EMFS: Repurposing SMTP and IMAP for Data Storage and Synchronization

William Woodruff
william@tuffbizz.com

January 29, 2016
v1.0

Abstract

Cloud storage has become a massive and lucrative business, with companies like Apple, Microsoft, Google, and Dropbox providing hundreds of millions of clients with synchronized and redundant storage. These services often command price-to-storage ratios significantly higher than the market rate for physical storage, as well as increase the surface area for data leakage. In place of this consumer-unfriendly status quo, I propose using widely available, well standardized email protocols like SMTP and IMAP in conjunction with free email service providers to store, synchronize, and share files across discrete systems.

Contents

1 Introduction

Remote storage and synchronization of data, commonly referred to as "cloud" storage, has become increasingly popular with companies and consumers alike as both a redundancy measure and as a way to share informations across a diverse range of platforms. For companies, this reduces or eliminates the need to maintain internal network filesystems like NFS and CIFS. For consumers, having a remotely accessible copy of data simplifies common tasks like social sharing (pictures, video) and reduces the frustration of using multiple computers (desktop, laptop, smartphone) for a common task.

Cloud storage providers (CSPs) generally divide their service into a free tier and a paid tier. Free tier users are usually restricted in terms of how much they can upload, but may also be subject to slower upload speeds and limited trial periods. Paid users are also usually restricted in terms of total upload capacity, but at much higher limits and may not be subject to the same level of throttling as free users.

Beyond price concerns, the prevalence of CSPs has expanded the surface area for data breaches on both an individual and company level. Incidents like the Apple-owned iCloud leak [1] and security loopholes like Dropbox's lax public URL policy [2] reflect poor security choices by users and service providers alike. This is exacerbated by the widespread use of proprietary (and mutually incompatible) clients and protocols by CSPs, leaving users at the mercy of their provider of choice for updates.

There have been attempts to provide open-source and user-controlled alternatives to CSPs [3, 4], but all rely on a level of technical commitment and expertise comparable to managing a traditional networked filesystem. Since the goal of "cloud" storage is to eliminate the technical barrier to data synchronization, these alternatives are not currently suitable for the majority of CSP users.

As an alternative to both traditional CSPs and their open-source replacements, I propose a system that uses the already open Simple Mail Transport Protocol (SMTP) [5] and Internet Message Access Protocol (IMAP) [6] in conjunction with free email service providers (ESPs) to store, synchronize, and share data between both users and machines. I call this system the *EMail FileSystem*, or *EMFS*.

2 Email as a General-Purpose Storage Service

2.1 Usenet as a Precedent

2.1.1 Protocol and Topological Similarities

The topology of the Usenet network bears an uncanny resemblance to the contemporary email network, with clients connecting to a ring of relay servers responsible for distributing the latest posts to all recipient newsgroups.

The Network News Transport Protocol (NNTP) [7, 8], the dominant contemporary Usenet protocol, also closely resembles SMTP in message structure:

Listing 1: NNTP session

```
< CONNECT news.foo.com
> 200 news.foo.com NEWS
< POST
> 340 Ok, recommended ID
   ↪ <beef@news.foo.com>
< From: bar@baz.com
< Newsgroups: misc.quux
< Subject: Hello!
< Message-ID: <beef@news.
   ↪ foo.com>
< ↩
< Hello, World!
< .
> 240 article posted ok
< QUIT
> 205 Goodbye
```

Listing 2: SMTP session

```
< CONNECT smtp.foo.com
> 220 smtp.foo.com SMTP
< HELO smtp.foo.com
> 250 Hello!
< MAIL FROM: <bar@baz.com
   ↪ >
> 250 Ok
< RCPT TO: <quux@baz.com>
> 250 Ok
< DATA
> 354 End data with <CR><
   ↪ LF>.<CR><LF>
< Subject: Hello!
< From: bar@baz.com
< To: quux@baz.com
< ↩
< Hello, World!
< .
> 250 Ok: queued as 105
< QUIT
> 221 Goodbye
```

2.1.2 Usage Similarities

Usenet was originally developed to share text-based news and threaded discussions. As academic and social usage of the Internet expanded, Usenet users took advantage of the network's mirroring capabilities to share encoded binary files. The most notable Usenet hierarchy for binary sharing has historically been `alt.binaries.*`, with many providers and authorities limiting access to their entire `alt.*` hierarchy due to the abundance of illegal content [9, 10]. As of 2010, 99% of all traffic over NNTP is yEnc binary data [11]. Because Usenet mirrors its data across every provider it has also been used as a backup service [12], with users uploading encrypted data to globally visible newsgroups.

Email, like Usenet, was originally developed to share text-based messages, albeit between sets of addresses instead of discussion hierarchies. Just like Usenet, email adapted to share binary content as the needs of users diversified to include multimedia and rich document types. Unlike Usenet, email networks have never seen the proliferation of organized binary sharing.

2.2 Advantages over NNTP

There are a number of benefits to storing and sharing binary data over SMTP and IMAP instead of NNTP.

Unlike Usenet messages, email messages are not directed to a global hierarchy. The destination of an individual (SMTP composed) email may be a mailing list, N distinct addresses, or even the sending address itself. As a result, clients may exercise more autonomy over where their data is sent and when it requires encryption.

Although yEnc has emerged as a de-facto standard [11] encoding on Usenet, there is no *formal* standard for sending non-text data over NNTP. Consequently, Usenet newsreaders regularly suffer both false negatives and positives when fetching encoded binaries [13, 14]. By comparison, MIME-encoded binaries were introduced as an extension in 2000 and are now universally handled by rich email clients [15].

2.3 Advantages over CSPs

2.3.1 Security

The use of an extant email account for storage confers a number of security and usability benefits compared to a traditional CSP. Instead of having to register for a new service, users can access their files with their email credentials. This alone eliminates the attack surface normally associated with cloud storage, with no danger of data leakage from public URLs and half as many potential points of misconfiguration or attack.

The prevalence of TLS-secured open email protocols is also advantageous, as *EMFS* may take advantage of the existing email security and privacy infrastructure instead of implementing its own. In particular, STARTTLS for both SMTP and IMAP [16, 17] is universally supported by large email providers. Beyond transit security, an implementation of the open Pretty Good Privacy (PGP) standard [18] can be used to ensure storage security.

2.3.2 Extensibility and Transparency

To protect their intellectual property and remain competitive, large CSPs make extensive use of proprietary protocols and clients. Although these methodologies can be observed and analyzed [19], their closed nature hinders interoperability and locks users into vendor-specific ecosystems. Because *EMFS* operates solely on the email infrastructure and utilizes open protocols, it can be extended and modified trivially both by users and developers as demands evolve.

The presence of *EMFS* on a user's account is also fully opaque to the ESP, as *EMFS* traffic is identical to normal email traffic. From the user's perspective, the only evidence of *EMFS* is an IMAP folder of their choosing used to store messages. In the full spirit of the SMTP and IMAP protocols, this allows the user to treat their email client as a basic file manager by using standard email primitives (Compose, Edit, Delete) to manipulate synchronized data.

3 Architecture

Like any other filesystem, the architecture of *EMFS* can be broken into discrete primitives that can be categorized by type or operation. *EMFS* is also interacted with as a normal hierarchical filesystem, providing a tree whose root is the virtual mountpoint for the *EMFS* instance.

3.1 Filesystem Primitives

3.1.1 Types

EMFS is aware of two primitive types: files and directories.

3.1.1.1 Files *EMFS* files are chains of N email messages, both header and body, where N is greater than or equal to 1. In terms of lookup, a message chain for a given file F divided into N messages behaves like a linked list:

$$\begin{tikzcd}
\boxed{F_0 \mid \bullet} \arrow[r] & \boxed{F_1 \mid \bullet} \arrow[r] & \boxed{\cdots \mid \bullet} \arrow[r] & \boxed{F_{N-2} \mid \bullet} \arrow[r] & \boxed{F_{N-1} \mid \bullet} \arrow[r] & \boxtimes
\end{tikzcd}$$

Message chains represent literal data, with no facilities for UNIX-like symbolic linking. Two functions are required to generate the content in an *EMFS* message chain, an 8-bit encoding function *Encode8* and a hashing function *Hash*. *Hash* may be as simple as an iterative function.

An *EMFS* file might have a message-representation as follows:

Listing 3: EMFS Message

```
From:␣foo@bar.com
To:␣foo@bar.com
Subject:␣first-id-hash␣filename
EMFS-Filename:␣filename
EMFS-Next:␣next-id-hash

encoded-body
```

Where `first-id-hash` is the first ID hash in the sequence, `next-id-hash` is the ID hash of the next message in F, and the generic `id-hash` is found by:

$$\texttt{id-hash} \leftarrow \mathit{Hash}(\texttt{filename}, N)$$

And where `encoded-body` is found by:

$$\texttt{encoded-body} \leftarrow \mathit{Encode8}(\texttt{file-slice})$$

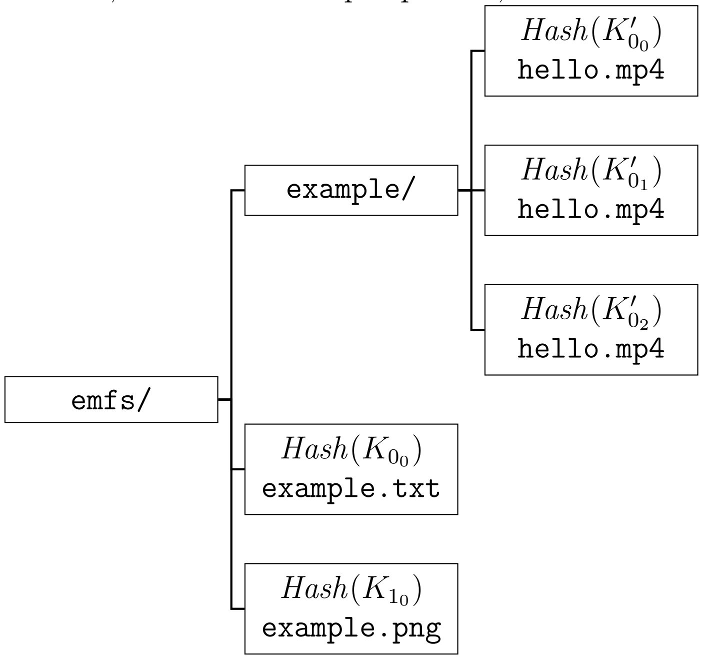
Where `file-slice` is the array of data of F_N.

3.1.1.2 Directories

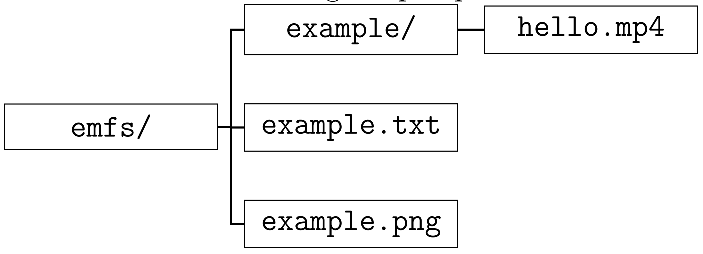
EMFS directories are mapped directly onto the IMAP notion of "folders". An IMAP folder is said to contain M messages for K files, each file split across K_N messages such that:

$$M = \sum_{F=1}^{K} F_N$$

As such, from the IMAP perspective, the *EMFS* hierarchy looks like this:


emfs/
example/
$Hash(K'_{0_0})$ hello.mp4
$Hash(K'_{0_1})$ hello.mp4
$Hash(K'_{0_2})$ hello.mp4
$Hash(K_{0_0})$ example.txt
$Hash(K_{1_0})$ example.png

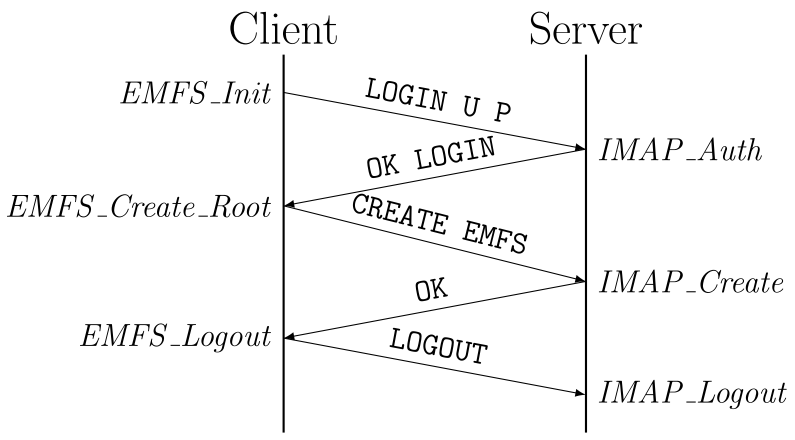
And from the file manager's perspective:



Files with only one message (i.e., $F_N \mid N = 1$) are constituted directly. More notably, `hello.mp4` is constituted into a single file from three messages (i.e., $F_N \mid N = 3$) under the IMAP `example/` folder.

3.1.2 Operations

3.1.2.1 Filesystem Creation Because the entirety of each discrete *EMFS* instance resides in its own IMAP folder hierarchy, the creation of an instance requires the creation of a root IMAP folder. This is accomplished by issuing a `CREATE` verb during a short IMAP session:



3.1.2.2 Directory Creation Like the creation of the *EMFS* instance, the creation of individual directories within an instance relies on the IMAP `CREATE` verb. To create nested directories below the root level in a fashion similar to UNIX's `mkdir -p`, the `SELECT` verb is also required. Shown iteratively:

Algorithm 1 Directory Creation

```
1: procedure EMFS_MKDIR(directory)
2:     delimiter ← the symbol used to delimit directories
3:     directories ← Split(directory, delimiter)
4:     for d ∈ directories do
5:         if ∄d then
6:             IMAP_Create(d)
7:         end if
8:         IMAP_Select(d)
9:     end for
10: end procedure
```

3.1.2.3 Directory Deletion

In the most extreme case, an *EMFS* directory deletion should remove all messages and subfolders in the corresponding IMAP folder in a fashion similar to UNIX's `rm -rf`. Because the IMAP `DELETE` verb will refuse to operate when given a folder with subfolders, `EMFS` must descend to all subfolders and perform `DELETE` on them first. Shown recursively:

Algorithm 2 Directory Deletion

```
 1: procedure EMFS_Rmdir(directory)
 2:     subdirs ← all subdirectories of directory
 3:     if subdir = ∅ then
 4:         IMAP_Delete(directory)
 5:         return
 6:     else
 7:         for d ∈ subdirs do
 8:             EMFS_Rmdir(d)
 9:         end for
10:     end if
11: end procedure
```

3.1.2.4 File Creation

The conversion of file data into a chain of messages is complicated by the lack of a standardized encoded message size limit across common mail servers [20, 21, 22]. For the sake of generality, the *EMFS* message chunking algorithm makes reference to this size limit as S.

Algorithm 3 File Creation

```
1: procedure EMFS_Put(filename)
2:     file ← Encode8(Read((filename)))
3:     slices ← file slices of size $\leq S$ of file
4:     messages ← EMFS_Pack(filename, slices)
5:     for m ∈ messages do
6:         SMTP_Send(m)
7:     end for
8: end procedure
```

Where *EMFS_Pack* is defined as follows:

Algorithm 4 Message Packing

```
1: procedure EMFS_PACK(filename, slices)
2:     messages ← an empty list
3:     size ← Count(slices) − 1
4:     for i ∈ Range(0, size) do
5:         id ← Hash(slices[i])
6:         next_id ← −1
7:         if i < size then
8:             next_id ← Hash(slices[i + 1])
9:         end if
10:        message ← Build_Message(filename, id, next_id, slices[i])
11:        Append(messages, message)
12:    end for
13:    return messages
14: end procedure
```

Where *Build_Message* constructs an SMTP envelope and body of the form specified in Listing 3.

3.1.2.5 File Deletion File deletion is a straightforward matter of following the hash chain after resolving the first node from the `EMFS-Filename` SMTP envelope header field.

Algorithm 5 File Deletion

```
1: procedure EMFS_DELETE(filename)
2:     message ← the first SMTP message in the file chain
3:     do
4:         next_id ← SMTP_Header(message, EMFS-Next)
5:         IMAP_Delete(message)
6:         message ← EMFS_Next(filename, next_id)
7:     while next_id ≠ −1
8: end procedure
```

3.1.2.6 Indexing Because operations that require file or directory discovery would become extremely expensive in terms of both network time and computation if each performed its own IMAP `LIST` verb, *EMFS* maintains its own cached index of the IMAP message hierarchy. This index is built at the beginning of each session and updated whenever an *EMFS* operation modifies the IMAP hierarchy.

4 Applying *EMFS* to Common ESPs

4.1 ESP Statistics

4.1.1 Google Gmail

Gmail is the world's largest free ESP, hosting over 900 million active users as of May 2015 [23]. It was also one of the first to offer large storage capacities for nonpaying users, beginning with 1GB and currently offering 15GB shared across the all services tied to a user's Google account [24, 25]. Gmail caps attachment size to 25MB [20].

4.1.2 Microsoft Outlook

Microsoft Outlook, previously Hotmail, is one of the earliest web-based ESPs. It is also currently the second largest ESP with a free plan, with 400 million active as of 2014 [26]. Outlook's free plan includes unlimited email storage and is not tied to quotas for other services on the same account [27]. Outlook caps attachment size to 20MB [22].

4.1.3 Shared Characteristics

Together, Gmail and Outlook store the emails of over 1.3 billion active users. They are both core components of mature corporations, and both have established themselves as standards for both personal and corporate email service [28].

In addition to their webmail interfaces, both give their users full SMTP and IMAPv4 access, including support for session encryption via TLS. Both offer large attachment sizes and large storage capacities (albeit ultimately limited in the case of Gmail). Together these qualities make Gmail and Outlook, as well as other large ESPs like Yahoo! and AOL, more than suitable as hosts for *EMFS* instances.

4.2 Potential Hurdles and Concerns

4.2.1 Countermeasures by ESPs

Because *EMFS* takes advantage of the storage offerings of ESPs without sending meaningful amounts of email traffic between distinct users, it's possible that large providers will take steps to curb such usage of their service. In addition, since many large ESPs are also CSPs, usage of *EMFS* on their free accounts may be treated as an attempt to circumvent payment for their services.

4.2.2 Sharing Between Users

Although IMAP allows email users to create folder hierarchies in their accounts and organize their messages into these folders, all ESPs employ a "standard" behavior of sending new incoming emails to the user's inbox folder. This behavior is in conflict with *EMFS*'s principles of invisibility and noninteraction with regards to "normal" email traffic. Although *EMFS* can veil this behavior on an individual email account by issuing IMAP commands to move *EMFS* messages to their dedicated hierarchy, sharing attempts between multiple email addresses may be complicated by a need to initially index all received mail for files sent before *EMFS* configuration by the recipient.

5 Conclusions

EMFS has many advantages over conventional CSPs, as well as some disadvantages.

In principle, *EMFS* is substantially simpler to use and secure than any CSP. Its simplicity derives from its use of the user's email account and credentials rather than a distinct account and credentials on a CSP, meaning that the user need only remember one login to access both files and emails. "Creation" of an *EMFS* store is the simple act of signing in to the client for the first time. It is also secured on a transport level by ubiquitous STARTTLS support in both SMTP and IMAP among large ESPs. Overall, the combination of simplicity in setup and a security ecosystem built on open standards makes *EMFS* an appealing alternative to distinct CSP accounts and unaudited encryption stacks.

In addition to its simplicity of operation, *EMFS* is also completely vendor independent. It makes a minimum number of assumptions about the capabilities of the servers it uses, allowing users to be flexible in their ESP choices. Although its structure is inspired by a desire to store data at no cost to the user, paid email plans are just as capable of hosting it.

With these advantages come some natural disadvantages. *EMFS* makes no attempt to provide a simple user-to-user synchronization mechanism equivalent to the "shared folder" idiom in cloud storage. It also does not provide an HTTP gateway for one-way file sharing, as many CSPs do. Neither of these tasks is impossible under the *EMFS* infrastructure, but both incur significant complexities and consideration in setup. *EMFS* performance is also heavily dependent on the good grace and bandwidth of the user's ESP.

Overall, *EMFS* is best suited to the personal storage and synchronization needs of a single user across multiple machines. Portability and platform support is limited only by the existence of a storage medium and a functional network stack.

6 Afternotes

6.1 Future Directions

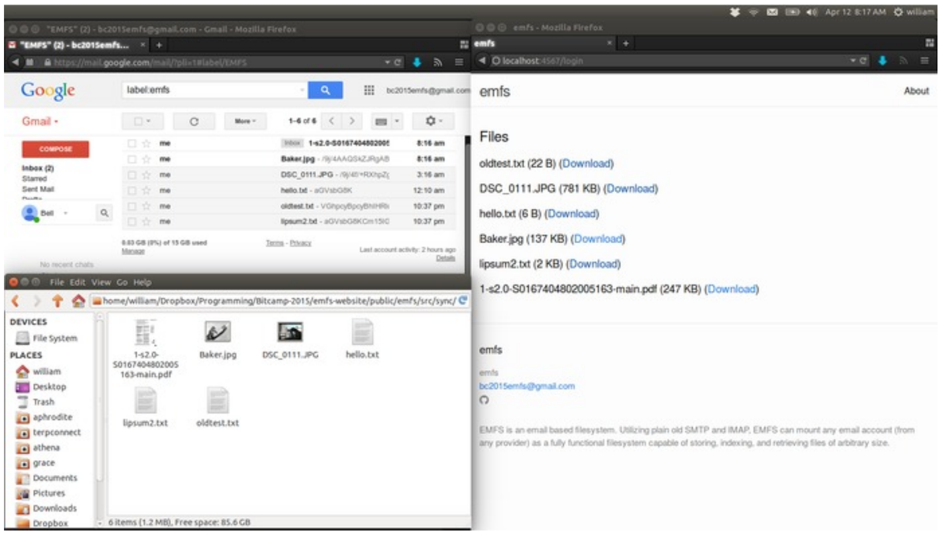
The potential use of PGP to ensure message security was noted in Section 2.3.1, but no approach was given. In principle, adding PGP to the *EMFS* architecture should be as simple as adding an encryption and decryption layer before and after each individual message transmission and retrieval.

A compression layer is another potential addition to *EMFS*, either applied early to the whole data being uploaded or after chunking (but before encryption). The application of this would be a performance tradeoff between (de-)compression times and long message chains. Depending on the frequency of access, performance of IMAP calls, and choice of hardware and compression algorithm, it may or may not be preferable to limit the length of message chains via such a layer.

Although this paper lays out algorithms for the most common *EMFS* operations, it does not provide much detail on aspects of synchronization from the user's perspective. The simplest potential synchronization mechanism would be an file event watcher, pointed at the directory of the *EMFS* "mount". As file events are observed, the appropriate *EMFS* operation could be dispatched. This has the benefit of being divorced from the heavy lifting done by *EMFS* in chunking and reconstituting messages into files.

6.2 Implementation

The first program resembling an *EMFS* implementation was written (well) before this paper, over a period of 48 hours at a hackathon. It lacks many of the abilities of the system described here, but provides a brief glimpse at a full user experience in the form of both web and virtual filesystem frontends. It can be found at `https://github.com/fcf634cbe8298176f7c576faed0e500a`, although the author does not advise its usage.



me	Inbox 1-s2.0-S0167404802005		8:16 am
me	Baker.jpg	- /9j/4AAQSkZJRgAB	8:16 am
me	DSC_0111.JPG	- /9j/4E+RXhpZ(	3:16 am
me	hello.txt	- aGVsbG8K	12:10 am
me	oldtest.txt	- VGhpcyBpcyBhIHRl	10:37 pm
me	lipsum2.txt	- aGVsbG8KCm15IG	10:37 pm

0.03 GB (0%) of 15 GB used
Manage
Terms - Privacy
Last account activity: 2 hours ago
Details

emfs

Files

oldtest.txt (22 B) (Download)

DSC_0111.JPG (781 KB) (Download)

hello.txt (6 B) (Download)

Baker.jpg (137 KB) (Download)

lipsum2.txt (2 KB) (Download)

1-s2.0-S0167404802005163-main.pdf (247 KB) (Download)

emfs

emfs
bc2015emfs@gmail.com

EMFS is an email based filesystem. Utilizing plain old SMTP and IMAP, EMFS can mount any email account (from any provider) as a fully functional filesystem capable of storing, indexing, and retrieving files of arbitrary size.

Figure 1: The earliest *EMFS*, showing email, filesystem, and web views.

A full implementation corresponding more closely to the system described here is in progress, but has not been released yet. When released, it will be available at `https://github.com/emfs-redux`.

References

[1] Apple Inc. *Apple Media Advisory - Update to Celebrity Photo Investigation*. 2014. URL: `https://www.apple.com/pr/library/2014/09/02Apple-Media-Advisory.html` (visited on 01/10/2016).

[2] Graham Cluley. *Dropbox Users Leak Tax Returns, Mortgage Applications and More*. 2014. URL: `https://www.grahamcluley.com/2014/05/dropbox-box-leak/` (visited on 01/10/2016).

[3] *Syncthing*. 2016. URL: `https://syncthing.net/` (visited on 01/10/2016).

[4] *ownCloud*. 2016. URL: `https://owncloud.org/` (visited on 01/10/2016).

[5] Jonathan Postel. *Simple Mail Transport Protocol*. RFC 821. 1982, pp. 1–72. URL: `https://tools.ietf.org/pdf/rfc821`.

[6] M Crispin. *Internet Message Access Protocol - Version 4rev1*. RFC 3501. 2003, pp. 1–108. URL: `https://tools.ietf.org/pdf/rfc3501`.

[7] Brian Kantor and Phil Lapsley. *Network News Transfer Protocol*. RFC 977. 1986, pp. 1–27. URL: `https://tools.ietf.org/pdf/rfc977`.

[8] C Feather. *Network News Transfer Protocol*. RFC 3977. 2006, pp. 1–125. URL: `https://tools.ietf.org/pdf/rfc3977`.

[9] Declan McCullagh. *N.Y. Attorney General Forces ISPs to Curb Usenet Access*. 2008. URL: `http://www.cnet.com/news/n-y-attorney-general-forces-isps-to-curb-usenet-access/` (visited on 01/11/2016).

[10] Declan McCullagh. *Verizon Offers Details of Usenet Deletion: alt.* Groups, Others Gone*. 2008. URL: `http://www.cnet.com/news/verizon-offers-details-of-usenet-deletion-alt-groups-others-gone/` (visited on 01/11/2016).

[11] Juhoon Kim et al. "Today's Usenet Usage: NNTP Traffic Characterization". In: *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*. 2010, pp. 1–6. DOI: `10.1109/INFCOMW.2010.5466665`.

[12] *Backup Your Data on Usenet for Free (uBackup)*. 2014. URL: `http://www.ngprovider.com/ubackup.php` (visited on 01/10/2016).

[13] Claus Färber. *yEnc Considered Harmful*. 2002. URL: `http://www.faerber.muc.de/temp/20020304-yenc-harmful.html` (visited on 01/10/2016).

[14] Jeremy Nixon. *Why yEnc is bad for Usenet*. 2002. URL: http://www.exit109.com/~jeremy/news/yenc.html (visited on 01/10/2016).

[15] C Feather. *SMTP Service Extensions for Transmission of Large and Binary MIME Messages*. RFC 3030. 2000, pp. 1–12. URL: https://tools.ietf.org/pdf/rfc3030.

[16] P Hoffman. *SMTP Service Extension for Secure SMTP over TLS*. RFC 2487. 1999, pp. 1–8. URL: https://tools.ietf.org/pdf/rfc2487.

[17] C Newman. *Using TLS with IMAP, POP3 and ACAP*. RFC 2595. 1999, pp. 1–15. URL: https://tools.ietf.org/pdf/rfc2595.

[18] J Callas et al. *OpenPGP Message Format*. RFC 4880. 2007, pp. 1–90. URL: https://tools.ietf.org/pdf/rfc4880.

[19] Idilio Drago et al. "Inside Dropbox: Understanding Personal Cloud Storage Services". In: *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*. IMC '12. Boston, Massachusetts, USA: ACM, 2012, pp. 481–494. ISBN: 978-1-4503-1705-4. DOI: 10.1145/2398776.2398827. URL: http://doi.acm.org.proxy-um.researchport.umd.edu/10.1145/2398776.2398827.

[20] Google Inc. *Maximum Attachment Size*. 2016. URL: https://support.google.com/mail/answer/6584?topic=1517 (visited on 01/12/2016).

[21] Yahoo! Inc. *Message Size Limit*. URL: https://help.yahoo.com/kb/SLN5673.html (visited on 01/12/2016).

[22] Microsoft Inc. *"Attachment Size Exceeds the Allowable Limit" Error When You Add a Large Attachment to an Email Message in Outlook 2010*. 2013. URL: https://support.microsoft.com/en-us/kb/2222370 (visited on 01/12/2016).

[23] Google Inc. *Gmail now has over 900M users! Thanks for helping us get there*. 2015. URL: https://plus.google.com/+Gmail/posts/AjktcDswdKh (visited on 01/15/2016).

[24] Susan Kuchinskas. *Endless Gmail Storage*. 2005. URL: http://www.internetnews.com/xSP/article.php/3494491 (visited on 01/15/2016).

[25] Google Inc. *Free Storage and Email from Google*. 2016. URL: https://www.google.com/intl/en_us/mail/help/about.html (visited on 01/15/2016).

[26] Microsoft Inc. *Microsoft by the Numbers*. 2014. URL: `http://news.microsoft.com/bythenumbers/ms_numbers.pdf` (visited on 01/15/2016).

[27] Microsoft Inc. *Storage Limits in Outlook.com*. URL: `http://windows.microsoft.com/en-CA/windows/outlook/email-storage-limits` (visited on 01/15/2016).

[28] Dan Frommer. *Google is Stealing Away Microsoft's Future Corporate Customers*. 2014. URL: `http://qz.com/243321/google-is-stealing-away-microsofts-future-corporate-customers/` (visited on 01/16/2016).