A Peer-to-Peer Personal Storage System

Adrien Ecoffet
UC San Diego
adrien.ecoffet@gmail.com

Fabrice Bascoulergue
UC San Diego
f.bascoulergue@gmail.com

ABSTRACT

Personal storage services have become increasingly popular in recent year, with services like Dropbox, Google Drive, Microsoft Skydrive or Apple iCloud[?, ?, ?, ?]. Because these services, by and large, are used to synchronize personal files across devices rather than to increase one's available storage space, It appears possible to devise a system in which users would share some of their existing disk space to be able to store files in the "cloud" for free. We propose a purely distributed peer-to-peer solution to that problem, and examine some of the challenges such a solution must face. We then present Jellyfish, our prototype for such a system, and the results of performance tests on a small cluster of Jellyfish instances.

1. INTRODUCTION

File synchronization services are more popular than ever today. This is in large part due to the increasing number of computing devices people have in their possession. On the other hand, commodity hard drives can now store huge amounts of data (typically 500GB to 2TB). Those two facts make it possible to design a system that would use unused space on user hard drives to provide an amount of cloud storage space proportional to the amount shared to the network by the user.

The system would store encrypted versions of the files on other users' filesystems so that files can be retrieved at any point in time by the user.

1.1 Related Work

Some similar systems have already been proposed. Clearly this somewhat resembles some distributed file systems, especially those that support disconnected operations, such as Coda[?], but some even more similar commercial services have also been created.

The first of them was probably Wuala[?], which initially featured a way to increase one's available storage space by sharing some disk space. Note that all files were still stored on their servers and that this was only used to increase performance. The feature has since been dropped.

Another example is SpaceMonkey[?], which allows users to rent large hard drives that are constantly connected to the Internet and are part of a peer-to-peer network which stores other user's files.

Very recently, Bittorrent announced Bittorrent Sync[?], which is very similar to our proposal. The differences we have seen are the following: Bittorrent Sync doesn't have a notion of an user account, rather users are expected to share a secret key across their devices. Additionally, Bittorrent doesn't ask the user how much of their hard drive it should share. It is unclear exactly how it determines that. Bittorrent Sync is also tracker-based, while our proposal is fully decentralized. Most importantly, Bittorrent hasn't released any paper or source code that we know of so far.

A company called Maidsafe is apparently currently developing a peer-to-peer system very similar to ours, but it hasn't been released so far. However, Maidsafe has developed and released an implementation of the Kademlia distributed hash table which we use in our implementation (its convenience is understandable since it was developed for this purpose exactly).

2. THREAT MODEL

A major issue in a decentralized system like this one is that of security. Here we consider the security of our design against three types of attackers.

2.1 Eavesdroppers

Eavesdroppers want to access user data, i.e. steal user files, steal user accounts, get some knowledge of exactly what files a given user has etc.

We simply use the typical way to protect oneself against eavesdropper: cryptography is used extensively in the system. User are each assigned a RSA private key which identifies them. The vast majority of operations require a signature from the user. Users also have a single AES256 key which is stored in the distributed hash table, encrypted using the user's public key (this is because AES cryptography is much faster than RSA). All files are encrypted using this AES key and a separate random IV. Some private user data is stored that way as well. Note that a separate key could also be used for each file, though this would incur a slight performance penalty.

Because we want to provide the abstraction of a user account rather than require users to transfer their private keys across their devices, private keys are actually store in the DHT, encrypted using an AES256 key that is derived from the user "password" using PBKDF2[?] (note: a better alternative for a production system could be scrypt[?], but PBKDF2 implementations are more widely available). This clearly makes the system vulnerable to dictionary attacks on passwords, so a real system should probably have a password strength verification module and should educate users about the importance of a strong password, especially in this context.

Note that an alternative would have been to generate the private key directly from the password using PBKDF2, but that would have made it extremely hard for users to change their passwords later.

3. EFFICIENT STORAGE

3.1 Data Flow

4. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

5. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

6. REFERENCES