can be reliably maintained and trading off bandwidth for storage when disk space is the more limiting resource. These challenges are not addressed in our earlier design [18].

There is a vast body of previous work in building reliable replicated storage systems [5, 8, 14, 16]. Many researchers have also recognized that bandwidth can often be the limiting resource when running over wide area nodes [6]. Our calculation for a node's maintainable capacity in Section 4 is directly inspired by [8] and similar in spirit to [25, 31]. Many storage systems use coding non-discriminately to store more information in the same amount of disk space. In contrast, Friendstore uses coding to trade off bandwidth for storage and hence it only applies coding when disk space is the more limiting resource.

8 Conclusion

This paper presents Friendstore, a cooperative backup system that gives users the choice to store backup data only on nodes they trust. Using trust based on social relationships allows Friendstore to provide a high assurance for reliable backup. Friendstore limits how much data a node stores according to its maintainable capacity and uses coding to store more information when disk space is the more limiting resource. Our initial deployment suggests that Friendstore is a viable solution for online backups. Friendstore is available publicly at `http://www.news.cs.nyu.edu/friendstore`.

Acknowledgments

We thank Frank Dabek who helped us greatly improve this paper. We are grateful to Robert Morris, Frans Kaashoek, Jinyuan Li and Friendstore's early users for their encouragement and insightful comments. This project was partially supported by the NSF award CNS-0747052.

References

[1] Amazon simple storage service. `http://www.amazon.com/gp/browse.html?node=16427261`.

[2] Crashplan: Automatic offsite backup. `http://www.crashplan.com//`.

[3] AIYER, A., ALVISI, L., CLEMENT, A., DAHLIN, M., MARTIN, J., AND PORTH, C. Bar tolerance for cooperative services. In *Symposium on Operating Systems Principles (SOSP)* (Oct. 2005).

[4] BATTEN, C., BARR, K., SARAF, A., AND TREPETIN, S. pstore: a secure peer-to-peer backup system. Tech. Rep. MIT-LCS-TM-632, Massachusetts Institute of Technology, Oct. 2002.

[5] BHAGWAN, R., TATI, K., CHENG, Y., SAVAGE, S., AND VOELKER, G. M. Totalrecall: System support for automated availability management. In *Proceedings of the ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2004).

[6] BLAKE, C., AND RODRIGUES, R. High availability, scalable storage, dynamic peer networks: Pick two. In *9th Workshop on Hot Topics in Operating Systems* (May 2003).

[7] BOLOSKY, W., DOUCEUR, J., ELY, D., AND THEIMER, M. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proceedings of the international conference on measurement and modeling of computer systems (SIGMETRICS)* (2000).

[8] CHUN, B.-G., DABEK, F., HAEBERLEN, A., SIT, E., WEATHERSPOON, H., KAASHOEK, M. F., AND MORRIS, R. Efficient replica maintenance for distributed storage systems. In *Proceedings of the 3rd Symposium on Networked System Design and Implementation (NSDI'06)* (May 2006).

[9] COX, L. P., AND NOBLE, B. Samsara: Honor among theives in peer-to-peer storage. In *19th ACM Symposium on Operating Systems Principles (SOSP)* (2003).

[10] DABEK, F., KAASHOEK, M. F., LI, J., MORRIS, R., ROBERTSON, J., AND SIT, E. Designing a DHT for low latency and high throughput. In *1st NSDI* (March 2004).

[11] Apple .mac service. `http://www.apple.com/dotmac/`.

[12] ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. Spki certificate theory. RFC 2693, Network Working Group, 1986.

[13] GARRISS, S., KAMINSKY, M., FREEDMAN, M. J., KARP, B., MAZIERES, D., AND YU, H. Re: reliable email. In *Proceedings of the 3rd Symposium on Networked System Design and Implementation (NSDI'06)* (2006).

[14] HAEBERLEN, A., MISLOVE, A., AND DRUSCHEL, P. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)* (2005).

[15] HOGG, T., AND ADAMIC, L. Enhancing reputation mechanisms via online social networks. In *Proceedings of the 5th ACM conference on Electronic Commerce* (2004).

[16] KOTLA, R., ALVISI, L., AND DAHLIN, M. Safestore: A durable and practical storage system. In *USENIX Annual Technical Conference* (2007).

[17] KROHN, M., FREEDMAN, M., AND MAZIRES, D. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2004).

[18] LI, J., AND DABEK, F. F2f: reliable storage in open networks. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)* (2006).

[19] LILLIBRIDGE, M., ELNIKETY, S., BIRREL, A., AND BURROWS, M. A cooperative internet backup scheme. In *USENIX Annual Technical Conference* (2003).

[20] MANIATIS, P., ROUSSOPOULOS, M., GIULI, T., ROSENTHAL, D. S. H., AND BAKER, M. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computer Systems 23* (Feb. 2005).

[21] MARTI, S., GANESAN, P., AND GARCIA-MOLINA, H. DHT routing using social links. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)* (2004).

[22] MISLOVE, A., GUMMADI, K. P., AND DRUSCHEL, P. Exploiting social networks for internet search. In *5th Workshop on Hot Topics in Networks (HotNets'06)* (2006).

[23] PATTERSON, D., GIBSON, G., AND KATZ, R. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (June 1988).

[24] POPESCU, B. C., CRISPO, B., AND TANENBAUM, A. S. Safe and private data sharing with turtle: Friends team-up and beat the system. In *Proc. 12th Cambridge International Workshop on Security Protocols* (2004).

[25] RAMABHADRAN, S., AND PASQUALE, J. Analysis of long-running replicated systems. In *Proceedings of the 25th IEEE Conference on Computer Communications (INFOCOM)* (Apr. 2006).

[26] RHEA, S., GODFREY, B., KARP, B., KUBIATOWICZ, J., RATNASAMY, S., SHENKER, S., STOICA, I., AND YU, H. OpenDHT: A public DHT service and its uses. In *Proceedings of ACM SIGCOMM* (Aug. 2005).

[27] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)* (Nov. 2001).

[28] ROWSTRON, A., AND DRUSCHEL, P. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles (SOSP)* (2001).

[29] SABATER, J., AND SIERRA, C. Social regret, a reputation model based on social relations. In *ACM SIGecom Exchanges* (2002).

[30] SCHROEDER, B., AND GIBSON, G. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean too you? In *Proceedings of the 5th Usenix Conference on File and Storage Technologies (FAST)* (2007).

[31] TATI, K., AND VOELKER, G. On object maintenance in peer-to-peer systems. In *Proceedings of the 5th International Workshop on Peer-to-peer systems (IPTPS)* (Feb. 2006).

[32] TOIVONEN, R., ONNELA, J.-P., SARAMÄKI, J., HYVÖNEN, J., AND KASKI, K. A model for social networks. *Physica A Statistical Mechanics and its Applications 371* (2006), 851–860.

[33] YANG, M., CHEN, H., ZHAO, B. Y., DAI, Y., AND ZHANG, Z. Deployment of a large-scale peer-to-peer social network. In *USENIX WORLDS* (2004).