A PERSONAL FILE SYSTEM OVER EMAIL STORAGES
Developing Towards a Persistent Internet Storage for Mobile Users

Chih-Yin Lin, Shih-Fang Chang and Chen-Hwa Song
Industrial Technology Research Institute,195 Sec. 4, Chung-Hsing Rd., Chutung, Hsinchu 310, Taiwan

Keywords: Email Quota, Internet Storage, Personal File System, Utility Computing.

Abstract: In 2004 Jones (Jones, 2004) developed a software tool called GmailFS that turns Google™ Gmail into a two-gigabyte file folder in the personal computer. Such virtual folder is not just an interesting innovation but quite useful as a backup depot hosted remotely at Google. GmailFS is technically sound but relies critically on the hypertext scripts of Gmail web pages. Thus, it can be easily averted by the mail server administrator with minor changes of the html scripts or web topology. In this paper, we use another approach to achieve the same goal of GmailFS and extend the capability to accommodate all email servers rather than just Gmail. The proposed system is named ePFS that utilizes three primitive email protocols, i.e. SMTP and POP3/IMAP, to respectively store and retrieve the data in and from the e-mail servers. In addition, we enforce security features of user privacy, data confidentiality and error recovery in ePFS. The proposed system is a work in progress heading toward a persistent Internet storage for mobile applications.

1 INTRODUCTION

For the past decade, the storage technology has expanded the disk space of a standard computer hard drive from several megabytes to hundreds of gigabytes. Although the disk space is still an important hardware resource, the average cost per storage unit has significantly declined. Therefore, the available space is now not as limited as it was when provided to software programs in a personal computer or to an individual user space in a networked service. In the near future, it might be possible for a server to allocate virtually unlimited space for every user that, ideally, would facilitate the development of all kinds of large scale Internet services.

Email is the primary service that almost everyone who has Internet access has at least one email accounts, e.g. school email, company email, Internet webmail, etc. A typical email service provides one hundred megabytes space for each user, if it is complimentary. Recently, there is a trend for email service providers to give its users free of charge a relatively large enough email quota. The email quota is now, for instances, two gigabytes in AIM mail (http://mail.aol.com), 2.7 gigabytes in Gmail (http://mail.google.com), five gigabytes in Inbox (http://www.inbox.com/), and unlimited in Yahoo mail (http://mail.yahoo.com/), etc. The quota of several gigabytes in practice allows an active user to accommodate almost all her/his emails without having to worry about exceeding the mail quota. On the other hand, however, for most users a lot of portion of this enormous space is simply left unused and wasted.

Lately there is an idea to make a different use of such large size email quota (Jones, 2004; Viksoe, 2005; Traeger et al., 2006), and some tools are being developed. The most famous tool is GmailFS (Jones, 2004), developed by Richard Jones no longer after Gmail's beta released. In essence, GmailFS mounts Gmail's email space as a networked drive in a Linux system. Although a two gigabyte space is not very large nowadays, as a remote folder hosted by a known service provider, it might be a good alternative for the file backup purpose (Chervenak et al., 1998). After all, such virtual folder is somehow more crash-free than the file system within any consumer level computers. The downside of the idea is that files placed in a virtual folder might lose the privacy and confidentiality since they are actually stored remotely at somewhere else with no security guarantee.

Inspired by GmailFS, in this paper we will propose a new personal file system named ePFS to employ email spaces as virtual file folders. ePFS

Lin C., Chang S. and Song C. (2007).
A PERSONAL FILE SYSTEM OVER EMAIL STORAGES - Developing Towards a Persistent Internet Storage for Mobile Users.
In *Proceedings of the Second International Conference on Wireless Information Networks and Systems*, pages 195-200
DOI: 10.5220/0002147801950200

uses three primitive mail protocols, simple mail transfer protocol - SMTP (Postel, 1982), post office protocol version 3 - POP3 (Myers, 1996) and Internet message access protocol - IMAP (Crispin, 2003), to respectively function as the operations of file storing and retrieval. Moreover, we consider several security features in ePFS, including confidentiality, integrity, and availability. With these features, ePFS can provide access to email space as access to a local file folder, and in a more secure manner. As a software application, ePFS supports friendly user experiences of file storing and retrieval, and outdoes GmailFS with a unified method and interface to communicate with different email servers.

The idea and implementation of using POP3/ SMTP to backup files was proposed in 2006 and the system being developed is called MailBackup (Traeger et al., 2006). Their prototype uses optional encryption to provide data confidentiality. However, it does not employ any fail recovery mechanism. The file being processed is encrypted and sent as a whole to an available email server, which is not loss guarantee once the server fails.

In the long run we hope to improve ePFS to support mobile devices like cellular phones with a unified, effective and efficient virtual storage access via Internet. ePFS is a work in progress that validate the concept of using today's large email quota as virtual file folders. In this paper, we will show how ePFS is designed and constructed. We consider ePFS the first phase development of a persistent Internet storage for mobile users. Eventually, the system will provide mobile users and mobile applications with virtually unlimited and secure storage.

ePFS has the following advantages:

- Apply to all email servers that support SMTP and POP3/IMAP protocols;
- Provide guaranteed retrieval of remotely stored files with a tolerance ratio of server failures;
- Dispatch sensitive files to email servers of higher degree of trust;
- Dispatch frequently accessed files to email servers of better availability;
- Assure the confidentiality of user data and files stored at email servers;
- Assure the integrity regarding the data being retrieved from the email servers;
- Support the integration of Windows™ file manager user interface.

2 RELATED WORKS

We briefly review two related works, GmailFS and Parchive (http://parchive.sourceforge.net/). GmailFS is a Linux platform software that magically transforms Gmail storage into a file folder. Parchive is a data parity checking and archive tool using Reed-Solomon codes (Plank, 1997; Plank & Ding, 2005). We employ Parchive in ePFS to against server failure when retrieving files from the email server.

2.1 GmailFS

Gmail is a free, search-based webmail service provided by Google™. Like its Internet search engine, there is no fancy user interface or complicated functions, but simple control tabs and large 2.7 gigabytes email storage. The free storage space that a Gmail account provides is larger than almost all free Internet storage services. It soon becomes a trend igniting free emails to upgrade their services for their users with mail quota in terms of gigabytes. Some major free email services, e.g. Microsoft, Yahoo, and AOL, are as depicted in Table 1.

Table 1: Comparison of some free email services.

	Gmail	Hotmail	YahooMail	AIM Mail
Space	> 2.7 GB	1 GB	*unlimited*	> 2GB
Max. message size	10 MB	10 MB	10 MB	16 MB
C/S protocols	POP3	POP3	POP3	IMAP
Virus scan	Yes	Yes	Yes	Yes
Spam filter	Yes	Yes	Yes	Yes

In August 2004, Richard Jones tried to utilize the large space provided by Gmail in a different way and released the first version of the famous software tool called GmailFS (Jones, 2004). GmailFS provides a mountable Linux file system that uses a valid Gmail account as its storage space. Almost all UNIX/Linux file manipulative instructions can be used to access files stored in GmailFS. For GmailFS, it uses the libGMail (http://libgmail.sourceforge.net/) to access email through the Gmail web user interface, and then translates email actions to file operations, applies the file operations to a user space virtual file system created by an application interface FUSE (http://fuse.sourceforge.net/). LibGMail is a Python binding to provide access to Google's Gmail web-mail service, where FUSE is a Linux kernel module bridges actual kernel to a user space file system. The system architecture of GmailFS is shown in Figure 1.

There are some drawbacks in GmailFS. Firstly, it does not provide confidentiality regarding the file

when transferring over the network or storing into the email server. Anyone with network sniffing skills can easily listen and obtain the plaintext of the file content. Like all email servers, the administrator of Gmail can access all emails and files stored in Gmail system. Secondly, The GmailFS does not work once Gmail changes its web user interface. This is the most critical disadvantage of GmailFS and the reason why it has to update its version very frequently to reflect the updates of Gmail. For the purpose of consistent access, since GmailFS works for Gmail only, its stability relies on the accessibility and availability of Gmail. Moreover, if Gmail crashes, or gets blocked behind the firewall, its user has no way to acquire the files.

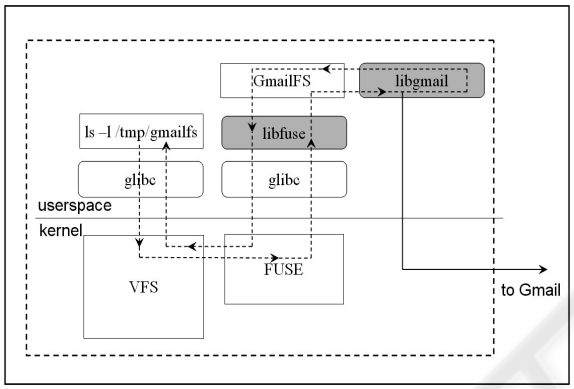

Figure 1: GmailFS system architecture.

GmailFS is not the only attempt that covets after Gmail accounts for its 2.7 gigabyte storage. Firefox GSpace (https://addons.mozilla.org/firefox/1593/) is an add-on with the same purpose and functions, which allows users to virtually transform their Gmail space into personal file storages. Unlike GmailFS works on Linux platform, similar tools include GMail Drive (Viksoe, 2004) that works on Windows platform, gDisk (http://gdisk.sourceforge.net/) that works on Mac OS X, and GDrive (Kane, 2006) developed by Google that will soon become one of its new services. However, all of them are designed in dedication to Gmail only, which is an unwelcome constraint.

Furthermore, as a storage tool, or as a personal file system, no user would like to count on single email portal that is not controlled by his own for his files' stability and viability. In addition, the file being stored at Gmail using GmailFS is not confidentiality guaranteed. There is no recovery mechanism if the files are damaged, and there is no checksum mechanism if the files are compromised. These drawbacks devalue GmailFS in practice.

2.2 Parchive

Parchive is a tool designed to collect a set of files and allows recovery when one or more of the files are lost. Parchive implements Reed-Solomon error correction coding mechanisms (Plank, 1997; Plank & Ding, 2005) with two different and format incompatible versions, Par and Par2. These two versions are optionally incorporated by the USENET newsgroup to process files posted to it, which is a well-known application of Parchive.

When backup a file, a straightforward technique to against possible server failures or data loss is simply storing more replicates at different servers. The tradeoff here is the server resource, which is too expensive to be a practical option. Therefore we use Parchive to balance the resource and the recovery requirement. Unlike USENET, we deal with one file instead of a set of files. Thus, the recovery tool will work on the blocks that are slices of a file. In specific, blocks are defined as the proper divisions of a file that all of them together can reconstruct the original file.

Assume a data file *f* is going to be stored with the recoverable feature against server failure. *f* is firstly sliced into smaller equal-sized blocks b_1, b_2, …, b_n that all blocks form a set often called *recovery set*. The input of Par2 is the recovery set and its output is a set of redundant blocks d_1, d_2, …, d_m, called *PAR set*. The size of PAR set is determined by the size of recovery set and an adjustable parameter *R* called *recovery ratio*. R equals to $n/(n+m)$ and is defined by the file owner. When the file needs to be reconstructed, the collection of any *n* out of the $(n+m)$ blocks can re-build all the blocks in the recovery set and the PAR set, and consequently reconstructs the file. Notice that in the paper we denote *P set* as the joint of the recovery set and the PAR set.

3 THE PROPOSED SYSTEM

The proposed ePFS is a personal file system over email storages. All files in ePFS have two security properties: sensitivity and frequency that, sensitivity is the importance of the file and frequency is how often the file might be used by the user. All email servers in ePFS have two relative properties: trustworthiness and availability, which respectively imply how much the user trusts the email server and how stable the server is. We consider these properties for the current version of ePFS and will

include more necessary ones in the future development. Note that how to evaluate these parameters and how to update them are not within the scope of this paper.

There are two basic operations in ePFS, the file storing operation *Proc_FileStore* and the retrieval operation *Proc_FileRetrieve*. *Proc_FileStore* takes in a file, protects it by encryption, slices it into blocks, generates redundant blocks with Par2, distributes the blocks to appropriate email servers according to their attributes, and updates the file distribution table for where these blocks are stored. *Proc_FileRetrieve* takes in the file identifier, checks the file distribution table, contacts available email servers to retrieve enough number of blocks, rebuilds the encrypted file with Par2, and finally decrypts it to reconstruct the file. *Proc_FileStore* functions immediately when a file is placed into the ePFS folder, and *Proc_FileRetrieve* functions when a file in the ePFS folder is being moved out.

Notice that for servers supporting IMAP, ePFS reads the title first and determines if the mail should be downloaded; for severs supporting POP3 only, ePFS reads the title while downloading the mail, and discards those that doesn't match the retrieval condition. Moreover, ePFS provides a user interface to configure individual account information of different email servers, as shown in Figure 2. If the email server requires SSL/TLS connection (Dierks, 1999) to receive the user identity and password, a check box can be w ePFS.

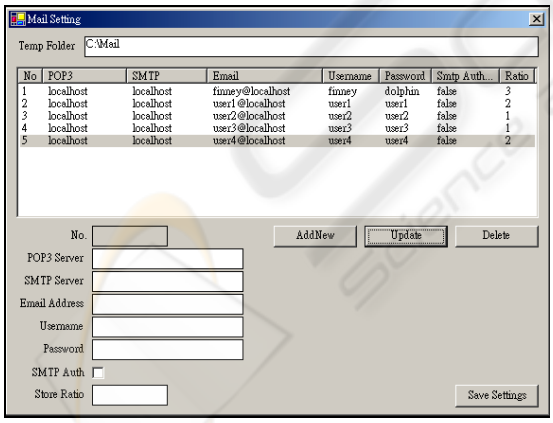

Figure 2: Email server configurations in ePFS.

Throughout this paper f denotes the file and its identity that will be processed by ePFS. All other parameters and notations include:

n: the number of blocks f is sliced into;

m: the number of redundant blocks generated by Par2;

$b_{f,i}$: i-th block in the P *set* of file f, where $b_{f,i}$'s are slices of encrypted f, for i = 1, 2, ..., n, and $b_{f,i}$'s are generated by Par2, for i = (n+1), (n+2), …,m;

θ_f : sensitivity value of file f, $0 < \theta_f < 1$;

δ_f : the value of frequency of use about f, $0 < \delta_f < 1$;

S_{id} : email server with identity id;

$T_{S_{id}}$: trustworthiness value of S_{id}, $0 < T_{S_{id}} < 1$;

$A_{S_{id}}$: availability value of S_{id}, $0 < A_{S_{id}} < 1$;

$Q_{S_{id}}$: the email quota of S_{id};

$E(\ )$: encryption operation with user key sk;

$D(\ )$: decryption operation with user key sk;

SP: the storage pool of all email servers with entries (S_{id} , $A_{S_{id}}$, $T_{S_{id}}$, $Q_{S_{id}}$);

DT_f : the distribution table of f that serially contains (n+m) S_i 's indicating that $b_{f,i}$ is stored at S_i .

In the followings, we describe *Proc_FileStore* and *Proc_FileRetrieve* accordingly. Figure 3 and 4 present respectively the conceptual views of these two procedures.

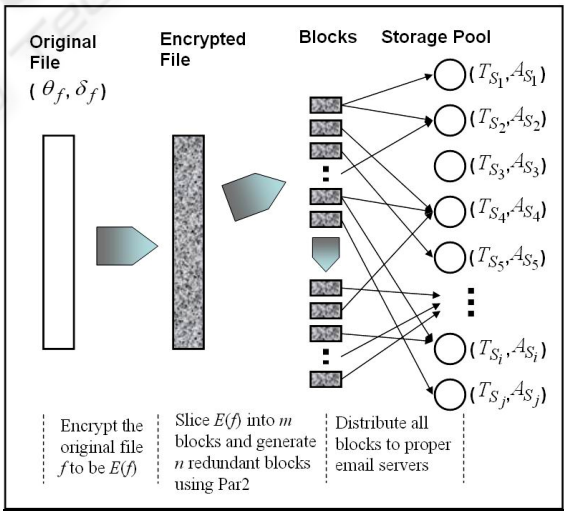

Figure 3: Store a file to email servers.

Proc_FileStore(f, θ_f , δ_f):

Step i. Encrypt f with sk;

Step ii. Slice $E(f)$ into n blocks $b_{f,1}$, $b_{f,2}$, …, $b_{f,n}$;

Step iii. Generate m redundant blocks $b_{f,n+1}$, $b_{f,n+2}$, …, $b_{f,n+m}$ via Par2 with input $b_{f,1}$, $b_{f,2}$, …, $b_{f,n}$, to constitute the P set of f;

Step iv. Select (n+m) S_{id}'s from SP satisfying $T_{S_{id}} > \theta_f$ and $A_{S_{id}} > \delta_f$;

Step v. Align selected S_{id}'s in random order and place them into DT_f;

Step vi. Send an email $eml_{f,i}$ with *f*-*i*-ddmmyyyy the title and $b_{f,i}$ the attachment to i-th S_{id} in DT_f using SMTP, for i= 1, 2, …, (n+m);

Step vii. Update SP by deduce the block size from $Q_{S_{id}}$ for each selected S_{id}.

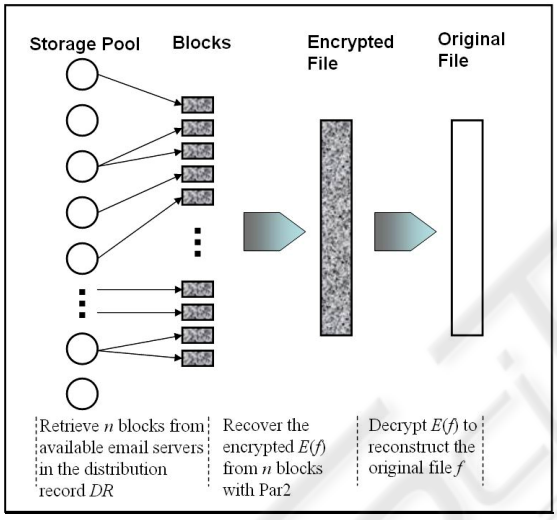

Figure 4: Retrieve a file from email servers.

Proc_FileRetrieve(*f*):

Step i. Denote S_{id}'s in DT_f serially as S'_1, S'_2, …, $S'_{(n+m)}$;

Step ii. Let j = 0 and k = n;

Step iii. Use POP3/IMAP to retrieve email $eml_{f,(j+i)}$ from $S'_{(j+i)}$ with email titled initiated with f, for i= 1, 2, …, k;

Step iv. Update j = k and update k to be the number of unsuccessful attempts in step iii;

Step v. If k <>0, goto step iii;

Step vi. Assume n collected emails be $eml_{f,i'}$, for i= 1, 2, …, n, and reconstruct $E(f)$ by feeding these $b_{f,i'}$'s to Par2;

Step vii. Decrypt $E(f)$ by sk to recover f.

Note that if there is no enough number of S_{id}'s in SP, or no enough number of S_{id}'s that satisfy the conditions of step iv in *Proc_FileStore*, , the user might have to adjust the values of n and m. If necessary, values of T_{S_i}, A_{S_i}, θ_f, and δ_f also have to be re-evaluated.

Moreover, when the email server S_{id} fails to provide the correct block $b_{f,i}$ previous stored in it, whether it be the connection failure or the damages found in the block, ePFS will invalidate the entry in the DT_f and pop out a notice to the user.

4 DISCUSSIONS

The design of ePFS is simple and effective. As an application level protocol, it integrates tools of state-of-the-art email protocols, security algorithms and error recovery mechanisms.

4.1 User Experience

The blocks sliced from $E(f)$ inherit the sensitivity and frequency attributes of the file f. During Step iv of the *Proc_FileStore*, ePFS selects proper email servers that satisfy the distribution conditions. Files that are more important to the user are dispatched and stored at places the user trusts more. The values of these attributes are adjustable and can be updated by the user.

When a file is pull into the ePFS folder to be distributed, ePFS runs in background to encrypt the file, connect to email servers and send out emails attaching sliced blocks. The user experience is consistent to the conventional file copy or move operation.

When a user selects a file f to be retrieved, ePFS takes about 2 to 10 seconds for the reconstruction, for a file of size about 400 KB. The performance varies due to the number of the blocks that directly incurs a great number of file I/O, which is the heaviest operation in ePFS. The current performance of file retrieval and reconstruction is absolutely not satisfactory to any user experience. In the future we hope to find the optimized values of block size, recovery ratio, etc, and to establish some kind of proxy scheme to speed up the process.

In ePFS, we also provide the function to view all blocks in a specific email server. This allows the user to manually check and clean unused blocks in the email server.

4.2 Security

In terms of security, ePFS assures the confidentiality of files being stored at email servers with AES symmetric encryption (Daemen & Rijmen, 2001). All files processed with ePFS are in practice confidential as long as the AES implementation is sound and robust. On the other hand, although files are not 100% guarantee retrievable in ePFS, the Par2 provides a very strong confidence of getting the files back with adjustable server failure ratio. The integrity of the files also relies on the Reed-Solomon codes.

There is another security issue that concerns user privacy when a user' network behaviour is maliciously monitored. In ePFS the file identity *f* is not enciphered, so even if *f* itself is not indicative, the occurrences of *f* are linkable to translate into meaningful behaviours. The privacy of user behaviours against the email server or any third party can be enhanced by enforcing a scrambling function to the file identity. That is, the file identifier *f* can be transformed into plural unlinkable aliases being applied to different email servers.

5 CONCLUSIONS

The storage of mobile devices is innate limited by the flash ROMs and memory cards, in this paper ePFS proposed a possible way to extend available storage by trading bandwidth for it. Although currently ePFS works on Windows PC only, it has shown the feasibility of utilizing email space over Internet. As the appearance of mobile devices continuously gets tinier and tinier, we believe the function that ePFS provides will soon be the need of mobile applications.

In order to shape ePFS, we will re-examine current file and server attributes to adopt more proper parameters. Along with ePFS, we use a small program to delete emails older than a specific date to release unused email space. We will develop a new garbage collection mechanism that takes file attributes into consideration. Moreover, the performance of block retrieval is a critical problem, so we will target some specific applications that do not require instant access to remote backups.

There are some future works based on ePFS that require continuous efforts. The porting of ePFS to a handset model is underway, i.e. we picked Windows Mobile platform. Besides email services, we will include more Internet storage resource like photo albums, video blogs, free FTPs, etc., to be the candidates of backup depots. In the long run we will try to enhance ePFS to support a persistent storage for mobile devices.

ACKNOWLEDGEMENTS

This research was sponsored by the Mobile Digital Life core technology development projects 2006-2009, monitored by the Ministry of Economic Affairs, Taiwan.

REFERENCES

Chervenak, A., Vellanki, V., Kurmas. Z., 1998. Protecting File Systems: A Survey of Backup Techniques. In *Proceedings of Joint IEEE and NASA Mass Storage Conference*.

Crispin. M., 2003. Internet Message Access Protocol version 4 revision 1. In *STD 1*, *RFC 3501*, IETF.

Daemen. J., Rijmen. V., Rijndael Specification. In *Advanced Encryption Standard*. NIST FIPS PUB 197.

Dierks, T., 1999. The TLS Protocol version 1.0. In *STD 1*, *RFC 2246*, IETF.

Jones, R., 2004. GmailFS: A Gmail Based Linux Filesystem. *http://richard.jones.name/google-hacks /gmail-filesystem/gmail-filesystem.html*.

Kane, M., 2006. Going for a GDrive with google. In *CNET News, 7, March 2006. http://cnet.com/2061-11199_3-6046686.html.*

Klensin, J., 2001. Simple Mail Transfer Protocol. In *STD 10*, *RFC 2821,* IETF.

Plank, J.S., 1997. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. In *Software, Practice & Experience*, Vol. 27 (9).

Plank, J.S., Ding, Y., 2005. Note: Correction to the 1997 Tutorial on Reed-Solomon Coding. In *Software, Practice & Experience*, Vol. 35 (2).

Traeger, A., Joukov, N., Sipek, J., Zadok, E., 2006. Using Free Web Storages for Data Backup. In *Proceedings of the Second ACM Workshop on Storage Security and Survivability*, ACM Press.

Viksoe, B., 2004. GMail Drive Shell Extension. *http://www.viksoe.dk/code/gmail.htm.*

Myers, J., 1996. Post Office Protocol version 3. In *STD 53*, *RFC1939*, IETF.