

# PeerStore: Better Performance by Relaxing in Peer-to-Peer Backup

Martin Landers  
Fakultät für Informatik  
Technische Universität München  
Boltzmannstr. 3, 85748 Garching  
landers@in.tum.de

Han Zhang, Kian-Lee Tan  
School of Computing,  
National University of Singapore  
3 Science Drive 2, Singapore 117543  
{zhanghan, tankl}@comp.nus.edu.sg

## Abstract

*Backup is cumbersome. To be effective, backups have to be made at regular intervals, forcing users to organize and store a growing collection of backup media. In this paper we propose a novel Peer-to-Peer backup system, PeerStore, that allows the user to store his backups on other people's computers instead. PeerStore is an adaptive, cost-effective system suitable for all types of networks ranging from LAN, WAN to large unstable networks like the Internet. The system consists of two layers: metadata layer and symmetric trading layer. Locating blocks and duplicate checking is accomplished by the metadata layer while the actual data distribution is done between pairs of peers after they have established a symmetric data trade. By decoupling the metadata management from data storage, the system offers a significant reduction of the maintenance cost and preserves fairness among peers. Results show that PeerStore has a reduced maintenance cost comparing to pStore. PeerStore also realizes fairness because of the symmetric nature of the trades.*

## 1 Introduction

A typical PC user today has so much data in his (her) hard disk that performing a regular backup has become a cumbersome activity: the amount of backup data may not fit into a single backup device (e.g., disk, CD-ROM, ZIP-DISK), (s)he must ensure these backup storage are carefully kept in a safe place, (s)he must remember the versions that have been backed up, and so on.

Interestingly, the capacities of modern hard disks have outgrown the needs of many users, leaving them with much idle storage space. While this space cannot be used to back up one's own data, a collection of PCs can be "connected" in a collaborative fashion to utilize their free space to backup one another's data. This has prompted several recent studies (e.g., pStore [1], Pastiche [3]) to design cheap, trans-

parent and efficient backup solutions in peer-to-peer (P2P) networks.

The steps of a P2P backup process are: the user indicates the data to be backed up, and the system distributes the backup data to other peers, probably making multiple copies to ensure availability of the data even when some peers storing the data become unreachable. Each node has to contribute a certain amount of space to back up data from other peers to make the system work. Designing a P2P backup system is challenging: the P2P network is inherently dynamic (nodes join and leave anytime), fairness must be enforced to prevent free-riding, the maintenance traffic has to be kept at an acceptable level and heterogeneous needs and capacities of nodes have to be considered.

In this paper, we propose a novel P2P backup system called PeerStore. PeerStore distinguishes itself from existing systems by decoupling the metadata management from the actual backup data storage. This has several advantages: (a) Different strategies can be employed to meet the different needs of the two subsystems: efficient searching for the metadata layer and flexible data placement for the storage mechanism. (b) The storage layer can be tailored towards fairness (c) The metadata can be used to quickly locate all replicas of a block, even if the replicas are stored at disparate locations, (d) Because metadata is comparably small, we can maintain the metadata *aggressively* to keep the information up-to-date.

In PeerStore, metadata management is accomplished by using a distributed hash table (DHT). By storing the metadata records this way, duplicate detection can be done efficiently. At the same time no real data needs to be migrated when nodes join and leave the network; only the information contained in the metadata records needs to be transferred and updated which largely saves the maintenance cost. Data storage, on the other hand, relies on a *symmetric trading* scheme. A peer that wants to backup its data must also store some data from each of its trading partners. We have implemented a simulation model of PeerStore, and evaluated its performance against pStore. Our

results show that PeerStore incurs less data migration overhead than pStore. Moreover, PeerStore’s maintenance overhead is significantly lower than that in pStore.

The remainder of this paper is organized as follows to present more detailed information: Section 2 discusses related systems. Section 3 gives an overview of the PeerStore design, while Section 4 introduces important components of the system in greater detail. Section 5 presents our experimental results from simulating the system, followed by a brief analysis of the data. Section 6 discusses weaknesses of the current system and outlines further research directions in Peer-to-Peer backup. Section 7 concludes the paper, summarizing our results.

## 2 Related Work

A number of Peer-to-Peer backup systems have been proposed in the last few years.

pStore [1] is an incremental backup system based on Chord [13]. It splits the files into equal-size data blocks, storing them in a distributed hash table (DHT). Blocks with identical contents will be shared among all peers, lowering storage requirements, especially when different versions of a file only show minor modification.

Cooperative Internet Backup Scheme [8] puts a strong focus on fairness: it forces all peers doing backup to exchange disk space in a symmetric manner. To facilitate these trades a central “Matchmaking” server is used. Peers create a “virtual disk” using Reed-Solomon erasure codes. The scheme cannot exploit overlap between peers.

Pastiche [3], similar to Cooperative Internet Backup, uses pairs of peers for backup. Using directed random walks in a DHT, peers in Pastiche try to find “buddies” with a similar set of files. All data is kept in a special file system, and buddies only exchange data not common for both.

Samsara [4] is a general fairness mechanism for Peer-to-Peer storage systems. It uses *storage claims* to reserve space for a peer at its partners, thus converting asymmetric trades into symmetric ones.

Venti-DHash [12], similar to pStore, uses a DHT to store data. It is a backend for the Venti backup system, storing disk blocks in DHash, a replication enabled DHT.

PAST [5] and OceanStore [6] are storage systems not primarily aimed at backup. PAST is a persistent Peer-to-Peer archival mechanism based on a Pastry DHT [11], offering replicated storage of *immutable* files. OceanStore mainly deals with dynamic, nomadic data that is frequently modified, but does offer an archival mode. This mode uses erasure codes to create replicated backup fragments of data.

None of the existing systems is well suited for a scenario where a private user backs up her data on the Internet. The systems either fail to offer good performance in unstable networks, or they have other issues preventing a simple

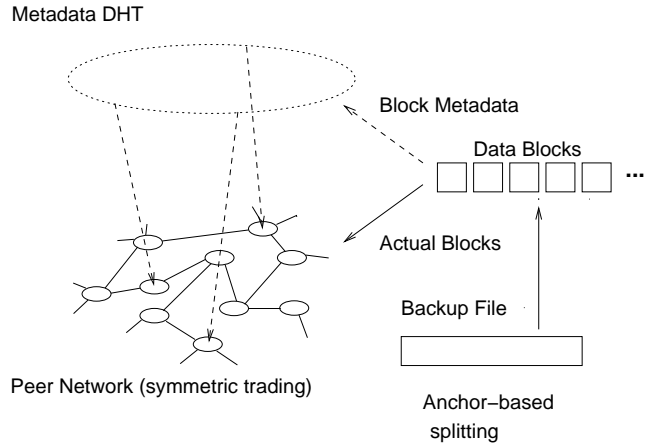


Figure 1. PeerStore overview

“home use”. PeerStore tries to address this, bringing P2P backup one step closer to widespread adoption.

## 3 Design of PeerStore

The key idea of PeerStore is to decouple metadata management from actual data storage. Decoupling allows us to use different Peer-to-Peer topologies in the layers. The metadata layer is based on a DHT, since it provides fast exact searching, needed to support duplicate detection. The storage layer, on the other hand, is based on symmetric trading [8] in an unstructured network, because this approach offers much better flexibility and avoids the high maintenance cost of a DHT. Furthermore, the symmetric nature of the trades considerably eases the addition of a fairness mechanism. Both overlay network span the entire network of peers. Figure 1 gives an overview of the system structure. The files to back up are split into blocks, encrypted and stored at partners in the symmetric trading layer. To efficiently retrieve blocks later, a metadata record is maintained for each block. These metadata records are stored in the *Metadata DHT*. This also enables efficient duplicate checking.

### 3.1 Backup

Before interacting with the network, a peer starts by splitting the backup files into data blocks. PeerStore then generates unique block identifiers and encrypts the data blocks. Both splitting and encrypting the blocks are done in a fashion that enables peers to share identical data in backups. Each file is represented as a list of unique block identifiers.

Before partners for storing the blocks are sought, PeerStore first eliminates all blocks that already have a sufficient number of replicas in the network. The number of replicas of

each block is determined by consulting the metadata DHT. This step avoids wasting storage space for duplicate data. After this elimination step, PeerStore starts looking for trading partners and creates new replicas of blocks. Once all blocks have a sufficient number of replicas, the trading process finishes. In a last step, the metadata records are updated to reflect the new replicas created.

### 3.2 Restore

The restore operation in PeerStore is similar to pStore: in order to restore a file, a peer first obtains the list of block identifiers for each version of a file (the *File Block List* in pStore [1]), indicating which data blocks are needed to re-assemble a certain version of the file. Using the metadata layer, the peer determines the *keepers* of each block, storing replicas, and downloads the block from one of them. Once a copy of each block has been obtained, the blocks can be decrypted and the original file reassembled.

### 3.3 Fairness

There are two interrelated aspects to fairness in a Peer-to-Peer data storage network: *safekeeping* of data and *fair contribution*. The main concern of a fairness mechanism in PeerStore is *safekeeping*, as the trading system naturally takes care of fair contribution once safekeeping is ensured. With PeerStore's trading system, peers only make trades which contribute resources equal (or close to) their demand, keeping the total amount of resources in the system balanced. As a result, no free-riding is possible under this scheme. But as peers might not keep their promises, rejecting blocks that a partner tries to store on the donated space or even silently discarding blocks, a mechanism to monitor data stored at other peers is needed. We propose that peers regularly challenge each other to verify the partner is still storing the blocks entrusted to it.

To make this monitoring effective, a peer failing to meet the challenges must be punished. In PeerStore punishment is achieved by discarding information that a peer has stored on a partner. However, punishing a peer that fails to answer a single challenge might lead to peers experiencing technical failure or downtime losing their backup, while, on the other hand, a too relaxed strategy will encourage malicious behavior. Samsara [4] offers a possible solution by combining spot checks with a *probabilistic punishment* model. For each challenge a partner fails to answer within a reasonable amount of time, a peer discards a small random set of replicas, with *exponentially* increasing probability. Since replicas are discarded randomly, the chances of losing all replicas of a data block are very low when the initial challenges are missed, but grow very fast for each consecutive unanswered challenge. When combined with the symmetric

trading scheme this peer monitoring model offers an effective fairness mechanism.

### 3.4 Short-term vs long-term availability

Data migration costs are a major contributor to the overall maintenance costs of a Peer-to-Peer storage system. Data migration costs are incurred by two things: re-creating "lost" replicas in order to achieve high short-term availability, and repairing data misplacement in a DHT. While the latter is not a problem for PeerStore (since no actual data is stored in the DHT), the former strongly influences the resource requirements of the system. In unstable networks, the cost of aggressively re-creating replicas can easily be unbearable. To support such networks, PeerStore focuses on providing long-term availability (i.e., individual blocks may be unavailable temporarily if all keepers of a block are offline, but in the long run all blocks can be obtained). When peers leave the network, no immediate action is taken. The metadata record of a block is just a list of all the peers that are known to store a replica, regardless of whether they are online or offline. No direct data migration cost is incurred by leaving peers. The re-creation of replicas is triggered by the infrequent challenges between partners instead, leading to a low data migration cost. If a peer finds that one of its partners is not answering to challenges for a too long time, it replaces the partner with a new one. If the challenging mechanism between partners is working correctly, no data should be lost *permanently*. Problems with this lazy strategy are the possibly long waiting time to finish a restore operation, and the uncertainty in deciding whether a partner is temporarily off-line or has left the network permanently.

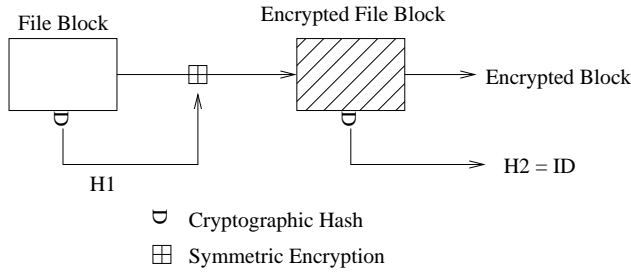
## 4 Basic Components in PeerStore

In this section, we introduce important components of PeerStore in greater detail.

### 4.1 File Blocks and File Block Store

The backup data and metadata is organized in the same fashion as in pStore [1]. Each file in a backup is represented by a number of encrypted data blocks and a descriptor (file block list) that lists all the blocks needed to reconstruct a particular version of the file. The descriptor is simply treated as another data block. To split files anchor-based indexing [9] using Rabin fingerprints [10] is used, to ensure peers create the same blocks for identical byte sequences. The unique identifier of each block, *ID*, is generated using the following formula:

$$ID = h(h(C))$$



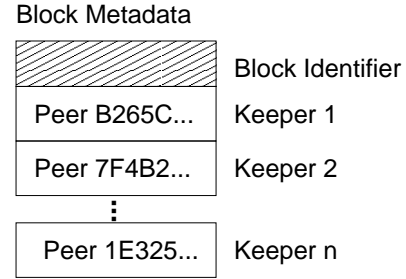
**Figure 2. Block encryption and identifier creation in PeerStore. Blocks are encrypted using convergent encryption which ensures identical blocks yield the same ciphertext and identifier.**

where  $C$  is the content of the block and  $h$  is a cryptographic hash function. Each block is encrypted using *convergent encryption* [2]. Under this scheme, the hash value of the block content,  $h(C)$ , is used as key for a symmetric cipher used to encrypt the block (for this reason we cannot use  $h(C)$  as block identifier directly). With the combination of these techniques, peers generate the same ciphertext blocks and identifiers for identical byte sequences in files. Combined with the nature of a DHT, this allows us to efficiently share duplicate blocks between different peers, while only allowing the legitimate owners of a block (peers in possession of the key ( $h(C)$ , stored in the file descriptor) to read the block’s contents. The process of creating the block identifier and encrypting the data block is shown in Figure 2. Every node maintains two block stores for different purposes. The “candidate” blocks are kept in the *temporary block store* before they are examined and sent out. The *temporary block store* will always be empty after all the blocks have been backed up. The *real block store* contains blocks stored on this node by other peers and is never modified or deleted on conforming peers.

## 4.2 Metadata Layer

The metadata layer’s responsibility is to achieve efficient data block retrieval and duplicate detection. It is implemented on top of a distributed hash table. For each data block generated by the peers, a block metadata record is created to record the existence and location of this block’s replicas in the system. Using the blocks unique identifier,  $ID$ , as key, the metadata record is inserted into the DHT.

Each block metadata record, as depicted in Figure 3, maintains a mapping from a block’s unique identifier,  $ID$ , to a list of *keepers* - peers storing a replica of the block the identifier is associated with. This offers an efficient way of locating all replicas of a block given its identifier. If there is



**Figure 3. Internal structure of a metadata record. The record maps a unique block identifier to the list of keepers storing replicas of this block.**

no metadata record matching a block’s identifier, the block is considered not to be present in the system.

During restore, a peer simply queries the metadata DHT to retrieve the metadata records for all the blocks it wants to restore and retrieves the real data blocks directly from any of the *keepers* listed inside the metadata records obtained. The availability of all the metadata records is crucial to the proper functioning of the system. For this reason, the metadata DHT needs to be maintained aggressively, to provide high short term availability. We propose using a scheme similar to DHash [12], that stores replicas of each metadata record on a number of consecutive peers in the DHT, allowing fast recovery if a peer fails. We use a simplified implementation, that broadcasts updates and synchronizes replica sets by complete retransmission.

While we cannot avoid the relatively high cost of aggressive maintenance here, the total maintenance cost is significantly reduced (compared to pStore) because of the small size of the metadata records: the metadata records are at least one order of magnitude smaller than the actual data blocks.

## 4.3 Identifying Keepers

In order to directly communicate with the keepers storing the block to facilitate block retrieval, a mechanism of uniquely identifying peers is needed. We propose implementing a *lookup* service as an additional functionality of the metadata DHT, using hash values - suitable as DHT keys - as *permanent* identifiers for peers, and storing the current address binding of each peer in the DHT. This address binding must also be maintained for high short-term availability, lest the lookup fail frequently.

## 4.4 Symmetric Data Trading

Symmetric trading allows PeerStore to implement a straightforward fairness mechanism and to deal with peer

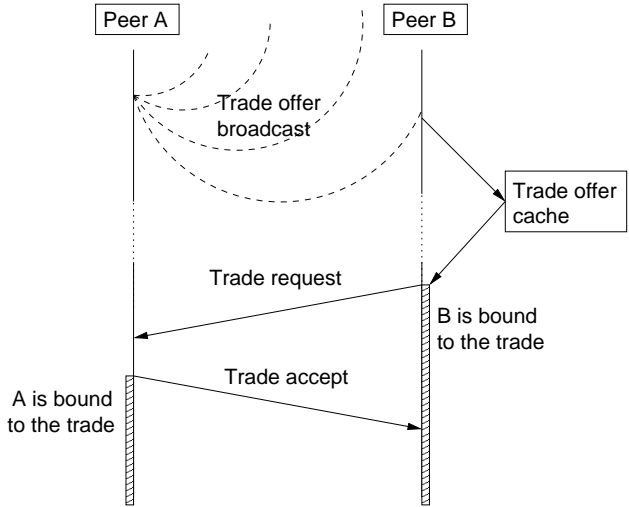
heterogeneity. The symmetric nature of the trades guarantees that data is distributed among peers with respect to their consumption of system resources, and it ensures that peers can punish misbehaving partners by dropping their data (which usually is not the case in a DHT-based scenario). Since there is no restriction on the trading size, peers with different storage requirements can find a matching partner. Lastly, the stored data blocks are not subject to data migration (as is needed in a DHT to ensure a correct identifier-peer mapping), which saves a lot of maintenance and network traffic compared to a typical DHT-based P2P backup system.

**Making a Trade** As illustrated in Figure 4, making a trade is a three step process:

- When looking for a new partner, a peer (A) broadcasts a *trade offer* to the network in a limited-scale random broadcast. The offer includes the amount of data the node requires at a potential partner. An offer is not binding for the peer.
- Reacting to the offer, a peer (B) can send a *trade request* to the originating peer (A). The trade request contains the amount of storage B needs at A. Both peers have specified their demand at this point. The peer sending the trade request is bound to it (unless the trade is rejected).
- The originating peer (A) responds by either accepting or rejecting the trade, sending a *trade accept* or a *trade reject* message. If the trade is accepted, the peers are trade partners from then on.

Each node maintains a *trade offer cache*, in which recently received offers are kept for a later referral. Additionally, a list of trade partners is maintained. When a peer starts the trading phase, it needs at least one trade partner per block replica, to ensure all replicas fail independently. Storing multiple replicas of a block on one node does not increase the robustness of the system.

Ideally, a peer would have exactly the same number of partners as the number of replicas. This means, when a peer starts looking for new trades, it should rather try to seek potential trades from existing partners instead of broadcasting to find new ones. If no existing trade offers enough space to backup more data, a peer first tries to make a new trade with one of the existing partners by sending a *trade request* to the set. Only if all partners reject the trade, the peer starts searching for new partners. To avoid unnecessary broadcasting, the *trade offer cache* is checked first for possible matching offers. If there are no suitable offers, the



**Figure 4. Messages in establishing a trade**

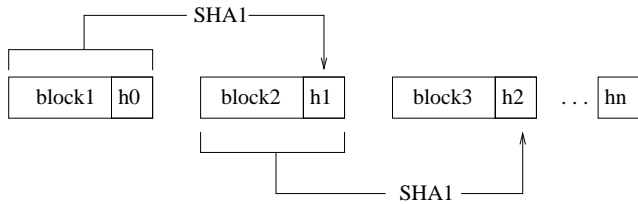
peer starts sending *trade offers* to the network using limited-scale random broadcasting based on the concept of a “lighthouse sweep” introduced by Pastiche [3]. The limited-scale random broadcast avoids the use of central server acting as “Matchmaker”, as in [8], while it ensures the broadcast does not consume too much network resources.

The nature of the *lighthouse sweep*, combined with the properties of DHT routing, guarantee that each trade offer broadcast reaches a distinct set of peers. However, as each *lighthouse sweep* only reaches  $O(d \log n)$  out of  $n$  peers – with  $d$  being the number of individual messages sent during the sweep – a peer may miss potential trade partners. This is a problem for peers with rare trading requirements (either exceptionally large or small). These peers might need to repeat the broadcast several times, or they might even fail to find a partner. In this case, a peer has the option of splitting its trade, or to accept largely imbalanced trades for a small trade, requiring the peer to provide much more space to the partner than it consumes.

**Trade Ratio** As exact matches between two partners are unlikely, each peer maintains a *trade ratio*, indicating how much imbalance in a trade a peer is willing to accept. The trade ratio  $r$  is expressed as the maximum ratio by which the amount of storage offered by a trade,  $S_o$ , may deviate from the amount of storage space sought by a peer,  $S_w$ , for a trade to be accepted. A peer will only accept a trade if

$$r \geq \frac{\max(S_o, S_w)}{\min(S_o, S_w)}$$

Trade ratio  $r$  is an indication of how much imbalance one peer is willing to afford. If  $r$  is set to 1, a peer only accepts exactly matching trades, severely reducing its chances of



**Figure 5. Safekeeping verification. By chaining the hash values of blocks in the depicted fashion a list of blocks can be challenged by only transmitting the values  $h_0$  and  $h_n$ .**

finding a partner. The larger  $r$  is, the higher the chances for a peer to find a matching trade. As  $r$  is symmetric, a peer increasing  $r$  not only increases its chances of finding a larger trade, but also a smaller trade.

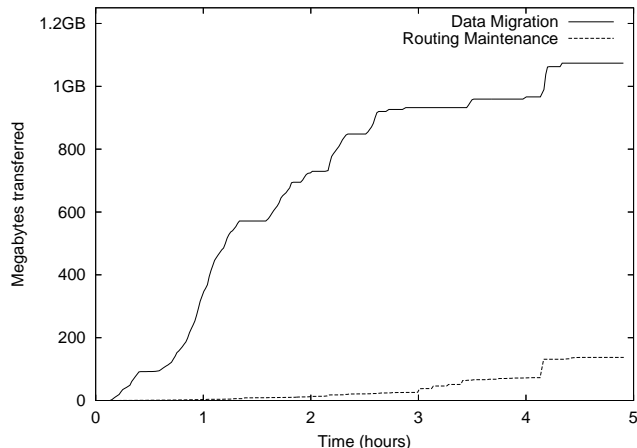
#### 4.5 Ensuring Safekeeping

In order to ensure safekeeping of its backup, a peer has to challenge all of its trade partners regularly, asking them to prove they are still storing all block replicas sent to them. However, retrieving all data blocks from the keepers is not a feasible solution as it consumes too much bandwidth. Considering we only need to verify the data blocks are still available, we propose the scheme shown in Figure 5: the challenging peer sends a unique value,  $h_0$ , along with the list of  $n$  data blocks to verify to the partner. To answer, the partner appends  $h_0$  to the first data block in the list and computes the SHA1 hash,  $h_1$ , of this concatenation.  $h_1$  is appended to the second block to compute  $h_2$  and so on. The partner only needs to return the  $h_n$  to prove that it is storing all the data blocks in the list.

Using this scheme, a peer challenges its partners occasionally. If a partner fails to answer a challenge within a reasonable amount of time (say a day or even a week), the peer drops a small, randomly-selected number of blocks it is storing for that partner. Every block is dropped with a certain probability  $p$ , that increases exponentially should the peer fail to answer subsequent challenges. Because each block is replicated throughout the network and every peer independently deletes blocks, the chances of losing all replicas of a block are low for a peer not answering only a few challenges. But peers that keep failing to answer challenges will soon lose data blocks.

## 5 Experimental Results

The goal of PeerStore is to provide Peer-to-Peer backup with low maintenance cost and good support for heterogeneity. The current DHT-based approaches suffer from

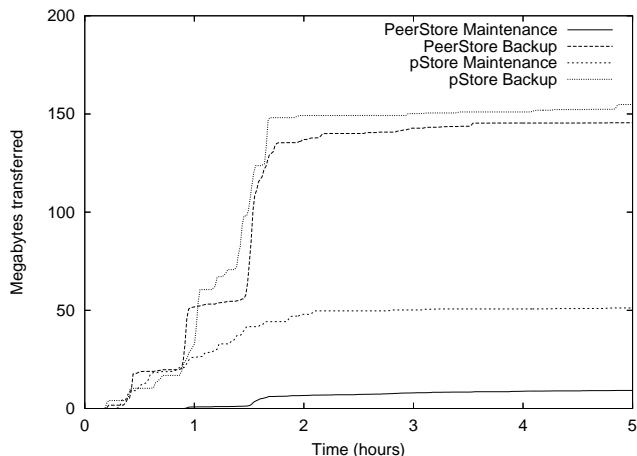


**Figure 6. Cumulative curves for data migration and routing maintenance cost of pStore in an unstable network. With a total of 100MB of backup data on 50 nodes, the data migration cost clearly dominates the DHT-maintenance cost of the system.**

having a high maintenance cost in unstable networks. Maintenance cost in DHT-based backup consists of routing maintenance cost and data migration cost. When nodes join or leave the network, the actual data stored on affected nodes needs to be transferred or replicated to make sure they are stored on the correct responsible node. This is the data migration cost. Traffic caused by re-organizing the Peer-to-Peer network is the routing maintenance cost. In PeerStore, we concentrate on reducing the data migration cost by relaxing the strict guarantees a DHT imposes on data placement. We focus on data migration, which has not received much discussion in the literature, but according to our experiments, is the dominating factor in DHT maintenance cost in unstable networks.

We ran two different simulations: the first one uses pStore to prove our forecast of the dominance of data migration cost in a DHT-based Peer-to-Peer backup system, the second one compares the performance of pStore and PeerStore, focusing on their maintenance overhead in the comparison. For both simulations, we chose a testbed approach, running actual implementations of the systems on a network of 50 computers running the Sun Solaris operating system. During the simulations, nodes join and leave the network at exponentially distributed time intervals, chosen from two distributions, reflecting the different mean values of up- and downtime. FreePastry was used as the peer-to-peer substrate.

For the first simulation – testing our hypothesis that data migration cost dominates total maintenance cost – we created

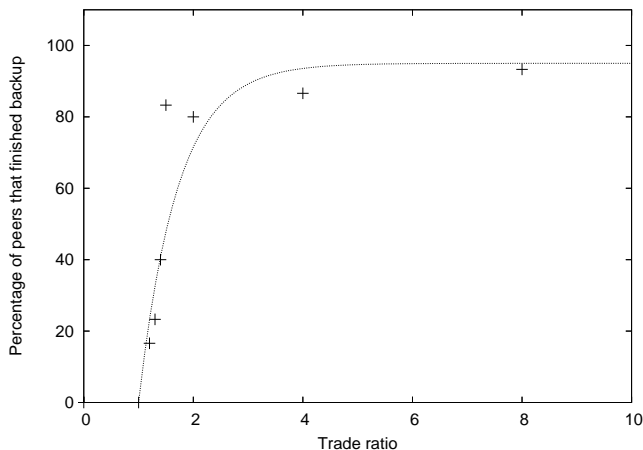


**Figure 7. Comparison of PeerStore and pStore backup and maintenance traffic. While the backup traffic (top curves) is similar for both systems, pStore generates significantly more maintenance traffic, as it stores larger amounts of data in the DHT.**

50 pStore nodes, each doing backup with a backup data set of 100MB. By instrumenting FreePastry, we were able to record incoming peer-to-peer traffic at each node. Figure 6 shows the results: While the routing maintenance cost is already significant, the data migration cost is several orders of magnitude larger. This result should be independent of the underlying DHT type because according to a comparison study of the performance of different distributed hash table under churn [7] is very similar.

The second simulation compares the performance of pStore and PeerStore. Again, both systems were run on a testbed setup of 50 computers. In order to do a fair comparison, we generate the same network scenario for both networks, that is, the number of nodes connected to the network at certain time, the time they start doing backup, the time they join network and the time they leave network are identical. We also use identical data sets for all test simulations.

As depicted in Figure 7, the amount of backup data traffic is quite similar for pStore and PeerStore. The backup data traffic is the traffic caused by actual backup operations, that is, the initial transfer of backup data from a peer to its partners in the network. As both systems have to backup the same data, we would expect the backup data traffic to be identical for both systems. However, PeerStore requires slightly less backup data traffic than pStore, because the metadata records in PeerStore enable the system to detect duplicate blocks *before* they are transferred over the network. pStore, on the other hand, needs to transfer



**Figure 8. Trade ratio vs. Percentage of successful backup. An increased trade ratio makes peers accept more imbalanced trades and thus allows a larger number of peers to successfully finish their backup.**

each block, as only the receiving node is able to judge if a block is a duplicate.

The maintenance cost for these two systems displays a sharp contrast. The main reason for this is the different amount of maintenance required for the DHT in each system. pStore, since it stores all data blocks directly in the DHT, must do actual data migration, moving data blocks between neighbors in the DHT, when a node joins or leaves. PeerStore, on the other hand, only requires metadata maintenance to synchronize the metadata among the set of consecutive peers affected, nothing else needs to be done. This figure shows that PeerStore requires significantly less maintenance traffic than pStore in unstable scenarios.

Since PeerStore contains a trading mechanism for data distribution, it is interesting to explore the effect of applying different *trade ratios* to the system. The result of this third simulation, which is shown in Figure 8, clearly shows that the higher the trade ratio, the higher percentage of peers are able to finish their backup. As the data points indicate, increasing the trade ratio only marginally in the range of [1, 2] allows a large number of additional peers to finish. Beyond 2, the effect of increasing the trade ratio is less pronounced, hinting at asymptotic behaviour.

## 6 Future Work

PeerStore still has several limitations: the main disadvantage of the proposal at the moment, shared with systems like Pastiche and Cooperative Internet Backup Scheme, is that it cannot guarantee a backup for every peer. If a peer is

unable to find a sufficient number of suitable trading partners, it cannot fully backup its data. Another problem is the fragmentation of backup sets introduced by the duplicate removal process, leading to the situation where a peer might not have a trade for each of its blocks. Usually this is no problem, as the blocks are covered by other trades, but can lead to silent block loss in the unlikely case that all peers directly involved in these trades lose interest. If the fragmentation is substantial it would also have a negative impact on the fairness mechanism, as trades are no longer symmetric in the sense that a peer can punish every other peer storing blocks for it. Finally, this complicates the inclusion of a deletion primitive. A security flaw PeerStore shares with all systems using convergent encryption, is allowing an attacker to probe backups for known data. All these limitations indicate future research areas.

## 7 Conclusions

By combining different peer-to-peer network topologies, PeerStore offers advantages in peer-to-peer backup in unstable networks. By separating the concerns of block management and actual block storage, PeerStore can offer good long-term availability with low maintenance cost. As a general result, we have shown that the data migration cost is the dominant factor in the maintenance cost of distributed hash tables in unstable networks.

For this reason, we expect to see more systems combining multiple peer-to-peer topologies, as none of the current topologies can on its own efficiently support tasks as different as searching and data distribution. Especially the combination of (small) distributed hash tables with less rigid mechanisms sounds promising, offering both good search performance and a high flexibility.

## References

- [1] C. Batten, K. Barr, A. Saraf, and S. Treptin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, MIT Laboratory for Computer Science, December 2001.
- [2] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 34–43. ACM Press, 2000.
- [3] Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of the Fifth ACM/USENIX Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.
- [4] Landon P. Cox and Brian D. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.
- [5] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS VIII)*, pages 75–80. IEEE Computer Society Press, May 2001.
- [6] John Kubiawicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*. ACM Press, November 2000.
- [7] Jinyang Li, Jeremy Stribling, Thomer M. Gil, Robert Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, San Diego, CA, February 2004.
- [8] M. Lillibridge, S. Elnikety, A. Birrel, M. Burrows, and M. Isard. A cooperative internet backup scheme. In *Proceedings of the 2003 Usenix Annual Technical Conference*, pages 29–41, 2003.
- [9] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Fransisco, CA, USA, 17–21 1994.
- [10] Michael O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-18, Harvard Aiken Computer Laboratory, 1981.
- [11] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
- [12] Emil Sit, Josh Cates, and Russ Cox. A dht-based backup system, August 2003.
- [13] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.