Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Bc. Jan Kundrát

IMAP extension for mobile devices

Department of Software Engineering

Supervisor of the master thesis:	RNDr. Ing. Jiří Peterka
Study programme:	Informatics
Specialization:	Software Engineering

Prague 2012

Acknowledgement

I would like to thank my supervisor, RNDr. Ing. Jiří Peterka, for his valuable advices and numerous consultations.

Many experts on various IMAP-related mailing lists, especially Timo Sirainen, Bron Gondwana, Arnt Gulbrandsen, Mark Crispin, Alexey Melnikov, Dave Cridland and Adrien de Croy, have shared explanations about corner cases concerning various protocol features and provided feedback on my Internet-Draft proposals — thanks for that.

Portions of my work on Trojitá were sponsored by KWest GmbH. and OpenMFG LLC, dba xTuple. Thanks to Sebastian Wendt and John Rogelstad for making this collaboration possible.

Adam Kudrna kindly corrected many mistakes in this text; the remaining errors, however, are mine.

Finally, special thanks belong to Ms. Julie Růžičková for her willingness to listen whenever I had something to discuss, patient support when I was debugging intricate bugs and tremendous amount of empathy throughout the time, as well as to my parents for supporting my studies.

Prague, July 27, 2012 Jan Kundrát

Název práce: IMAP extension for mobile devices
Autor: Jan Kundrát
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Ing. Jiří Peterka

Abstrakt:

Masové rozšíření chytrých telefonů, ke kterému došlo v posledních letech, s sebou přineslo i zvýšený zájem o mobilní přístup k elektronické poště. Protokol IMAP prošel od svého vzniku mnoha úpravami; objevila se rozšíření přidávající nové funkce, ale i modifikace optimalizující komunikaci přes potenciálně nespolehlivou síť. Tato práce si klade za cíl provést analýzu stávajících rozšíření protokolu IMAP z pohledu jejich použitelnosti a přínosu využití v mobilních zařízeních. Obsahuje tři nově vytvořené návrhy rozšíření, z nichž každé obohacuje protokol IMAP z jiného směru. Součástí práce je též popis, jak byla tato rozšíření zahrnuta do programu Trojitá, autorova mobilního e-mailového programu šířeného jako svobodný software.

Klíčová slova: IMAP, e-mail, rozšíření, mobilní přístup, optimalizace, datový tok, Lemonade, Trojitá

Title: IMAP extension for mobile devices
Author: Jan Kundrát
Department: Department of Software Engineering
Supervisor: RNDr. Ing. Jiří Peterka

Abstract:

With the mass availability of smartphones, mobile access to e-mail is gaining importance. Over the years, the IMAP protocol has been extended with many features ranging from extensions adding new functionality to those improving efficiency over an unreliable network. This thesis evaluates the available extensions based on their suitability for use in the context of a mobile client. Three new extensions have been developed, each improving the protocol in a distinct way. The thesis also discusses how most of these extensions were implemented in Trojitá, the author's free software open source IMAP e-mail client.

Keywords: IMAP, e-mail, extensions, mobile access, optimisation, data traffic, Lemonade, Trojitá

Contents

Chapter 1

Introduction

I also believe the only reason we still use email is that it's impossible or very difficult to replace, kind of like Facebook in a way.

(Heidar Bernhardsson [1])

The boom of mobile computing in the last few years has changed the way how people work with their e-mail. After a short period where it appeared that the webmail was the future and days of standard protocols were numbered, suddenly people realized that in fact, there *is* a merit in having a standardized protocol for e-mail access.

Contemporary smartphones started to smear down the border between how a text message, an instant-messaging (IM) communication and a traditional e-mail work. Suddenly, it is common to have a single "messaging" application aggregating communication history gathered from many distinct channels.

Even though some people prefer not to use these integrated facilities, their mere existence and certain demand from the end-users present an unique opportunity for standardization — it is easier to develop, test and deploy *one* particular data provider suitable for use with many different services from multitude of vendors than having to deploy a custom implementation for each social network which happens to be popular this year and on the continent the device manufacturer decided to target. Engineers working for cell phone and tablet vendors who are taking place in the IETF standardization process are a clear evidence that the market recognizes this potential and that nobody wants to loose.

In this thesis, I would like to explore the IMAP protocol [2] and its rich extension family, evaluating their features by a prism of a *mobile client* — a device which might have a decent amount of CPU and memory resources, as common with today's smartphones and modern tablets, but whose network connection is prone to frequent interruptions and might be payed based on the amount of transferred data, and whose battery would get extinguished in a few hours if certain precautions are not taken. It turns out that although many of the IMAP extensions are extremely useful for increased efficiency of the client's operation, there are still quite a few opportunities for improvement, or outright deficiencies to correct.

1.1 Structure of the Thesis

The next chapter (p. 9) of this thesis provides a brief introduction to the baseline IMAP protocol, its strengths and weaknesses and the general mode of operation. The third chapter (p. 16) analyzes the existing IMAP extensions based on the "layer" on which they add features to IMAP and on how they could be useful for the clients. Their usefulness is illustrated through my experience in development of *Trojitá*, an open source mobile IMAP e-mail client which I started and have been maintaining for the last six years. In chapter four (p. 37), I have selected three completely different areas in which the state of IMAP, as of 2012, can still be built upon. My improvements are delivered in the format of the so-called *Internet Drafts*, materials directly used by the IETF as the source of the RFC documents, the specifications which drive innovation on the Internet. All of the extensions which I propose were presented for expert review through the relevant communication channels and are on track to become the Internet-Drafts and, hopefully, RFCs.

After the theoretical section, chapter five (p. 46) evaluates the status of the extension support among a selection of existing mobile clients available on the market and on real devices. Chapter six serves as a short introduction for programmers into how Trojitá, the mobile e-mail client developed for this thesis, operates. The work is concluded and a discussion of the work planned in future is available from chapter seven (p. 62).

All of the extensions which I am proposing are available in their traditional, formal format in Appendix A (p. 64).

Finally, as Trojitá is a free software, open-source project, Appendix B (p. 94) acknowledges the work of other developers who have contributed to Trojitá over the years and lists the open source libraries required for its operation. It also mentions two companies which decided to build their e-mail products on top of Trojitá. The instructions on how to build Trojitá from source as well as the structure of the accompanying CD are in Appendix C (p. 97).

Chapter 2

IMAP Protocol Essentials

This chapter provides a gentle introduction to peculiarities of the IMAP protocol and presents an analysis of how its users can benefit from the unique protocol features.

2.1 IMAP

The IMAP protocol, as defined by RFC 3501 [2], is an internet protocol suitable for managing e-mail folders and individual messages stored on a remote mail server. In contrast to the older POP3 protocol [3], IMAP is actually intended to serve as an *access* protocol. Where a POP3 client would happily download a full message from the mail server, store it into a local mailbox and perform all further processing locally, the IMAP mode of operation is much more complicated. These complications, however, bring a whole slew of new features and interesting applications along.

For one thing, IMAP presents a single authoritative place storing messages — that feature alone is a must in today's world where everyone expects to be able to access mail from their cell phones. Furthermore, given that all messages are located on a single place, it is possible to perform efficient server-side operations like searching or sorting over the whole mail store. IMAP also makes it possible to access individual message parts like attachments separately, eliminating the need to download a huge message before reading a short accompanying textual information. Finally, advanced servers can recognize clients with limited resources and only present a subset of messages to them.

At the same time, IMAP is an old protocol burdened with many compatibility warts. Its designers were struggling with people objecting to novel ideas due to legacy code in their mail implementations. Over the years, though, various protocol extensions appeared. Some of them are extremely useful for contemporary clients, yet cannot be relied upon as there is no general agreement on what extensions are really crucial, and hence available on most IMAP servers.

The rest of this chapter provides a quick overview of the basic IMAP concepts and how they relate to the usual client's workflow. A detailed introduction to the basic IMAP concepts can be found in my bachelor thesis on this topic [4, p. 9 - 19].

2.1.1 Basic Features

An IMAP server exports a set of *mailboxes*, folders which contain individual messages (and further mailboxes, if the server allows that). Each message can be identified either by its *sequence number*, an order in which it appears in mailbox, or by its *UID*. Sequence numbers are by definition very volatile (deleting the first message in a mailbox changes sequence numbers of all subsequent messages, for example) while the

UIDs provide better chances of persistence across reconnects. [1] When the UIDs have to be invalidated for some reason, a per-mailbox integer property `UIDVALIDITY` is incremented to signal all clients that previously used UIDs are no longer valid.

2.1.2 Cache Filing Protocol

As Mark Crispin, the principal author of the IMAP standard, has to say [5] [6], IMAP is a *cache filing protocol*. That means that whatever the server thinks about a mailbox state is *the truth*, and any state stored on the clients can be invalidated by the server at any time. This critical design choice has impact on all further operations. IMAP clients which do not anticipate such a behavior [2] are bound to operate in an inefficient manner or fail in unexpected scenarios.

The first issue which typically comes up on the `imap-protocol` mailing list is treating UIDs as a persistent identifier of some kind. In fact, IMAP guarantees that a triple of (mailbox name, `UIDVALIDITY`, `UID`) will not refer to *any other* message at any time, but there's no guarantee that the very same message, quite possibly in the same mailbox, will not get another UID in future. [3] That said, on reasonable server implementations, the UIDs should not get invalidated too often under normal circumstances. Given the IMAP protocol doesn't offer anything else, they are widely used (along with the `UIDVALIDITY` and when limited to the scope of a single mailbox) as a semi-persistent identification of a message.

UIDs are assigned to the messages in a strictly monotonic sense, i.e. if message A has a sequence number seq_A and message B has sequence number seq_B such as $seq_A < seq_B$, it is true that $UID_A < UID_B$. UID numbers are also guaranteed to never be reused in the same mailbox, unless the `UIDVALIDITY` changes.

Due to the facts described above, virtually any IMAP client which maintains a persistent cache of the downloaded data uses UIDs to assign the cached data to individual messages. Such an approach leads to a need to maintain a mapping between the sequence numbers and the UID numbers of all messages in the mailbox — upon reconnect, clients have to recognize whether any messages previously available in the mailbox disappeared, and if they did, the clients should remove the cached data for these messages. [4] This is in a strong contrast to the usual POP3 mode of operation where the clients are expected to prune their cache only based on their local policy, perhaps moving older messages to a designated archive, but definitely not discarding the retrieved data as soon as the server doesn't present the message any longer.

Furthermore, even during an established session the IMAP server informs about messages being permanently deleted through the `EXPUNGED` response which contains sequence number only. Given that the cache is usually addressed by UID, a caching client shall maintain full UID mapping at any time.

[1]It shall be noted that IMAP does *not* guarantee UIDs to be persistent at all. The reason behind this decision was to allow IMAP to publish messages from obsolete mail stores which could not have been extended to support UIDs at all. Even today, UID changes have to be expected when signalled through `UIDVALIDITY`.

[2]Such clients are usually called "POP3 clients converted to speak IMAP" on various IMAP-related mailing lists. [7] [8]

[3]People have been trying to solve this issue for quite some time, but no standardized solution is ready yet. The recent iterations of these proposals concentrate on providing a cryptographic hash of a message body, but is far from clear whether doing so would get any traction. Furthermore, the hashes are typically too long to serve as the only identifier of a message, so UIDs will definitely be around in future.

[4]The reader shall be reminded that IMAP is a *cache filing protocol*, i.e. the server is always right about what messages "are" in a mailbox and what messages are gone.

2.2 Mailbox Synchronization

When an IMAP client opens a mailbox, the server provides it with a few data points about the state of the mail store. Among these data, there's a number representing the total amount of messages in the mailbox through the `EXISTS` response, the current `UIDVALIDITY` value and, finally, the `UIDNEXT` which represents the lowest UID which the next arriving message could possibly get assigned. Please note that the `UIDNEXT` is merely a lower bound of the future UID; there is no guarantee that a message with such UID would ever exist in the mailbox in future.

Having obtained these three values, the client can perform a few optimizations before it proceeds to fetch an updated UID mapping from the IMAP server:

- If the `UIDVALIDITY` has changed, the client is obliged to completely purge any data which it might have accumulated in its local persistent cache. This is a hard requirement allowing the server to inform the client that no state whatsoever can be reused from the previous connections. In the real world, this situation shall be only reached under exceptional circumstances (like when migrating to a completely different server implementation, or after having to restore the data after an inadvertent damage caused by a reckless system administrator. [5]

- If `UIDNEXT` is not available, the client has to resort to asking for the whole UID mapping from scratch.

- If the `UIDNEXT` has decreased, the IMAP server exhibits a bug. This situation is explicitly forbidden by the IMAP standard. Trojitá will, nevertheless, try to work in this scenario by purging its cache and continuing as if no state was cached locally.

- If the `UIDNEXT` has not changed since the last time the client has opened the mailbox, the IMAP protocol says that no messages could have been delivered to the mailbox at all.

  – If the `EXISTS` remains constant as well, it is clear that no deletions have taken place. This means that the cached sequence → UID mapping from the last time is directly usable, and the UID syncing phase of the mailbox synchronization is concluded.

  – Otherwise, if the `EXISTS` has grown, the client is talking to a non-compliant IMAP server which failed to adjust either `UIDNEXT` or `UIDVALIDITY`, and cannot assume anything about the server's behavior. Trojitá will gracefully degrade to a complete UID mapping resynchronization.

  – If the `EXISTS` has decreased, one can be sure that some messages have been deleted. In this situation, the client has two possible options on how to proceed:

    ∗ One can try to perform a binary search in the list of messages to find the first deleted message and ask for UIDs of all messages at the subsequent positions. This is a heuristics which relies on an observation that it is more likely for users working with big mailboxes to delete

[5]The IMAP standard nevertheless allows servers to increment the `UIDVALIDITY` upon each reconnect to accommodate server implementations which are unable to assign persistent UIDs at all. It shall be noted that although such servers are *compliant* with the IMAP specification, they offer severely limited user experience and little room for further optimization — the clients cannot reuse any data from previous connections, so the overall efficiency is similar to accessing e-mail through the POP3 protocol.

messages at the end of the mailbox. However, each step in this incremental search requires a complete round trip to the IMAP server over a network; with a mailbox with tens of thousands of messages, this could lead to 17 round trips. Given that real-world cellular networks like the GPRS/EDGE infrastructure, unfortunately still common in the Czech Republic, exhibit the RTT latencies which can often be larger than one second [9], such an approach to incremental synchronization of the UID mapping will have severe impact on the total synchronization time.

    * Another way is to give up on possible bandwidth reduction possibility and to fetch the complete UID mapping.

- If the `UIDNEXT` has grown, some messages might have arrived into the mailbox. There's no guarantee that any of them are still present, though, so the clients could use another set of heuristics:
  - If the increase in `EXISTS` is exactly the same as the growth of the `UIDNEXT`, all of the new arrivals are still present in the mailbox and no message have been expunged since the last time. The client can ask only for UIDs of the new arrivals.
  - In any other case, the situation is very similar to a changed `EXISTS` with constant `UIDNEXT` and the same possible optimization about the binary search might apply. Alternatively, clients could fetch a complete UID mapping.

If the decisions described above suggest that at least a part of the UID mapping shall be updated, an IMAP client can — in absence of the optional extensions — use one of the following ways to update the map. The first one is through the generic `FETCH` command:

```
C: y1 UID FETCH 1:* (UID)
S: * 1 FETCH (UID 123)
S: * 2 FETCH (UID 125)
S: * 4 FETCH (UID 127)
S: * 3 FETCH (UID 126)
S: y1 OK Fetched
```

This command simply requests the `FETCH` response containing UID for each and every message in the mailbox. The sample results show that the received data are in no particular order and demonstrate that the UID range is not necessarily continuous. If the heuristics shows that there is just a subset of messages with unknown UIDs, the sequence range (the `"1:*"` string in the example above) shall be changed to only refer to the relevant subset, like the `"last_uidnext:*"`. It is also possible to request `FLAGS` (which will be described later on) at this point.

Alternatively, the `UID SEARCH` command can be used as follows:

```
C: y1 UID SEARCH UID ALL
S: * SEARCH 123 125 127 126
S: y1 OK search completed
```

As one can see, the `SEARCH` response is much more compact. In practice, the bandwidth saving is slightly lower as the UID discovery and `FLAGS` synchronization can be merged into a single `FETCH` command, but the overhead is still at least four bytes for each message in the mailbox, [6] which leads to at least 200 kB of useless data on a mailbox with fifty thousands of messages.

2.2.1 Message Flags

I have mentioned message flags when describing the mailbox synchronization. These flags allow the system and the mail user to attach a certain state to the messages — information like whether the message has been read or replied to is tracked at this level. Further applications include user-level arbitrary tagging of messages with flags like "important" or "todo".

Strictly speaking, asking for message flags of all messages in a mailbox is not necessary, provided the program is capable of lazy-loading — flags could, for example, only be fetched for those messages which are immediately visible on screen (probably with some intelligent preload of items which will likely get to the viewport in near future), avoiding a potentially expensive operation. On the other hand, contemporary user agents typically have to display an aggregated statistics like "X unread messages, Y total" to the user. IMAP certainly has methods for delivering such statistics, however, the baseline specification's only two ways of conveying that information are through the `STATUS` command or via an explicit `SEARCH`. In practice, this design leads to a pressing need to load *all* flags for all messages at the start of the session.

The problem with the `STATUS` command is that it is unfortunately forbidden from being used on an actively selected mailbox [2, p. 43]. That makes this command usable for an initial estimate, but prevents further updates — consider that an IMAP client has opened a big mailbox and scrolled to the end of the message listing. Suddenly, an `* EXPUNGE 3` arrives, informing the client that the third message in the mailbox is now gone. Because the flags of those "older" messages haven't been loaded in this scenario, the client has no way of knowing whether the number of unread messages shall be updated. At this point, the client has no choice but to explicitly ask for message flags of all messages or conduct a special `SEARCH`. The `SEARCH` command looking for unread messages (or for any set of messages tagged with a certain flag, for that matter) can surely be constructed, even the baseline IMAP4rev1 provides a way of requesting that information. However, each `SEARCH` only provides the client with an information about one kind of a particular flag. It is not an unreasonable idea to design a client with further development in mind, most notably it might make a lot of sense not to special-case the `\UnSeen` message flag — after all, certain applications will benefit from having access to all messages matching the `$SubmitPending` flag or those which were marked as a "Draft" by the user, for example. Unfortunately, statistics about these user-defined flags cannot be determined via the `STATUS` command and have to be discovered explicitly, either through a lot of separate `SEARCH` commands, one for each "interesting" flag, or via an explicit synchronization through the `FETCH (FLAGS)` command.

In short, deferring the flag synchronization certainly has some merit, but at the same time, special-casing the `\UnSeen` flag for unread messages is not a viable long-term solution. Given that extensions designed for speeding up the flags resynchronization exist, Trojitá will always ask for a full flags mapping when synchronizing through the baseline IMAP profile.

[6] If the `FLAGS` are fetched as well, the real overhead is just the `"UID<space>"` string — the number and its trailing space is present in the `SEARCH` response as well and the overhead of the `FETCH` response format is required for the updated flags anyway.

2.2.2 Immutable Data

In the previous sections, I have spoken about data which have to be resynchronized during each reconnect to the mailbox, be it message flags or the UIDs. Other data available through IMAP are, however, immutable by nature. Examples of these data are message headers or the individual body parts.

IMAP is pretty unique in allowing its implementors to dissect a MIME message [10] into individual body parts. In practice, this is a very useful feature — clients can read a short textual body of a message before deciding whether to download a big binary attachment etc. On the other hand, it requires servers to include a full MIME parser. Some of them, notably the Google's GImap, have been struggling with this requirement for many years [11].

IMAP defines a data structure called `ENVELOPE` containing some of the most interesting headers from the RFC 2822 [12] message. Among them, the `Subject`, `Date`, `From`, `Sender`, `Reply-To`, `To`, `Cc`, `Bcc`, `In-Reply-To` and `Message-Id` are included. Unfortunately, the `References` header is missing. [7] Even with this omission, the `ENVELOPE` is useful for clients which do not necessarily have to include RFC 2822-style header parsing code. However, this usefulness is unfortunately further limited by not including an RFC 2047 [13] decoder, so non-ASCII data in fields like senders' human readable names or in the subject field have to be decoded by the clients.

2.3 Protocol Design

The baseline version of IMAP, as defined in RFC 3501 [2], contains a few features which limit its performance by a fair amount. One example of these features are IMAP's *synchronizing literals*.

Before the client is allowed to send a big amount of data to the server, it has to ask for its explicit permission via a continuation request. While such an idea is good on paper (and is probably intended to *save bandwidth* by allowing the server to refuse huge uploads before the client sends them), in reality this leads to rather slow operation because each transmission requires a full roundtrip over the network. Fortunately, extensions like the `LITERAL+` (see section 3.1.1 on page 16) have eliminated this bottleneck.

Another manifestation of a situation which could potentially use an improvement is the protocol's requirement on clients to accept any response at any time, [8] which is not applied consistently — the same RFC also mandates that the servers cannot send `EXPUNGE` when no command is in progress [2, p. 72]. This particular wording certainly has merits (it encourages client implementors to *really* accept anything at any time) and is required for proper synchronization — if `EXPUNGE`s were allowed when no command was in progress and a client issued a `STORE` command referencing a message through its sequence number, that action could affect a completely different message. This design is probably required due to the old decision to support addressing through both sequence numbers and UIDs, but has a side effect of requiring constant polling for mailbox updates. Again, extensions have emerged (see section 3.1.4 on page 18) which try to eliminate this drawback through a special mode.

[7] The `References` header is useful when the client wants to be as compatible as possible with the other agents that deal with message threading. Strictly speaking, the `Message-Id` and `In-Reply-To` headers are sufficient for some forms of threading, but MUAs should strive to be "good citizens" and support the `References` header as well.

[8] "`The client MUST be prepared to accept any response at all times.`" [2, p. 61]

2.3.1 Additional Server-Side Features

Having all messages available without much effort (or, certainly with much less effort than a client), servers are in a unique position to make certain operations smoother, faster and more efficient than performing on the client side.

The baseline IMAP specification contains provisions for server-side searching. Features notably missing, however, are server-side sorting and conveying information about message threading [9] which are available through optional extensions.

Certain scenarios (like having a cell phone with severely limited resources) could benefit from server-side content conversion, similar to how the Opera Mobile browser converts images to low-resolution versions for display on the phone's screen. An extension for just that exists, but its support is rather scarce among the IMAP servers — in fact, the author of this thesis was not able to find a single reasonably-deployed server which would offer such a feature.

At the same time, the overall design of the IMAP protocol is rather promising; it allows executing commands in parallel through pipelining and even the most basic profile provides enough features to implement a reasonably efficient client which can maintain its state over reconnects. It is therefore reasonable to start with the existing state and try to build upon its solid foundation, improving the whole experience in the process.

[9]Message *threading* refers to a mechanism which allows graphical user agents to present related messages together. Recently re-branded as "conversations", threading is in fact a pretty old idea which builds on the `Message-Id`, `References` and `In-Reply-To` headers, or, in its crudest form, just on similarity of message subjects. Threading presents the human with a tree of messages where each immediate child of a particular node represents a reply to the parent message.

Chapter 3

IMAP Extensions

It might be concluded from the brief analysis presented in the previous chapter that certain features of the IMAP protocol are rather limiting in its real-world deployment. Fortunately, IMAP was designed to include support for extensions which allow rather substantial changes to its mode of operation. Throughout this chapter, I provide a detailed analysis of opportunities where the baseline IMAP protocol leaves something to be desired. Many such gaps have been addressed by various extensions over the last twenty years; these extensions are thoroughly explained and evaluated on their merits. Information about their support in Trojitá is also included.

3.1 Optimizing the Protocol

Before dwelling into more advanced topics like improving the synchronization performance or adding new features, let's have a look at the basic layer of the IMAP protocol and investigate how these affect performance.

3.1.1 The LITERAL+ Extension

One of the lowest-hanging optimization fruit to cater are IMAP's synchronizing literals. In the basic IMAP, before a client proceeds with tasks involving upload of binary data (or any data over a certain size, for that matter), it has to ask for an explicit server's approval based on the length of the data in question. As it has been shown previously, this confirmation imposes a full round trip over the network, inducing latency and destroying any potential pipelining improvements.

The LITERAL+ extension (RFC 2088 [14]) simply lifts the requirement of having to wait for the server's continuation requests by a subtle change of the syntax. Adding an overhead of just one byte, the latency is completely eliminated and communication gets rapidly streamlined. One can go as far as to say that the code paths for dealing with LITERAL+ data are actually simpler than having to deal with the old-fashioned synchronizing literals. Consider the following example:

```
S: * OK
C: A001 LOGIN {11}
# The client has to wait for server's response before proceeding any further
S: + go ahead
C: FRED FOOBAR {7}
# A second round-trip wait occurs here
S: + go ahead
C: fat man
S: A001 OK LOGIN completed
```

When using the LITERAL+ syntax, the whole interaction happens without having to wait for the server:

```
S: * OK
C: A001 LOGIN {11+}
C: FRED FOOBAR {7+}
C: fat man
S: A001 OK LOGIN completed
```

> *Trojitá includes full support for the LITERAL+ extension — when it detects the* `LITERAL+` *capability, it will immediately switch to using non-synchronizing literals for increased performance.*

3.1.2 Data Compression

IMAP is a textual, line-based protocol. As such, it presents extremely good opportunities for compression — using the tried DEFLATE algorithm [15], the basic IMAP chatter can be easily compressed to 25 - 40 % of its original size [16, p. 4]. RFC 4978 [16] provides mechanism for exactly this functionality through the `COMPRESS=DEFLATE` capability.

> *Trojitá ships with full support for this extension through the permissively licensed* `zlib` *library. Unfortunately, the Qt's* `QSslSocket` *currently doesn't provide a way to reliably tell whether the SSL connection is already employing compression. When combined with IMAP servers hidden behind SSL accelerators or load balancers (i.e. in situations where the server does not have a clear idea whether the session is already compressed either), this has a risk of needlessly trying to compress data twice. This is a limitation in system libraries which cannot be overcome without resorting to patching system components or conducting non-portable hacks.*

3.1.3 Improving Security through Cryptography

RFC 2595 [17] deals with best practices for establishing SSL/TLS connections to the IMAP server.

> *Trojitá follows these recommendations, most notably it tries to establish a secure channel over* `STARTTLS` *command even without an explicit action on the user's side, should the server be configured to advertise itself as not accepting logins over insecure connections through the* `LOGINDISABLED` *capability. A manual override is available in situations where the SSL encryption is not available.*
>
> *In advent of the recent breaches of many well-known (and widely trusted) Certificate Authorities [18], Trojitá also comes with support for SSL key pinning [19]. The trust model presented to the user is similar to handling of SSH servers' public keys with OpenSSH — upon first connection, the user is always presented with a choice of whether to accept the certificate or not, along with a confirmation about whether the operating system and its policy considers the certificate as "trusted". No matter what the system-wide policy says, a changed public key is always considered a threat and the situation is presented to the user accordingly.*

3.1.4 The IDLE Mode

I have mentioned that even though the protocol requires clients to be ready to accept any responses at any time, in practice, servers are forbidden to send EXPUNGEs when no command is in progress. This requirement is necessary to prevent a dangerous resynchronization as the server cannot possibly know whether the client has started to issue an UID-less STORE command which references messages through their sequence numbers. Unfortunately, this directly translates to clients having to *poll* the server quite often if they care about updates concerning the deleted messages.

Any protocol which uses polling looks bad on paper — having to poll leads to increased latency and higher power usage because the equipment has to actively check for updates every now and then. In contrast, *push-based* updates allow the client to enter a low-power state where it merely waits to be woken up when a change occurs. Such a mode is exactly what the IDLE extension defined by RFC 2177 [20] adds to IMAP. It must, however, be said that real-world concerns related to firewall timeouts and especially the NAT traversal has limited the usefulness of the IDLE command somewhat, even to the extent where Mark Crispin, the original author of the IMAP protocol, claims that "I see no particular benefit to use of IDLE on a desktop machine" [21] — a view which is not shared by the wider community [22] [23], yet certainly worth a consideration.

The IDLE extension is basically a hack on top of the IMAP protocol which reverses a mantra of the basic IMAP specification [2, p. 72]:

> A command is not "in progress" until the complete command has been received; in particular, a command is not "in progress" during the negotiation of command continuation.

With IDLE, a typical interaction might look like this one:

```
C: A004 IDLE
S: * 2 EXPUNGE
S: * 3 EXISTS
S: + idling
...time passes; another client expunges message 3...
S: * 3 EXPUNGE
S: * 2 EXISTS
...time passes; new mail arrives...
S: * 3 EXISTS
C: DONE
S: A004 OK IDLE terminated
C: A005 FETCH 3 ALL
S: * 3 FETCH (...)
S: A005 OK FETCH completed
C: A006 IDLE
```

The whole effect of the IDLE command is therefore to indicate to the server that the client is *really* willing to listen for any updates to the mailbox state. Because of compatibility concerns with legacy mail stores, the IDLE extension still does *not* mandate the server to actually send updates about any changes as soon as they are conducted — indeed, a server which internally polls every fifteen minutes to check whether a message has arrived is fully compliant with the IDLE extension, albeit rather useless to user who might expect (and, one might add, rightly so) to be *instantly* notified about changes to the mailbox.

Trojitá includes full support for the IDLE extension and will enter that mode automatically shortly after a mailbox is selected. A simple heuristics is implemented which delays re-entering the IDLE command if it is likely that the connection will be reused for any other purpose in near future, further eliminating needless data transfers. Unfortunately, Trojitá is at the mercy of the IMAP server when it comes to superfluous data transfers, so it cannot prevent the "pings" sent even when the connection does not contain a gateway with overly short timeouts.

3.2 Improving Mailbox Synchronization

The previous section dealt with optimizing the overall IMAP protocol as a whole. At this stage, let's have a look at more specific issues which cannot be easily overcome through generic measures like data compression using off-the-shelf algorithms or updates to the basic protocol flows.

In the basic IMAP, neither the server nor the client are required to keep any persistent state. Clearly, it is beneficiary for a client to keep downloaded copies of the immutable mailbox/message data (consult section 2.2.2 on page 14 in its persistent cache for some time, should the device constraints allow such a storage. There is still quite a lot of other data which has to be validated while the mailbox is being resynchronized. Consider the following scenario where a mail user agent opens a mailbox with a thousand of messages which has witnessed expunges and new arrivals since the last time it was opened:

```
C: y1 SELECT foo
S: * 1000 EXISTS
S: * OK [UIDVALIDITY 12345] UIDs valid
S: * OK [UIDNEXT 2345] Next UID
S: y1 OK Selected
C: y2 UID SEARCH ALL

  The following response has been shortened for demonstration purposes. In
  practice, it will have to contain a thousand of numbers.

S: * SEARCH 2 4 5 6 7 8 9 10 ... 2290 2310 2311 2312 2333
S: y2 OK Search completed
C: y3 FETCH 1:1000 (FLAGS)
S: * 1 FETCH (FLAGS ())
S: * 2 FETCH (FLAGS (\\Seen))
S: * 3 FETCH (FLAGS (\\Recent \$Answered))

  996 additional FETCH responses were omitted from this example for brevity.

S: * 1000 FETCH (FLAGS (\\Seen))
S: y3 OK fetched
```

Let's identify two steps which substantially contribute to the transferred data:

- synchronizing the UIDs,
- updating flags.

The rest of this section takes a look at optimization opportunities at each of these stages. Please keep in mind that some basic optimization heuristics concerning the UID synchronization were discussed in section 2.2 on page 11 along with reasons on why these steps are necessary in clients willing to maintain an offline cache of immutable data.

3.2.1 The ESEARCH Extension

As seen in the protocol sample, the `SEARCH` response containing UIDs of all messages in a mailbox can be rather large. At the same time, chances are that at least some of the adjacent messages might have been assigned contiguous UIDs — this is certainly not a requirement per se, but quite a few IMAP servers internally *do* assign UIDs from a per-mailbox counter. Real-world, albeit anecdotal evidence [24] indicates that this scenario is very common, and therefore it might make sense to transmit the UIDs of all messages using the `sequence-set` [2, p. 89] syntax. The ESEARCH extension, as defined in RFC 4731 [25], allows exactly that:

```
S: * ESEARCH [TAG "y12"] UID 1,3:9,17:25,30:1000
```

At the time of the ESEARCH adoption, the imap-protocol mailing list witnessed a disagreement on how exactly the `sequence-set` shall be interpreted. Mark Crispin, the author of the original IMAP protocol (but not of the ESEARCH extension) implemented ESEARCH in a different manner. He chose to take an advantage of the RFC3501-style definition of UID sequences where the RFC mandates that servers shall treat non-existent UIDs given in sequence sets as if they weren't referenced from the command at all. For example, if the mailbox contained just UIDs 3, 5 and 10, a client using the `3:10` construct has to be interpreted as if it requested `sequence-set 3,5,10`. Doing so present certain optimization opportunities to the servers, for example when the client already knows the UID mapping and performs a server-side search for messages matching certain criteria *and* the result set accurately matches an adjacent range of messages, the server could take advantage of this adjacency a return a `sequence-set` in the form of `10:150`, even though the mailbox contains only a few UIDs from this range [26]. Furthermore, his another point is that the clients already have two other ways of obtaining the UID mapping, either through the `UID SEARCH ALL` command or via an explicit `UID FETCH 1:*`. Needless to say, such a reasoning fails to take into account potential bandwidth savings which can be rather substantial on "reasonable" mailboxes. In the end, the authors of the RFC 4731 disagreed with Crispin [27] [28].

> *The ESEARCH extension allows nice bandwidth savings, so Trojitá tries to use it if the server says that it is supported.*

In addition to that, format of the returned responses is changed so that it also includes the tag of the command which caused it, allowing much more aggressive pipelining — for example, clients are free to perform the UID discovery at the same time as running a user-initiated search. On the other hand, even in presence of ESEARCH, the UID mapping still has to be synchronized explicitly. This requirement is only lifted in the QRESYNC extension (section 3.2.3 on page 22). Before describing that, though, it is necessary to have a look at the CONDSTORE.

3.2.2 Avoiding Flags Resynchronization via CONDSTORE

Leaving the UID synchronization alone for a while, let's have a look at various ways of eliminating the need to ask for changed message flags. In this case, no extension trying to reduce the data overhead of the FETCH response was proposed, but the problem got attacked from another side.

The whole point of flags synchronization is to be able to pick up changes which have happened since the last time the mailbox was selected. If only the server was somehow able to assign a “serial number” to each change, clients could subsequently ask for all changes which have happened after a certain point. The CONDSTORE extension from RFC 4551 [29] works in this way.

CONDSTORE-capable servers share a concept of “modification sequence”, a MODSEQ. Each message in a mailbox is assigned an unsigned 64bit integer. Whenever message metadata (like its flags) change, the MODSEQ of that particular message gets increased. Each increment is also required to reach a value which is higher than MODSEQ of any other message in that mailbox. Similarly, a mailbox is assigned a HIGHESTMODSEQ, an unsigned 64bit integer which is interpreted as “no message has ever had a MODSEQ higher than this number” — of course subject to the usual UIDVALIDITY rules. [1]

When a CONDSTORE-capable client opens a mailbox which was previously synced (and if the server supports CONDSTORE as well, of course — keep in mind that the IMAP extensions are strictly voluntary in their nature), at first it synces the UID mapping as usual, possibly through the ESEARCH command discussed earlier. After that, the client can use an extended variant of the FETCH command to ask for flags of those messages whose *current* MODSEQ is higher than the HIGHESTMODSEQ which the client has remembered previously. The server will respond with regular FETCH responses for each affected message. In result, after this interaction is completed, the client is aware of all pending flag changes and is fully resynchronized again.

This is how a typical synchronization might look like:

```
C: y1 SELECT foo (CONDSTORE)
S: * 1000 EXISTS
S: * OK [UIDVALIDITY 12345] UIDs valid
S: * OK [UIDNEXT 2345] Next UID
S: * OK [HIGHESTMODSEQ 715194045007]
S: y1 OK Selected

  At this point, the client will obtain the UID mapping, likely through the
  UID SEARCH or its ESEARCH variant:

C: y2 UID SEARCH RETURN () ALL
S: * ESEARCH (TAG "y2") UID ALL 2,4:10,21:1008,2290,2310:2312,2333
S: y2 OK Search completed

  At this point the CONDSTORE extension can be finally utilized -- only changed
  flags will be transmitted:

C: y3 UID FETCH 1:1000 (FLAGS) (CHANGEDSINCE 613184045007)
S: * 997 FETCH (UID 2310 MODSEQ 715194045007 FLAGS (\\Seen \\Deleted))
S: * 1000 FETCH (UID 2333 MODSEQ 715194045005 FLAGS (\\Seen))
S: y3 OK Fetched
```

[1] As always, any change in UIDVALIDITY directly translates to a full cache flush and discarding any data previously remembered for the affected mailbox. This includes not only the immutable data of messages, but also the UIDs, message flags and — in this extension — the MODSEQ and HIGHESTMODSEQ values.

The algorithm is race-free — as every message has a separate `MODSEQ` counter, the delay between the `SELECT` and `FETCH` command doesn't lead to data loss; by the time the `FETCH` completes, the server guarantees that the client has received any pending updates since the last synchronization.

The CONDSTORE is an extremely valuable extension; its savings on big mailboxes are predictable and automatic — instead of having to transmit $O(n)$ responses where n is the number of *messages*, only $O(m)$ are required under QRESYNC with m being the number of *modifications*. This is an extension which, unfortunately, places a certain burden on the IMAP server which has to track the serial numbers of messages' meta-data; however, given the obvious reductions in bandwidth, many servers have already implemented it, most notably the Dovecot and Cyrus open source IMAP servers.

> *Trojitá includes full support for this extension, making use of it whenever it is available.*

3.2.3 Optimizing UID Synchronization with QRESYNC

The CONDSTORE extension discussed earlier has brought in the concept of a server-side state tracking and used that to allow bandwidth-efficient way of synchronizing flag changes. Given that a CONDSTORE-capable server already tracks certain state, it might be worthwhile to somehow extend this state to cover the deleted messages as well. As it turns out, such a mechanism is implemented in the QRESYNC extension which is defined by RFC 5162 [30].

The basic idea behind QRESYNC is that as long as the UID mapping was fresh at some point in past, it is only necessary to inform the client about which UIDs from that set are no longer there and push out the UIDs of newly arrived messages.

The QRESYNC extension modifies the `SELECT command` so that it includes a few more parameters. First of all, the updated version of this command includes a tuple of `(UIDNEXT, HIGHESTMODSEQ)` as known to the client. If the `UIDNEXT` did not change, the server will have a look at the `HIGHESTMODSEQ` value and in addition to essentially behaving as a CONDSTORE server, it also sends out a list of expunged messages. [2] A new response is defined for this purpose, the `VANISHED EARLIER`.

In format similar to the `ESEARCH` response, the `VANISHED EARLIER` contains a `sequence-set` of UIDs which the server believes that the client considers to be present in the mailbox. Not only are these UIDs transmitted in a compact syntax, thanks to the `sequence-set` format, but the response typically contains only such UIDs which were *just* removed. The actual wording (and therefore the implications of this extension) are slightly different — the server is free to inform the clients about any UIDs, as long as they aren't in the mailbox right now, at the time of the sync. This is motivated by the need to relieve the servers from having to maintain a list of expunged UIDs indefinitely, just in case a QRESYNC-enabled client reconnects after two years of inactivity. When such a situation happens, a server which cannot remember expunges going so far in history has no other option but to send a `VANISHED EARLIER` for *all* UIDs lower than the `UIDNEXT`, no matter if they *ever* were present in the mailbox. This fallback suggests that the QRESYNC extension could very well have a negative net effect overall, at least in certain pathological situations — essentially when the list of expunges grows so long that the server decides to prune some of its records.

In order to mitigate this issue, a few other options were added to QRESYNC. The first of them is a way of indicating to the server the range of UIDs about which the

[2] Technically, the expunges are sent out before the information about the updated flags, but that isn't the point here.

client actually cares. The idea here is that if the client only cached a subset of messages (for example those with UID higher than 50000), there isn't much point in informing the client about each and every UID which might had been in the mailbox before (like those 49,999 of UIDs lower than 50,000).

However, chances are that this optimization is not enough to overcome the danger of having to sync too many UIDs — and indeed, some user agents might want to preemptively load messages both from the beginning *and* from the end of the mailbox in an effort to optimize preloading. Such user agents would not be able to benefit from the "range of known UIDs" optimization.

Fortunately, the SELECT QRESYNC command includes provisions for passing another type of data around — it is also possible to provide a representative list of $(sequence, UID)$ pairs. Using technique similar to the proposed binary search when discovering UIDs, the client can decide to send along a UID from roughly middle of the mailbox as the first one, followed by another one located at circa 75 % of the mapping, next one from $\frac{7}{8}$ etc., halving the interval in each step. The concrete strategy to be pursuit is left to the client, as it is basically a policy decision. Using more fine-grained interval means that more data is sent along during each resynchronization without a direct merit (i.e. when the server *still* remembers the previous HIGHESTMODSEQ and can provide the client with relevant data), while on the other hand sending less UIDs leads to minor data savings during ordinary reconnects while causing potentially huge amounts of data to be transfered when the server is unable to use the — perhaps very old or otherwise stale — HIGHESTMODSEQ.

The presented version of Trojitá always uses halving of sequences, effectively transmitting $log_2(n)$ sequence-UID pairs:

```
Sequence knownSeq, knownUid;
int i = oldUidMap.size() / 2;
while (i < oldUidMap.size()) {
    // Message sequence number is one-based, our indexes are zero-based
    knownSeq.add(i + 1);
    knownUid.add(oldUidMap[i]);
    i += (oldUidMap.size() - i) / 2 + 1;
}
```

Our example synchronization (a thousand messages in a mailbox, unspecified changed since the last time) could therefore look like this — the client has fresh enough UID mapping up to UID 1008 (sequence number 995). The server does not remember deletions so far to the past, but the passed UID mapping fragment could nevertheless be used to optimize the delivered data:

```
C: SELECT foo (QRESYNC 12345 613184045007 (500,750,875,937,968,983,990,995,997
512,772,887,949,980,985,995,1002,1008,1111))
S: * 1000 EXISTS
S: * OK [UIDVALIDITY 12345] UIDs valid
S: * OK [UIDNEXT 2345] Next UID
S: * OK [HIGHESTMODSEQ 715194045007]
S: * VANISHED (EARLIER) 1009:2289,2291:2309,2313:2332,2343,2344
S: * 997 FETCH (UID 2310 MODSEQ 715194045007 FLAGS (\\Seen \\Deleted))
S: * 1000 FETCH (UID 2333 MODSEQ 715194045005 FLAGS (\\Seen))
S: y1 OK Selected
```

The received data contain enough information to reconstruct complete UID mapping even under these unfavorable conditions. The redundant UID information in the `FETCH` responses can be also used to make sure that both sides of the connection arrive to the same final state.

Trojitá will throw an error when it detects failure at any time.

To further reduce the amount of data transmitted during the IMAP session, the QRESYNC extension also introduces a second kind of the `VANISHED` response — the one without the `EARLIER` modifier. Serving as a substitute for the ordinary `EXPUNGE`, the `VANISHED`'s biggest advantage is that it can inform about multiple expunges in a single response. Somewhat ironically, this modification also relieves the clients of their need to maintain a complete UID-sequence mapping at all times — but only after providing a method of making this synchronization severely less painful in the first place. The whole matter is a bit more complicated by the wording of the RFC which is pretty clear on that the `VANISHED` responses *should* be sent instead of `EXPUNGE` — a language which, in RFC terms, means that the servers are supposed to do so, yet the clients are forbidden from relying on such behavior because under special circumstances, the servers might very well have a good reason to defer back to the `EXPUNGE` [31].

Unfortunately, the combination of offset-based `EXISTS` (which is a response used to inform the client about growth of the number of messages in a mailbox) with UID-based `VANISHED` can lead to races, a dangerous condition which I have identified [32]. The problem lies in QRESYNC's allowance for non-existent UIDs to be included in the `VANISHED` response. Consider the following scenario where the client is fully synced with a mailbox with just a single message bearing UID 5. The mailbox' `UIDNEXT` is 11:

```
S: * 3 EXISTS
C: x UID FETCH 11:* (FLAGS)
S: * VANISHED 12:20
```

The server is telling the client that any UIDs between 12 and 20 are gone. The problem is that the client cannot possibly know whether any message has got any of these particular UIDs, i.e. whether the messages #2 and #3 (the first and second arrival) fall into that range. Trojitá will immediately send out a request for UIDs of the new arrivals (that is the `UID FETCH` command in the previous example), but due to the timing issues, it is perfectly possible that these messages are "long" gone (and the appropriate `VANISHED` sent) by the time the server receives the `UID FETCH` command. There isn't much that a compliant IMAP client can do at this point besides issuing an explicit command for finding out whether any new messages have actually remained in the mailbox. This is a minor deficiency in the QRESYNC extension which could be easily avoided by replacing the `EXISTS` in manner similar to how `EXPUNGE` got replaced by `VANISHED`. The previous example would look like this one, eliminating any possibility of races:

```
S: * ARRIVED 12,33
C: x UID FETCH 11:* (FLAGS)
S: * VANISHED 12:20
```

The `ARRIVED` command is defined in a proposed draft in section 4.1 on page 37. A similar functionality could be achieved through the NOTIFY extension defined by

RFC 5465 [33], but supporting NOTIFY is a rather strong undertaking for an IMAP server and — as of July 2012 — support for that extension is still rather scarce. [3]

The QRESYNC extension also mandates an interesting mechanism for its activation. The `SELECT` ... `QRESYNC` command does not work when the IMAP client opens a mailbox for the first time — the client has no cached state, so it cannot construct any of the optional arguments to the `QRESYNC` select modifier. Furthermore, it cannot fabricate a proper `HIGHESTMODSEQ` in a safe way – using value too low would result in needlessly sending a huge number of `VANISHED EARLIER` responses, while using a too high value could confuse the server and treat the client as a buggy one. This is why a special `ENABLE QRESYNC` command (from the ENABLE extension, RFC 5161 [34]) is defined to be a *QRESYNC-enabling command*, activating any necessary `MODSEQ` tracking on the server side. This requirement might be non-obvious at first, for the `SELECT` ... `QRESYNC` on its own should be sufficient to inform the server that the client indeed wants to speak QRESYNC.

Unfortunately, there is also a certain murkiness about the `ENABLE` command — the errata #1365 for RFC 5162 [35] proposes to add an explicit note that "(A server MUST respond with a tagged BAD response if) (...) or the server has not positively responded to that command with "ENABLED QRESYNC", in the current connection", even though the RFC 5161 explicitly allows for aggressive pipelining of `ENABLE` and `SELECT`. [4] I have raised this issue on the `imap-protocol` mailing list and the consensus there was that it is indeed allowed not to wait for the server's `ENABLED` before issuing a `SELECT` ... `QRESYNC` [36]. I fully support such an outcome as it would be rather awkward to see a requirement for extra network round trips in contemporary IMAP extensions.

3.3 Fetching the Data

The IMAP protocol contains rather rich set of features aimed at downloading the message data in an efficient manner; clients can defer parsing of the MIME message parts [10] to the server and deal with the individual data separately. In spite of that, there are a few optimization opportunities which can drastically reduce the amount of data to be transfered.

3.3.1 The BINARY Extension

Historically, e-mail messages could only contain English text, for which a 7-bit character set and the US-ASCII encoding was adequate. However, with the advent of "multimedia", a steady pressure had emerged, leading to the MIME standard family. Using MIME, complex tree-like structures can be embedded in e-mail messages and transmitted over the Internet mail. However, at the time these were introduced, there was a real risk of not being able to transmit such complex messages over traditional communication channels which were often only 7-bit safe. Due to these backward compatibility concerns, a few standard method of converting arbitrary data to a textual form were conceived under the name of *Content-Transfer-Encoding.*

[3] None of the widely deployed open source IMAP servers supported `NOTIFY` at the time this thesis was written. The Dovecot IMAP server had an experimental branch with partial support which would be enough to serve as a replacement for `ARRIVED`, but it could not be discovered through the IMAP capabilities because `NOTIFY` is an all-or-nothing extension; it mandates full server support and doesn't include provisions for partial functionality which would have been enough in this case.

[4] "There are no limitations on pipelining ENABLE. For example, it is possible to send ENABLE and then immediately SELECT, or a LOGIN immediately followed by ENABLE. [34, p. 2]

The two most common encoding schemes are Quoted-Printable [10, p. 18] and Base64 [10, p. 23], the latter of which is especially suitable for converting arbitrary binary data to a 7-bit form. However, it is clear that mapping generic 8-bit data into 7-bit octets (and eliminating the appearance of certain "magic" characters in the process) cannot possibly work without inflating the total size of the transferred data. For the Base64 Content-Transfer-Encoding, the mapping converts 8-bit input (i.e. 256 values per octets) into a target alphabet of only 64 characters, imposing an overhead of roughly 33 % compared to the raw binary form. Whenever the MIME-encoded message is transmitted, the amount of transferred data is therefore roughly one-third higher than strictly required.

RFC 3516 [37] adds a feature to work around this limitation through the `BINARY` extension. When the server supports this feature, clients can delegate the Content-Transfer-Encoding processing to the server and receive the raw binary data. Perhaps surprisingly, the `BINARY` extension was not supported in Dovecot, one of the most popular IMAP servers, at the time this thesis was written.

Nevertheless, Trojitá includes full support for this extension and will automatically fetch data using the appropriate FETCH modifier in order to reduce the amount of data to send over the network.

3.3.2 Server-side Conversions via CONVERT

Certain devices might have limitations which the sender might not have expected when she was preparing the message. For example, a screen of a cell phone could have a very low resolution. Unless the user really wants to see the full details after zooming in eight times, it might make sense to reduce the resolution of that 22-megapixel 5760 × 3840 image produced by Canon 5D Mk. III to fit on a 480 × 800 pixels screen of a high-end smart phone from 2012. Even if the user actually wants to see the real image, it might be worthwhile to offer an access to a lower-resolution version for a quick preview. This server-side conversion is what the `CONVERT` extension from RFC 5259 [38] enables.

Unfortunately, it appears that there are actually *no* publicly available servers which offer support for server-side conversions and the most popular open source implementations have not expressed much interest when asked for their future plans.

Due to this induced inability to test this feature for interoperability, Trojitá doesn't support the `CONVERT` *extension at this point.*

3.3.3 Metadata Decoding

IMAP requires compliant servers to support MIME message parsing and RFC 2822 header decoding. One feature which is notably absent, though, is a support for server-side decoding of RFC 2047-formatted message headers and IMAP's `ENVELOPE` fields. This shortcoming is partially addressed by two RFCs — the already mentioned `CONVERT` extension mandates support for character set decoding and conversions of RFC 2822 message headers while an experimental RFC 5738 [39] adds an "UTF-8" mode which switches all `FETCH` commands to return the decoded Unicode data, including the IMAP's `ENVELOPE`. Sadly, support for this extension is similar to what has been already said for the `CONVERT` and one can also safely claim that the possible data savings are minuscule, if any. In addition, the UTF-8 extension changes quite a lot of assumptions from the traditional IMAP protocol, to the extent that one could very well propose yet another extension which would *just* enable access to the decoded `ENVELOPE` and RFC 2822 header data. No such extension exists at this point, though.

If the IMAP protocol was designed today, mandating a full-featured RFC 2047 decoder would be an obvious addition, but with the legacy of the protocol history, a requirement to implement a client-side decoder anyway and no available server support and the RFC being marked as experimental*, Trojitá does not try to use the* `SELECT ... UTF8` *parameter.*

3.4 Updating Mailboxes

Previous sections have introduced the existing extensions aimed at improving mailbox synchronization and data download. In this part, I will talk about how to optimize access to an already selected mailbox and how to get updated information about other mailboxes.

3.4.1 Sorting Messages

In a typical IMAP scenario, the server can access data for any message in a mailbox in a very cheap way. This is in a strong contrast to its clients which are typically connected over a network whose quality could leave much to be desired. It might therefore make sense to offload the data-intensive processing to the IMAP server and only send the results to the client.

RFC 5256 [40] adds support for server-side sorting. Using these features allows clients to request the server to sort a subset of messages using predefined sorting criteria like the message date, the time of its arrival, name of the sender, contents of the subject field etc. This feature set is subsequently augmented by RFC 5957 [41] which adds another sorting method which prefers the “real name” included in the e-mail addresses instead of using the raw `foo@example.org` format. The results of the `SORT` operation are transmitted in a format similar to the `SEARCH` response.

Trojitá includes full support for both of the mentioned RFCs.

This feature alone is a tremendous improvements over the traditional method where clients would have to download envelope data for all messages in a mailbox and then sort the messages based on the obtained data. However, clients have no way of reusing the sort result when a new message arrives, mandating at least a limited support for client-side sorting as well. This limitation is mitigated only by the `CONTEXT` family of extensions which is discussed later in this chapter.

3.4.2 Threads and Conversations

RFC 5256 [40] also includes support for organizing messages into threads. The threading algorithm takes a look at various pieces of message metadata like their subjects or the `Message-Id`, `References` and `In-Reply-To` headers and builds a tree of messages where children of a particular node represent messages which were made as responses to the parent one. This RFC specifies a few threading algorithms from almost useless one (the `ORDEREDSUBJECT` which groups together messages sharing a similar subject, effectively bundling unrelated items like generic “info”, “inquiry” etc. messages) to almost perfect ones (the `REFERENCES` which works like the `ORDEREDSUBJECT`, but also takes the special machine-readable headers into account). Sadly, even the `REFERENCES` algorithm only sorts the threads by the time stamp of the thread root and not by the latest message in a thread, effectively “hiding” new responses deep in the mailbox history. It also still looks at the message subjects, potentially lumping unrelated messages together.

Both of these limitations are removed by the REFS threading algorithm [42]. Despite being still in the draft phase, the Dovecot IMAP server includes full support for it.

> *Trojitá will use it if the* THREAD=REFS *capability is advertised. In absence of the* REFS *algorithm, Trojitá degrades to* REFERENCES *and* ORDEREDSUBJECT*, respectively.*

Unfortunately, none of these extensions addresses the need to re-download the whole thread mapping when a new message arrives; matters are in fact even more complicated than when sorting because a single arriving message could have a wide ranging effect on the whole threading information. This issue is addressed by the proposed "extended INTHREAD" described in section 4.2 on page 39.

3.4.3 Incremental Sorting and Searching

Soon after the original SORT command got standardized, it became apparent that having to request a full update of the sort order whenever a new message arrived would nullify some of the bandwidth saving opportunitites of performing the server-side sort. A similar concern existed for server-side searching — here, too, the client would have to explicitly check if the new arrivals match the search criteria. This limitation was addressed by introduction of the so-called *contexts* in RFC 5267 [43].

This RFC defines three extensions, the ESORT, CONTEXT=SEARCH and CONTEXT=SORT. The first one is similar to the ESEARCH (and in fact reuses the ESEARCH response) in that it could reduce the amount of data transmitted in response to the SORT command. The two CONTEXT=... capabilities extend the SEARCH and SORT commands by a way to tell the server that it should tell the client whenever the results are updated. An efficient way of communicating the changes through the ADDTO and REMOVEFROM ESEARCH return values was introduced.

In absence of the search/sort contexts, a newly arriving message would typically result in client repeating the operation. [5] This is turn leads to excess data transfers like in the following example where a single arrival results in transferring the whole mapping again:

```
C: x1 UID SORT (SUBJECT) utf-8 ALL
S: * SORT 1 2 10 5 50 20 ... [rest of UIDs goes here]
S: x1 OK sorted
...time passes...
S: * 1007 EXISTS
C: x2 UID SORT (SUBJECT) utf-8 ALL
S: * SORT 1 2 1111 10 5 50 20 ... [rest of UIDs goes here]
S: x2 OK sorted
```

For brevity purposes, the sort order of the other 1,000 messages has been omitted. In presence of the CONTEXT=SORT extension, though, the same protocol interaction would look like this one:

```
C: x1 UID SORT RETURN (ALL UPDATE) (SUBJECT) utf-8 ALL
S: * ESEARCH (TAG "x1") UID ALL 1:2,10,5,50,20,90:1090
```

[5] When speaking about searching, there is a pretty straightforward optimization opportunity where the SEARCH command can be augmented to include an explicit "and the messages' UID is higher than the *old_uidnext*". Unfortunately, no such optimization can be performed for the SORT command where the sort order can possibly depend on each and every other message in a mailbox.

```
S: x1 OK sorted
...time passes...
S: * 1007 EXISTS
S: * ESEARCH (TAG "x1") UID ADDTO (2 1111)
```

Using the sort contexts therefore leads to dramatic bandwidth savings on "busy" mailboxes which keep receiving new mail over time; the ESEARCH response can also occasionally contribute to a reduced amount of transferred data when the UIDs were assigned in a favorable way. The biggest contribution is, however, the introduction of the ADDTO ESEARCH return response which obliterates the need to transfer sort order of the whole mailbox along with the single updated item.

The search and sort contexts also impose certain, albeit abysmal overhead, though — whenever a message is removed, an explicit ESEARCH REMOVEFROM has to be issued. This funcitonality is mandated by the relevant RFC, even though it doesn't provide any extra functionality — all clients *still* have to listen for EXPUNGE or VANISHED untagged responses (and rightly so, obviously), so there is no technical obstacle in requiring them to remove the freshly removed items from their search/sort result cache. Etiquette also dictates that clients shall cancel these updates when they are no longer needed through the CANCELUPDATE command. These drawback are however extremely small when working with larger mailboxes and absolutely worth the increased benefit of not transferring the whole set of UIDs over and over again.

> *Trojitá includes full support for both* CONTEXT=SEARCH *and* CONTEXT=SORT*. The* CONTEXT=SEARCH *has been tested for interoperability against the Dovecot IMAP server, one of the few implementations which actually offer this functionality. Unfortunately, I was not able to locate a single IMAP server supporting* CONTEXT=SORT*, so no real-world interoperability tests could have been performed.* [6] *This is a rather surprising outcome given that the RFC 5267 is not exactly a fresh standard now and that its authors work from a commercial software vendor offering a pretty advanced IMAP server implementation.*

Sadly, no equivalent of these incremental updates is defined for the THREAD command. I suspect that this is caused by the fact that a single new arrival can affect each and every message in the previously received thread mapping. I have attempted to address this limitation by augmenting the SEARCH=INTHREAD draft, see section 4.2 on page 39 for details about the selected approach.

3.4.4 Advanced Searching

Although the basic IMAP specification provides quite a rich set of features aimed at searching the currently selected mailbox, the specification leaves quite a fertile ground for improvements.

The biggest problem with IMAP's searching is that it requires a strict substring matching, a requirement which is openly ignored by at least Google's IMAP implementation [44]. Google's partial answer to this problem is at least an ability to issue "raw" GMail-like searches through the X-GM-RAW SEARCH operator [45]. Others have tried to add a similar feature through the SEARCH=FUZZY extension as defined in RFC 6203 [46].

[6] As any other feature of Trojitá, though, the support for CONTEXT=SORT is covered by the automated test suite which was carefully written to minimize the chance of any unwanted side effects and guard against unintended results when the servers deviate from the expected behavior.

Trojitá makes use of the fuzzy searching if available and announced in the capability response. There were also some experimental extensions [47] aimed at introducing search based on regular expressions, but they have not gained much traction.

A long-missing feature from IMAP is an ability to search multiple mailboxes at once. A very crude hack is the (unofficial) `SCAN` extension [48] which is said to be private to the University of Washington's `uw-imapd` daemon. More recent attempt at tackling down this problem is the `MULTISEARCH` extension [49] built on top of the `NOTIFY` [33]. As of July 2012, no publicly available IMAP servers have announced support for this extension.

3.4.5 Obtaining Statistics for Other Mailboxes

Many IMAP clients start their session by requesting a `LIST` of all top-level mailboxes. This command is then followed by a `STATUS` for each of them in order to obtain information like the number of messages in each mailbox. The obtained information is typically used in a GUI of some kind to show the mailbox list to the user.

The basic IMAP specification unfortunately doesn't convey the critical piece of information about whether a mailbox contains any child mailboxes — a data point typically required by any GUI to be able to show a proper widget for opening a list of these child items. In absence of the `CHILDREN` extensions, as defined by RFC 3348 [50], clients have no choice but to issue an explicit `LIST` command for all mailboxes, [7] trying to list their children.

The `CHILDREN` is a pretty straightforward extension which shall arguably be supported by any IMAP server worth its salt; its absence does not improve the situation in any way.

Trojitá supports it fully and will gracefully fall back to extra `LIST` *requests in case it is not available.*

The initial discovery of mailboxes also mandates a separate `STATUS` command for each mailbox — behavior which arguably goes against the spirit behind that command which was intended to *not* serve as a generic "tell me about updates to other mailboxes" feature. This initial idea no longer has its merit, unfortunately — users simply *expect* being able to see a number of unread messages right next to the mailbox name, and client authors have to deliver this information to them. Having to send an extra `STATUS` command for each mailbox during the initial discovery is not too evil thing per se (the total wasted bandwidth is negligible when compared to mailbox synchronization), but worth optimizing anyway. RFC 5819 [51] adds an option to the `LIST` command to request automatic sending of the untagged `STATUS` command along.

When the IMAP server announces the `LIST-STATUS` *capability, Trojitá will automatically make use of this extension.*

3.4.6 Push-notification of Other Mailboxes' State

A feature most notably absent from the `IDLE` is any support of passively monitoring changes of non-selected mailboxes. Over the time, many extensions have appeared in the state of various drafts [52] [53] [54], often simply requesting unsolicited delivery of

[7] The only exception being those marked with the `\NoInferiors` which is meant to indicate that this mailbox could *never* contain any child mailboxes, perhaps due to technical limitations on the server side — a very different case from not containing any child mailboxes at *this* time.

STATUS and FETCH responses. None of these extensions gained widespread support nor reached the state of a proposed standard, though.

Said status was reached by the NOTIFY extension codified in RFC 5465 [33]. It adds an impressive amount of features; compliant agents can listen for creation and deletion of mailboxes (eliminating the need to redo a top-to-bottom LIST discovery), changes in amount of messages in specified list of mailboxes and even for their flag changes. Sadly, support for this extension is scarce among the existing IMAP servers and its author reportedly has mixed feelings [55] about its fate where basically "noone implements it". In early 2012, a posting on the Dovecot mailing list announced [56] preliminary support for a part of this extension in Dovecot, one of the most widely deployed open source IMAP daemons. Unfortunately, this code was not complete as of July 2012 [57] and the branch I have tried contained regressions which prevented regular use of other features of the IMAP server altogether. [8]

> *Due to not being able to verify the* NOTIFY *operation against any available IMAP server implementation, Trojitá will not try to leverage this extension for the time being.*

3.5 Composing and Delivering Mail

The baseline versions of the IMAP protocol does not offer any substantial assistance in composing and delivery of new messages — the only feature even remotely related to this topic is the APPEND command which saves a message passed by the client into a mailbox. Over the time, several extensions appeared aiming at improving this area.

The first extension is the MULTIAPPEND command (RFC 3502 [58]) which allows the client to atomically upload many messages at once. Having such a feature could be a terrific boon in clients which support batched import of data from the existing mail store, but it is not so valuable in a generic client.

> *If Trojitá grows and a support for batched import becomes a wanted feature, the* MULTIAPPEND *command will doubtlessly contribute to a smoother experience.*

Much more useful is the CATENATE extension [59] which allows clients to build a message from a combination of uploaded parts and data already available on the IMAP server. This extension is crucial for implementing advanced forward-without-download feature. Suppose a user who is currently on her vacation high above some Nordic fjord, accessing e-mail over a metered GPRS connection, has just received a huge e-mail consisting of a big binary attachment. IMAP already has a feature which allows her to check the accompanying text without downloading the full message body. What is missing is some kind of support of forwarding the original message to another recipient.

This task consists of several steps — first of all, the body of the resulting message to be sent has to be composed. The CATENATE extension tremendously helps with this task. Using CATENATE, a client can compose a message consisting of a mixture of data, some of which is coming from the client over the network as raw literals, others being recycled from server-side data, be it full messages, arbitrary message parts or even byte-sized chunks of these. After the message composing is concluded, the message shall

[8] This is not to say that the Dovecot is a bad IMAP daemon — not at all. The version which was used is a development snapshot which is clearly marked as an experimental version which requires future work.

be submitted to an MTA [9], typically over the SMTP protocol [60]. Finally, a copy of the message shall be stored in the user's "sent" folder for future reference.

> *Trojitá will make use of the* `CATENATE` *extension when available.*

It can be seen that in absence of specialized extensions, this an interaction could possibly involve up to three transfers of the huge binary data, possibly in an inefficient transport encoding, over the unreliable or expensive network connection. Clearly, there's a huge room for improvements. The `CATENATE` extension assists in a server-side IMAP message assembly, but does not provide a way of improving actual message submission.

The first possible way of improving relies on a whole family of extensions concerning both SMTP and IMAP. The SMTP protocol is extended by the `BURL` command [61] while the IMAP server has to support the `URLAUTH` extension [62] and both daemons have to be properly configured. Using this protocol combination (which itself depends on quite a few more extensions, please see the respective RFC documents for details), the IMAP client can generate a single-use authorization token which — if used — enables its holder to access the given message part. This token is then passed to the SMTP daemon which will combine data obtained directly from the MUA[10] (like the accompanying text) with the data downloaded from the IMAP server via the passed authorization token, build a MIME message and take care of its further delivery through the usual means. As was already mentioned, if the IMAP server also supports the `CATENATE` extension, the client can build the message on the server at once from the mentioned fragments (the new accompanying text and the attachment from the original message) and pass this newly-formed message for submission through the `BURL` SMTP extension. This has a potential of eliminating the need to transfer the data to/from the client *at all*, leading to a drastic bandwidth reduction.

> *Trojitá includes full support for the* `BURL` *and* `URLAUTH` *extensions. Due to the interoperability troubles with misconfigured servers which were observed in real world [63], making use of* `BURL` *has to be explicitly allowed in the settings dialog of the application.*

Unfortunately, such mode of operation comes at a cost. Support for the required extensions is not ubiquitous among the deployed servers and even if the code contains all required features, it is often a policy decision whether an IMAP server should ever allow access to possibly privacy-sensitive data to the outbound MTAs.

Another possibility presenting slightly different set of features, advantages and disadvantages is implemented by deferring the actual message submission to the IMAP server as well. The `CATENATE` extension is still useful because it shall nonetheless be used to build the MIME message body from existing parts, eliminating the need to upload the data to the IMAP server. When the message body is built (no matter how, using `CATENATE` or deferring to legacy means), the IMAP server can be asked to submit the resulting data for delivery.

This former approach has been traditionally dismissed by many IMAP proponents, including Mark Crispin, on various grounds. The most common complaint is that the SMTP protocol is extremely mature, widely deployed and also complex — and that this inherent complexity is *required* by various use cases crucial for proper operation of today's Internet mail. The critics often argue that covering the whole feature set

[9] Mail Transfer Agent

[10] Mail User Agent, i.e. the program which acts on the user's behalf in accessing and submitting the Internet mail

of the (E)SMTP-based submission would lead only to an equivalent of tunneling the SMTP session over IMAP [64] [65], an outcome which is clearly not desirable. This approach was also subject to various standardization efforts, often literally tunneling the (E)SMTP conversations [66, p. 30] over an IMAP connection. Proponents of this submit-over-IMAP approach, however, counter-argument by stating that optimizing for a common use case has its own merits [67]. The critics concur that having two ways of obtaining an identical result is suboptimal and that an easier, yet more limited way would threaten the existence of the older, more flexible solution [68].

Finally, a third way of solving the forward-without-download problem was presented in September 2010 through the POSTADDRESS extension [69]. Using this mechanism, a client first obtains a valid e-mail address serving as a "delivery address" for the user's *Sent* mailbox. After getting hold of that information, the SMTP session then proceeds as usual with one difference — the obtained address is added as another recipient of the submitted e-mail message.

> *Besides including full support for the already mentioned "Lemonade trio" of extension (*CATENATE, URLAUTH *and* BURL*, Trojitá also includes experimental support for the second approach of forward-without-download, the submission-over-IMAP variant.*

Mail submission is an important topic, so I have dedicated a full section to a more detailed analysis of various advantages and disadvantages of competing approaches. More information is available in section (section 4.3 on page 41) where I present my Internet-Draft documenting a proposed extension in which I've tried to address many concerns raised during the previous discussion rounds.

3.6 Further Improvements

Many additional extensions have been defined over years, covering various areas of the protocol. This section deals with those extensions which do not quite fit into any of the previous categories.

3.6.1 Debugging

Most of other communication protocols contain a way of letting the other party know what software implementation and in which version it is talking to. In the web, this is usually accomplished through the `User-Agent` and `Server` headers, e-mail messages often contain either an `X-Mailer` or `User-Agent` field in theirs RFC 2822 headers, etc. IMAP adds a similar feature through the ID extension defined in RFC 2971 [70].

This RFC is pretty clear on that implementations are explicitly *forbidden* from using *any* knowledge obtained through the ID extension to alter their behavior. This is a reasonable decision intended to prevent clients implementing blacklists and whitelists of "known" servers. All IMAP protocol speakers are intended to only use the CAPABILITY responses (and the ENABLE extension, if present) to change their behavior.

> *Trojitá is fully compliant with these requirements.*

3.6.2 Internationalization

The "internationalization" RFC (RFC 5255 [71]) is focused on server implementations, mainly specifying how to perform search/sort collation under various circumstances. That said, it also presents two commands to the IMAP clients. The first of them is the

LANGUAGE command suitable for changing the language in which various error and notifying messages are generated and sent by the server.

> *Trojitá tries hard to rely on machine-readable response codes (RFC 5530 [72]) instead.*

The second command is the COMPARATOR, a feature intended to let clients specify which comparators and collators to use when performing search or sort operations (these comparators are defined in RFC 4790 [73]). By default, servers supporting at least the I18NLEVEL=1 extension are required to perform collations using the `i;unicode-casemap` comparator [74] — a feature which is very useful (and often sufficient) in countries using the Latin alphabet. Those servers which support I18NLEVEL=2 also accept client-specified preference about how to perform these operations.

> *Adding support for this higher level would be trivial on a technical front (the* LANGUAGE *and* COMPARATOR *commands are very simple with much more demanding requirements on servers), but no requests for such a feature in Trojitá were received yet — suggesting that the* `i;unicode-casemap` *comparator works well for most users at this point.*

There is also the experimental "UTF-8" RFC [39] whose aim is to get rid of any non-UTF8 data being transferred over IMAP. Unfortunately, as designed, this document presents backward-incompatible changes to the IMAP protocol and is hence not widely supported by the common server implementations. The client authors would very much prefer to stop dealing with various Unicode encoding schemes, but this RFC does not completely address all of the issues. An ongoing discussion is nevertheless taking place on official IETF's channels; it will be promising to watch its future and especially the fate of the "5738bis" Internet Draft [75].

> *It shall be noted that Trojitá's source is completely ready for internationalization and localisation of the application.* [11]

3.6.3 Other Supported RFCs

Trojitá is fully conforming to the description of the "Distributed Electronic Mail Models in IMAP4", as defined in RFC 1733 [77]. It also respect the recommendations about concurrent access to mailboxes from RFC 2180 [78], generic suggestions to the IMAP implementors (RFC 2683 [79]), Mark Crispin's famous "Ten Commandments of How to Write an IMAP client" [80] and Timo Sirainen's suggestions for client authors [81] [82].

It will make use of the UNSELECT command for internal technical reasons (RFC 3691 [83]), if available; if it isn't present, it will gracefully revert to using fabricated mailbox names with the EXAMINE command from the baseline IMAP specification. This failover is thoroughly verified by a suite of automated unit tests.

Trojitá is capable of recording responses from the UIDPLUS extension (RFC 4315 [84]); the author have also contributed [85] to the discussion related to the COPYUID equivalent for the proposed UID MOVE command.

[11] This support does not come at no cost, unfortunately. Qt's `QDateTime` class which is used for keeping track of the date and time information in all Qt-using projects unfortunately does not support tracking of timezone information [76]. Trojitá works around this limitation of the public API by manual hack in `Imap::Mailbox::formatDateTimeWithTimeZoneAtEnd()` method.

The code also includes full support for recognizing various extensions to the IMAP's grammar (RFC 4466 [86]), response codes (RFC 5530 [72]) and the reserved set of keywords (RFC 5788 [87]).

RFC 5258 [88] is used when available to explicitly encourage the IMAP server to send additional metadata in `LIST` responses — the biggest benefit of this extension is eliminating the need of sending explicit `LSUB` requests to discover mailbox subscriptions.

Finally, the Lemonade extension (RFC 5550 [89] and the obsolete RFC 4550 [90]) has compiled a set of requirements crucial to the mobile IMAP e-mail clients. Comparison of the proposed set of extensions with the Lemonade profile is presented in a dedicated section of this thesis on page 46.

3.6.4 Out-of-scope Features

The extensions presented so far all have a certain affinity towards the "mobile IMAP". Many other extensions have been introduced, though, often solving a real problem.

The first family of extensions with debatable merit are extensions providing support for so-called referrals. The RFC 2221 [91] adds a mechanism for servers to redirect clients based on their identity, a feature which was originally supposed to come handy in large corporate environment. Similar to that, the RFC 2193 [92] adds mailbox referrals, a feature where a subset of user's mailboxes might be stored on a remote server. As it happens, these features have not attained a big market share among client developers [93] and the servers which supports that are generally willing to act as a transparent proxy for their clients anyway [94].

Extensions which are useful in a general-purpose e-mail client are the `NAMESPACE` extension (RFC 2342 [95]) which would allow compliant clients to automatically discover where e.g. other users' mailboxes are located, support for managing access-control lists (ACLs) on the server (RFC 4314 [96] and its obsolete form given in RFC 2086 [97]) and finally support for reporting and managing storage quotas (RFC 2087 [98]).

> *These have not been implemented in Trojitá yet.*

A mechanism for increasing interoperability with organizations which have invested in a single-sign-on infrastructure like Kerberos could be improved through better support for SASL (RFC 1731 [99], RFC4959 [100]).

> *A request from users which would very much prefer to have a GSSAPI-enabled Trojitá was already received, but unfortunately this feature remains unimplemented due to time constraints.*

A few extensions might improve the general comfort of users setting up their e-mail clients for the first time. Without any doubt, autoconfiguration through the DNS SRV records (RFC 6186 [101]) falls into this category.

There are also certain features which might add a whole new level of functionality to working with e-mail – examples are the `ANNOTATE` extension (RFC 5257 [102]) for adding arbitrary annotations to individual messages or even their parts, which is still marked as experimental, or the similar `METADATA` feature (RFC 5464 [103]) adding the same functionality on a server or mailbox level. Needless to say, support for these extensions is scarce among the IMAP clients.

Other extensions try to fill a certain niche. Examples are the `WITHIN` extension (RFC 5032 [104]) which allows clients to search among messages of a certain age or the `SEARCHRES` (RFC 5182 [105]) adding a low-level pipelining optimization which would allow the client to re-use the previous search result in the subsequent commands. RFC 5466 [106] adds support for persistent storage of search criteria on the server through the already mentioned `METADATA` extension.

> *I have not found a use case for having these optional extensions utilized from Trojitá in any place.*

The RFC 3503 [107] deals with how to generate the message delivery notifications (MDNs) in IMAP.

> *This document clearly does not apply to clients which on purpose do not create MDNs for privacy reasons, such as Trojitá.*

Finally, certain extensions improve the user experience in specialized environments. One of them is RFC 5616 [108], an extension aimed at "Streaming Internet Messaging Attachments". One could imagine a use case where a carrier-level voice mailbox was implemented over IMAP; in similar situations, such a solution would have its merit. At the same time, this specific extension has so special requirements on the network architecture that it is clearly out-of-scope for a general-purpose e-mail client merely running on a cell phone.

3.7 Obsolete Extensions

IMAP is a rather old protocol (its history is slightly older than the author of this thesis, for that matter). Certain features have been therefore deprecated over time and it took years to grow to the current IMAP4rev1 version from the old standards of IMAP2 [109] [110], IMAP3 [111] and IMAP4 [112]. Attention has been paid to make this transition as smooth as possible through various compatibility recommendations [113] [114] [115] [116].

> *Trojitá requires the server to at least announce the* `IMAP4rev1` *capability. This protocol revision is currently defined by RFC 3501 [2] from March 2003; however, changes since the previous version from December 1996 ([114]) are mostly backward compatible — and in practice, no report of Trojitá not being able to work against any IMAP server implementation out there were received.*

The `UIDPLUS` extension got redefined from its former shape [117] to the current revision [84]; the new document states that the reason for this revision was to prevent sending of bogus UID replies when the target mailbox did not support persistent UIDs.

> *As such, Trojitá can deal with both revision of the* `UIDPLUS` *document.*

Chapter 4

Proposed Extensions

Previous chapters have shed some light on the complicated world of IMAP and showed how the protocol limitations affects the users' experience. I have also introduced some of the existing extensions which aim to address many of the presented shortcomings. There are still quite a few issues which make lives of the client implementors harder than necessary, though. At this point, I present three separate extensions which fix race conditions, improve the effectiveness of the protocol or add new features which contribute to smoother operation of the e-mail clients. This broad range of changes was selected to illustrate that improving IMAP can happen on many different fronts and that even after more than twenty years of "active service", the protocol can be actively improved to address newly emerging trends.

Internet Drafts are usually prepared in a special system [118] which handles the required strict document formatting using plain ASCII text. This chapter is therefore purposely very short, providing only the minimal descriptions of the proposed extensions. The Internet Drafts themselves are included in Appendix A on page 64 and are an integral part of this thesis.

4.1 Announcing the UIDs of Newly Arriving Messages during the QRESYNC mode: the ARRIVED Extension

The first extension I have implemented addresses a race condition in the `QRESYNC` extension [30]. In `QRESYNC`, the offset-based `EXPUNGE` responses known from the baseline IMAP protocol are replaced by `VANISHED` responses which use UIDs. Unfortunately, because the `EXISTS` still informs about the number of new deliveries only, without including the UIDs, and due to the fact that the IMAP server is explicitly allowed [1] to include non-existing UIDs in the `VANISHED` responses, a race condition exists where client does not know about the full span of the sequence $\rightarrow$ UID mapping,

[1] *"Note that a VANISHED response caused by EXPUNGE, UID EXPUNGE, or messages expunged in other connections SHOULD only contain UIDs for messages expunged since the last VANISHED/-EXPUNGE response sent for the currently opened mailbox or since the mailbox was opened. That is, servers SHOULD NOT send UIDs for previously expunged messages, unless explicitly requested to do so by the UID FETCH (VANISHED) command."*

"Note that client implementors must take care to properly decrement the number of messages in the mailbox even if a server violates this last SHOULD or repeats the same UID multiple times in the returned UID set. In general, this means that a client using this extension should either avoid using message numbers entirely, or have a complete mapping of UIDs to message sequence numbers for the selected mailbox." [30, p. 12] — in the RFC language, *SHOULD* means that implementations are suggested to use the recommended behavior, but can deviate from that as *"there may exist valid reasons in particular circumstances to ignore a particular item"* [119].

which in turn violates RFC 5162's requirement on clients having *"a complete mapping of UIDs to message sequence numbers for the selected mailbox"*.

The proposed extension addresses this issue through the ARRIVED response which informs the clients about the UIDs of new message arrivals. At the same time, it improves the protocol efficiency by freeing the clients from a requirement to explicitly ask for message UIDs when a new message is delivered.

This is how a typical session without the ARRIVED extension looks like. Suppose the mailbox previously contained just a single message with UID 5, the UIDNEXT is 11:

```
S: * 3 EXISTS
S: * 2 FETCH (FLAGS (foo))
S: * 3 FETCH (FLAGS (bar))
S: * VANISHED 12:20
C: x UID SEARCH UID 11:*
S: * SEARCH 21
S: x OK Search completed
C: y UID FETCH 21 (FLAGS)
S: * 2 FETCH (UID 21 FLAGS (foo))
```

The client had no chance but to ignore the unsolicited FETCH responses, recover the full UID mapping through the UID SEARCH command and finally re-request the flag data once again through the UID FETCH command.

In contrast to the above, the following is how the same session happens when QRESYNC is active and enabled:

```
S: ARRIVED 21
S: VANISHED 12:20
S: * 2 FETCH (FLAGS (foo))
S: * 3 FETCH (FLAGS (bar))
```

No client activity at all is required then the ARRIVED extension is available.

4.1.1 Alternatives

In absence of the ARRIVED extension, clients are required to perform an explicit UID rediscovery, possibly through the UID SEARCH $formerUidNext$:*; this could pose a problem when servers send any data using the FETCH responses without the UID field. Always including the UID in unsolicited FETCH responses, as recommended in RFC 5162's errata document, can mitigate this particular issue. However, servers which already do not send non-existing UIDs in the VANISHED responses will still benefit from implementing the ARRIVED extension as the clients will be able to refrain from performing an explicit UID SEARCH operations on them upon new deliveries. Furthermore, due to the fact that clients have *no way* of finding out whether servers include the non-existing UIDs in VANISHED responses, the benefits of the proposed extension are demonstrated whenever new arrivals are expunged before their UIDs become known, no matter whether the servers conform to the requirement imposed by the relevant errata.

Full text of the proposed extension in the format of an Internet-Draft suitable for IETF submission is included in section section A.1 on page 64. This extension was presented for discussion on the imap-protocol mailing list [120].

4.2 Improving Incremental Threading through Modified INTHREAD

Delegating message threading to the server-side can provide clients with enormous benefits, especially when working with large mailboxes. However, these benefits can be significantly reduced when clients are forced to request full thread mapping over and over again.

Unfortunately, that is exactly the situation when new messages arrive. When clients use the server-side threading, they by design do not have to keep track of the `Message-Id`, `References` and `In-Reply-To` headers as the thread tree building is all done by the IMAP server. However, that also means that newly arriving messages cannot be easily "plugged" into the already known tree, even if full header set of the new arrival was known. Doing so reliably would require knowledge of the relevant headers of all messages in mailbox, knowledge which is rather expensive to obtain and avoidance of which is the whole point of server-side threading.

4.2.1 Existing Approach

Extensions exist solving this problem for both searching (the CONTEXT=SEARCH extension from RFC 5267 [43] which is reasonably wide-deployed) and sorting (the CONTEXT=SORT, support of which is extremely scarce despite being defined in the same RFC document), but no such proposal was ever submitted for threading. I suspect that the reason is inherent in the way the threading works — a single newly arriving message can indeed cause threading updates for *any* other message in a mailbox, even for *all* of them in a pathological case. [2] This is in a strong contrast to live updates of search results where a pair of simple "add item X to result" and "remove X from the result" is enough, or even to incremental sort order communication (where the operation is complicated a little more, requiring "add item X to the result at offset Y").

One *could* attempt to try the incremental threading by asking for UIDs of messages with the `Message-Id` header matching any referenced from the new arrivals' `References`, but doing so is hard in practice as some MUAs choose not to use `References` and set just the `In-Reply-To` header. Thread building exclusively through the `In-Reply-To`, however, completely misses the correct thread order for threads with "gaps" in them, a scenario very common when one's own replies are located in a dedicated *Sent* folder with the rest of the thread in e.g. the INBOX. Using just the basic SEARCH command is therefore not sufficient.

An existing draft proposal [42] extends the search query capabilities with the INTHREAD operator matching a *whole thread* of messages. This capability alone is not enough to fully accommodate the whole incremental threading problem, but it is an improvement good enough to build upon. The only missing piece of functionality is being able to tell where the new thread shall be positioned, but in absence of better tools, an approximation always showing threads with "new arrivals" at the very end of the view might be good enough. [3]

[2] Any new arrival could possibly join many existing threads previously considered to be individual and independent of each other to a single thread having all of them as subthreads.

[3] Displaying threads with "new arrivals" among the "last messages" is suboptimal because the newly arriving message *could* be in fact a result of a copy operation. Having these "older" messages suddenly appear should not interleave them with the recent content of the mailbox.

4.2.2 The INCTHREAD Extension

The proposed extension builds on top of INTHREAD, adding the exact positioning to each individual thread matching the search criteria. In addition, the format of the response does not use the THREAD untagged response from RFC 5256 [40], but instead uses the extensible ESEARCH response from RFC 4731 [25] — the ESEARCH already contains provisions for returning multiple types of data and as an added bonus ties the response to a particular command's tag, making it possible to parallelize threading operation. These reasons make the ESEARCH response better suited to accommodate the new result format.

The improvement, as measured in decreased bandwidth consumption, is not always as impressive as those from the CONTEXT extensions. It is conceivable that more advanced forms of conveying the modified threading information (e.g. the strictly incremental responses about how an update affects a particular branch in the threading tree) would result in smaller overall transmitted octet counts, but on the other hand implementing such an extension would prove challenging to both servers and clients. There are also circumstances under which the tree-oriented differential updates would not work; the most obvious one is when a client reconnected to a mailbox, detected a couple of new arrivals and now wants to ask for updates "ex post", long after they have physically happened in the mailbox. Finally, the client-driven mode of operation through the proposed extension better fits the IMAP behavior where clients specify in what kinds of information they are interested and servers optimize their operation by only transmitting what is strictly necessary.

Using the proposed extension, a typical communication between two compliant IMAP protocol speakers might look like the following:

```
S: * 666 EXISTS
C: x1 UID FETCH 665:* (FLAGS)
S: * 666 FETCH (UID 1666 FLAGS ())
S: x1 OK fetched
C: x2 UID THREAD RETURN (INCTHREAD) REFS utf-8
     INTHREAD REFS 666
S: * ESEARCH (TAG "x2") UID INCTHREAD 400
     (600 601 (640 666)(602 603))
S: x2 OK sent
```

At first, the server informs the client about new delivery. Client responds with a request for UID and message flags of the new arrivals. When both are known, the new form of the UID THREAD command is issued, specifying that the threading algorithm REFS shall be used, searching shall be done in the utf-8 character set and that the returned value shall include the relative thread position among other threads in the whole mailbox.

The result instructs the client that messages with UIDs of 600, 601, 602, 603 and 640 shall be removed from their previous positions in the threading tree, and that they together with UID 666 form a new thread with the specified shape. This new thread immediately follows a thread whose thread root has UID 400. Figure 4.2.2 shows how the new threading for this mailbox looks like.

Even though the updated extended THREAD command can still send data which are already known by the client, the design is a reasonable compromise between one imposing overly complex requirements on both clients and servers on one hand, and needlessly transmitting the whole mapping over and over again in absence of any extensions.

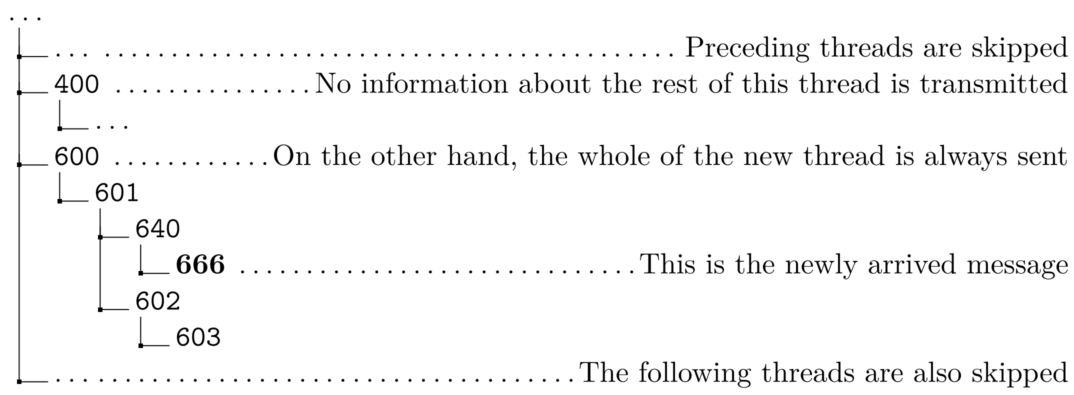
...
... Preceding threads are skipped
400 No information about the rest of this thread is transmitted
...
600 On the other hand, the whole of the new thread is always sent
601
640
666 This is the newly arrived message
602
603
The following threads are also skipped

Figure 4.1: The incremental threading response conveyed information about one message thread in a mailbox. No data for other threads have to be transmitted, leading to a significant performance improvements on slow networks or bandwidth-constrained devices.

The full text of the proposed extension, including the formal grammar, is available from section section A.2 on page 72. The extension was presented for expert review on the `imapext` mailing list [121].

4.3 Submitting Internet Mail — the SENDMAIL Extension

Message submission is one of the controversial subjects, along the "imap5" and "move messages" discussions — whenever any of these topics is brought up on the imap-protocol mailing list, an interesting discussion is guaranteed to happen. In this proposal, I have tried to accommodate criticism from many previous review rounds.

The baseline IMAP protocol does not offer any way of e-mail submission. Mail User Agents willing to send mail are supposed to use the (E)SMTP protocol [122] [123], preferably over a dedicated submission service [60]. This is how most of contemporary e-mail clients (at least those using the IETF standards in contrast to the proprietary ones) work, but it also brings along a set of issues.

First of all, the clients have to be *properly configured*. Given that ESMTP and IMAP can be (and often are) managed separately, clients have to ask their users for two sets of accounts, one for each type of service. Proposals exist trying to eliminate much of this complexity, especially through the DNS system [101] or via non-standard mechanisms like those proposed by Mozilla [124] — but as usual, these mechanisms often cover only a subset of service providers. Client programmers are required to implement and test full support for both protocols. IMAP is doubtlessly the more complicated one, exceeding ESMTP both in syntax and semantic, yet adding a requirement for a proper SMTP implementation causes a measurable burden on the developers.

Furthermore, network firewalls and other filters along the way have to be properly configured to allow for a reliable pass-through for both services [125]. Even though the situation has much improved with a dedicated "Submission" service [60] which moved the e-mail submission to a dedicated port to not interfere with the traditional SPAM-laden TCP port 25, there are still certain situations where customers cannot use both e-mail services, leading to confused support calls [126] [127].

In addition to the above, many users wants to store their outgoing e-mail in a separate IMAP mail folder. This means that under typical circumstances, a message being sent has to be uploaded twice over the network, once for IMAP, the second time for ESMTP delivery. In case when a message contains an attachment previously already available on the IMAP server, the same data can in fact travel over the network *three times* – at first when being downloaded by the IMAP client only to be subsequently sent after the proper MIME encapsulation to the destined "Sent" folder, and finally over SMTP as usual. As a last point in this quick list, even in presence of specific extensions, the time required to actually *establish* a separate connection, setup proper TLS confidentiality and start tunnelling data over it is often non-negligible.

All of the above suggests that there are certain benefits in choosing to deliver e-mail messages from MUAs [4] over IMAP.

4.3.1 Competing Proposals

Over the years, many proposals have appeared trying to accommodate this issue [128].

The "Lemonade Trio"

The most widely deployed mechanism aiming at bandwidth reduction is the Lemonade extension family [89]. Through the use of IMAP's `CATENATE` [59] and `URLAUTH` [62] along with SMTP's BURL [61], conforming clients can:

- compose a message on the IMAP server's side, reusing any existing data,
- deliver that message over SMTP without having to upload the data.

At the same time, this approach has the following disadvantages:

- a trust relation (at least a limited one) has to exist between the SMTP and IMAP servers,
- both SMTP and IMAP servers have to be properly configured,
- clients still have to maintain a separate SMTP protocol stack,
- an extra connection has to be opened.

Trojitá includes full support for these extensions. However, because there is *no* way of discovering whether the IMAP and SMTP daemons "trust" each other, [5] Trojitá requires the user to explicitly enable a checkbox in the settings dialog to activate these features. [6] Using this explicit confirmation is intended to deter bugs like those which have plagued other MUA implementations [63] from affecting Trojitá. This implementation might change in future.

[4] Mail User Agents

[5] The mere presence of the `URLAUTH` capability on the IMAP server side and the advertised `BURL` extension by the ESMTP service does not imply that an eventual submission will succeed.

[6] The `CATENATE` extension is not subject to this limitation; it will be used whenever the server announces its presence, unless the user has explicitly forbidden its usage. This is equivalent to how Trojitá handles any other IMAP extension.

Tunneling SMTP inside IMAP

A second approach in which the IETF community and related researchers have tried to tackle down the e-mail submission was via actually tunneling the real ESMTP session through the IMAP protocol [66, p. 30]. This approach removes the burden of establishing a second connection, but retains the required complexity of having to ship and test a full ESMTP client stack. This approach is *by definition* as flexible as any future ESMTP extension and does not require any changes on the (ESMTP) server side (besides support for SMTP pipelining [129]), with only a limited amount of modifications for the IMAP clients. On the other hand, the requirement to tunnel a second protocol through IMAP adds a lot of complexity to their interaction and it appears that either the IMAP daemon or the SMTP server has to include support for BURL nonetheless. One has to wonder if the ESMTP serialization, no matter how useful when speaking ESMTP, can be replaced with something terser. Consider the following example from the proposed draft:

```
C: a004 XDELIVER CAPABILITY
S: * XDELIVER CAPABILITY (8BITMIME EXPN HELP)
C: a005 XDELIVER TEXT {123+}
C: EHLO
C: MAIL FROM: john@smith.com
C: RCPT TO: mooch@owatagu.siam.edu
C: DATA
C: URL /Inbox;UIDVALIDITY=9999/;UID=33;Section=BODY
.
S: * XDELIVER {321}
S: 220 mail.metastructure.net ESMTP
S: 250-mail.metastructure.net
S: 250-AUTH LOGIN CRAM-MD5 PLAIN
S: 250-AUTH=LOGIN CRAM-MD5 PLAIN
S: 250-PIPELINING
S: 250 8BITMIME
S: 250 ok
S: 250 ok
S: 354 go ahead
S: 250 ok 1126337586 qp 28229
```

This communication is indeed rather verbose. The same result is achieved in a clearer way through the `UID SUBMIT` command I propose later in this chapter.

No known deployments of these drafts exist and no further standardization process has been observed on the relevant mailing lists.

The POSTADDRESS Draft

For the sake of completeness, one should also mention the `POSTADDRESS` draft [69]. This extension tried to provide a way for servers to announce an Internet e-mail address for each mailbox which could act as the "Sent" folder. The idea behind this proposal was that clients should obtain this e-mail address and include it in the `Bcc` field of the outgoing e-mail messages. Doing so would facilitate the same result as the `APPEND` command, but without having to send the data explicitly. Drawbacks of this method included privacy concerns and the fact that this extension might not work with Sieve or other server-side filtering [130]. As of July 2012, this idea appears to have been abandoned for good.

4.3.2 The SENDMAIL Extension

In mid 2011, a few requests for e-mail submission over IMAP have appeared on the `imap5` mailing list [131] (with the expected outcome of calling names [132]). The idea presented by proponents of the "submit mail over IMAP" camp appeared to be that:

- Using two protocols "for e-mail" is a significant source of support requests for large service providers.
- The IETF-approved approach to the "forward-without-download", i.e. the URLAUTH and BURL extensions, are not widely deployed. They are also notoriously hard to implement and deploy for server vendors and system integrators.
- Extending IMAP to allow for message submission simplifies the number of authorization channels.
- Handling the "common case" in an efficient manner outweighs the drawback of enabling a second e-mail submission protocol.

Based on the said discussion, it appears that there is a strong demand for having "such a feature" in IMAP. I've therefore read through various mailing list archives, studied previous iterations of the discussion and tried to address many issues which were previously considered to be a blocking issue (like the apparent need to rewrite message bodies when dealing with blind-carbon-copies (the `Bcc` headers), having to scan message contents unconditionally, or a lack of delivery status notification (DSN) control). The extension which I propose as a part of this thesis has the following advantages:

- It removes the need for clients to speak both ESMTP and IMAP protocols.
- It reduces the amount of account details to ask users for.
- Whole communication is performed over a single connection, eliminating a significant cause of support requests.
- All features can be implemented as a thin wrapper over a `sendmail`-compatible binary which is nowadays shipped by most MTAs.
- Messages can be submitted using a single round trip once stored on the server.
- The extension plays well with CATENATE.
- Further ESMTP extensions can be trivially integrated through IMAP capabilities.

The extension proposes a single IMAP command, the UID SENDMAIL. This commands accepts a reference to an already existing message to be sent along with a complimentary list of submission options. This list is intended to serve as a substitute to the missing ESMTP envelope; in the initial version, clients can use it to specify senders and receivers or for control of the DSN options. The command is also ready for future extensibility; other options can be easily added to the specification when further ESMTP extensions are defined.

A typical conversation with a SENDMAIL-capable IMAP server can therefore look similar to the following (note that white space has been added to the UID SENDMAIL command for clarity):

```
C: x UID SENDMAIL 123 (FROM "foo@example.org"
     RECIPIENT "a@example.org"
     RECIPIENT "b@example.org"
     DSN (delay failure)
   )
S: * 10 FETCH (UID 123 FLAGS (\$Submitted))
S: x OK Message passed to the sendmail binary
```

Changing Trojitá to support e-mail delivery via the proposed extension was just a matter of plugging another implementation of its abstract MSA [7] interface.

Full text of this specification is available in section section A.3 on page 83. An interesting discussion followed [133] after I posted my initial draft to the `imapext` IETF mailing list.

[7]Mail Submission Agent

Chapter 5

The Mobile IMAP

Many of the existing IMAP extensions discussed in section 3 on page 16 have the potential of improving the client's operation tremendously. At the same time, experience has shown that there is a certain chicken-and-egg problem with new proposals where server vendors are not willing to invest their time into promising extensions which no client supports yet and clients are not interested in implementing extensions which they cannot test for interoperability. In this chapter, I am trying to provide a concise summary of individual merit of these extensions.

5.1 The Lemonade Profile

The Lemonade profile, as defined in RFC 4550 [90] in 2006 and later updated through RFC 5550 [89] during 2009, provides a list of extensions considered "critical" for any mobile IMAP e-mail client. The set of mandatory extensions is rather big, though, and to the best of my knowledge, there is *no* server on the market implementing all of the compulsory features. One might therefore wonder what were the reasons for this lack of general availability of the Lemonade extension family.

5.1.1 Cross-Service Requirements

One unique feature of Lemonade is the possibility to *forward messages without their prior download.* The three ESMTP [122] and IMAP extensions, often referred to as the *Lemonade trio*, namely the `CATENATE`, `URLAUTH` and `BURL`, allow the clients to compose a message using existing parts available from the IMAP mail store, provide a way of generating single-purpose "pawn tickets" for making the composed messages available to the submission server, and replacing the `DATA` SMTP command with a way of downloading the message from the IMAP server, respectively. This feature prevents having to transfer potentially huge data over the network three times — once when the users wants to read it, second time when the message is saved to the sent folder, and finally when delivering via SMTP.

Unfortunately, a big problem with said approach is the fact that it mandates collaboration across different services — an explicit trust path between the IMAP and ESMTP servers have to be set up, which is a process prone to errors [63]. This matter is also complicated by the fact that no open source MTA [1] ships with official support for `BURL`. [2] Situation is better on the IMAP server front with Cyrus supporting the

[1]Mail Transfer Agent, typically an SMTP or ESMTP server

[2]Unofficial patches exists for Postfix dating back to 2010 [134], but they have not been integrated into the mainline version as of July 2012 (the `postfix-2.10-20120715.tar.gz` development snapshot.

URLAUTH and CATENATE extensions out-of-box with Dovecot's support scheduled for its upcoming 2.2 release [135].

5.1.2 Complicated Extensions

Some of the extensions whose support is mandated by the Lemonade proposal seems to be notoriously hard for the server vendors to implement.

A perfect example is the CONTEXT=SORT extension [43]. As a client developer, I recognize its extreme usefulness and appreciate its design. Availability of such an extension would make it extremely easy to implement live-updated sorting in my Trojitá (and Trojitá *does* make use of the sort context extension). That said, given that no IMAP server which I am aware of announces its availability, clients have to deal with the status quo in the meanwhile.

The CONVERT extension [38] belongs to a similar category — the features it offers, like the server-side downscaling of JPEG images, would be *very* handy on a cell phone, yet no IMAP server known to the author includes that functionality.

Both of these RFCs were published four and five years ago, respectively, and were designed by engineers working for an IMAP server vendor. One cannot therefore dismiss them altogether as a product of people not having any say in the server development. My opinion is that the allocation of engineering resources required for shipping a particular feature in a finished product is based on another criteria than the research activity.

5.2 State of Other Client Implementations

To obtain a better understanding on how the existing solutions available on the market today use IMAP, this section takes a look at some of the most popular solutions.

5.2.1 Apple iOS

Apple's devices generally ship with a decent implementation of their IMAP stack, an evaluation shared by independent researchers [136]. The list of extensions supported by the application includes CONDSTORE and ESEARCH for improved mailbox synchronization, COMPRESS for transparent deflate compression and BURL for the forward-without download.

It is, however, surprising that their support of extensions aimed at making mailbox resynchronization more efficient does not include the QRESYNC extension [30], especially given that its implementation does not impose much in terms of additional requirements on top the already-supported CONDSTORE.

The iOS also notably does not use the IDLE command at all. The reason, according to a message allegedly sent by Steve Jobs [137], is that is is *"a power hungry standard"*. Systematic measurements [138] [139] and experience alike [3] shows, however, that the mere act of having a TCP connection open with an occasional keep alive "pings" being transfered have no significant impact on battery life on other platforms.

5.2.2 Android's Native E-mail Client

There is not much to be said about Android's native client's IMAP performance — the stock client does not offer push notification through IDLE [23] and the list of extension

[3] Mark Crispin's famous *"I have built on-demand networks which shut down until signaled back on, and then happily resumed all the active TCP sessions even though the "network connection" had been powered off for days."* [140].

identifiers referenced from the application's source [4] only references the NAMESPACE, UIDPLUS and STARTTLS capabilities. None of the extensions which try to improve synchronization performance (the ESEARCH, CONDSTORE and QRESYNC) are available. No provisions for Lemonade's family are present at all. The LITERAL+, an extension which takes literally no effort for clients to support and removes one network round trip when uploading messages, is not supported.

Further analysis shows that the code is blocking and incapable of issuing or processing requests in parallel. These observations are consistent with what users generally describe as a "slow" experience. This might not come surprising given that Google would likely prefer its users to choose Google's own e-mail offering, the GMail, over various private IMAP accounts for business reasons.

5.2.3 Android's K-9 Mail

The Android's K-9 mail [141] is a fork of the original e-mail application from Google. The developers have managed to add support for two extensions, namely the IDLE and the COMPRESS=DEFLATE. More advanced features like the ESEARCH / CONDSTORE / QRESYNC are however still missing.

Furthermore, comments like *"TODO Need to start keeping track of UIDVALIDITY"* [5] might make one feel nervous about the safety of the data being accessed.

5.2.4 Modest / Tinymail

The several years old Nokia N900 shipped with a mail client called Modest, an application based on top of the Tinymail framework [142]. The underlying library supports an efficient mailbox synchronization through QRESYNC and the sources contain a reference to the CONVERT extension — however, this functionality appears to be limited and in an early state. The "Lemonade trio" for forwarding without download is not supported.

The code is written using the synchronous idioms. No support for command pipelining is present.

5.2.5 Nokia's Qt Messaging Framework

Nokia's Qt Messaging Framework [143] powers the e-mail functionality in the MeeGo Harmattan line of phones, the N9 and the N950. Support for extensions appears to be rich — the Lemonade trio is supported in its entirety, [6] as well as are the COMPRESS=DEFLATE, BINARY, IDLE, CHILDREN and many others.

The underlying IMAP library exhibits regular "batched synchronization" mode of operation; connections to mailboxes are typically not left alone for long, but the code makes aggressive use of the QRESYNC extension to benefit from extremely cheap mailbox synchronization. Code is written with memory-constrained devices in mind, care is taken to prevent data copying if possible. The general behavior focuses on making the specified subset of messages always available on device — there is no support for server-side threading or sorting, but some form of a server-side search *is* available.

[4]File `src/com/android/email/mail/store/ImapConnection.java` from Android's `platform/packages/apps/Email` repository as of the `android-4.1.1_r3-35-g01c55fd` revision.

[5]File `src/com/fsck/k9/mail/store/ImapStore.java`, line 103 as of the Git revision `3.512-1249-g5ce0e19` of the K-9's source code repository.

[6]An interesting detail is that the BURL extension is disabled if the account is configured to use IMAP over an SSL port instead of the STARTTLS; this is caused by a deficiency in the IMAP-URL standard [144].

Live updates through CONTEXT=SEARCH are not employed. Support for the LIST-STATUS is not available.

An interesting extension which is supported by the QMF is Google's XLIST [145] extension. However, the code [7] appears to only use the
Inbox mailbox flag for forcing case-insensitive mailbox comparison, a mechanism already mandated by the baseline IMAP protocol [2, p. 17].

All in all, the Qt Messaging Framework is a promising library with support for other protocols besides IMAP. Its IMAP implementation feels solid and is reasonably well covered by the unit tests.

5.2.6 Trojitá

Trojitá [146] is an advanced IMAP client which includes support for many different extensions from the basic ones like IDLE, LITERAL+ or the ID extension to the complex ones like the Lemonade trio or the ESEARCH / CONDSTORE / QRESYNC triplet. Trojitá can scale up to company-wide ERP-level e-mail processing (Appendix B.1.2, p. 94) while still using the same code base of the underlying IMAP library as the desktop and cell phone version.

5.3 Evaluating Extensions

As the previous section shows, the level of support for various extensions and their real-world deployment varies wildly. Many today's clients are using only a limited subset of features which are generally available, leaving opportunities for new applications entering the market to offer smoother, more efficient user experience.

Previous efforts aimed at standardizing a set of extensions to serve as a "mobile profile" of IMAP have not delivered a universally acceptable result. Lemonade, a specification universally considered to be *the* specification to go when evaluating clients and servers alike [147], mandates a set of extensions which — to the best of my knowledge — is not available on any single IMAP server in its entirety. Furthermore, the clients do not even attempt to use these extensions, probably due to not being able to verify them for interoperability and because of concerns about shipping something which could very well be considered a dead code.

The rest of this chapter tries to look at the available extensions through the optics of a client developer who is trying to push the IMAP server developers slightly, but not too much — pushing forward is crucial in attaining a sustainable development, yet applying too much pressure often leads to the whole movement getting stuck on a first obstacle. I make a few assumptions about the features which the client support; most notably, the client should have a persistent cache to store already downloaded data. The client is also expected to serve a human user who will work with the application for many days across a network which might fail occasionally or even very often.

This humble taxonomy is not an attempt to define a *level of support* — entire working groups dedicated to this task have failed to deliver a universally-accepted solution despite spending many years on this goal. Instead, it is intended to serve as a reference suggesting client and server developers alike what extensions might be useful.

Further details about the mentioned extensions are available in chapters three (p. 16 and four (p. 37).

[7] File src/plugins/messageservices/imap/imapprotocol.cpp as of QMF's Git revision 2011W26_2-263-gec26531.

5.3.1 The Bare Minimum

The first group of extensions lists those proposals which I consider to be a practical requirement for a well-behaving client to implement. Furthermore, their implementation does not impose an overly significant burden on the servers, neither in terms of complicated code and therefore increased development costs, nor in the runtime overhead.

LITERAL+

Implementing the LITERAL+ extension actually *reduces* the amount of special code which the client executes when transmitting data using literals. As such, there is no excuse for not implementing this extension.

IDLE

Entering the IDLE mode enables the server to issue any untagged response, including the EXPUNGE and VANISHED, at any time. Because the IMAP clients are already required to handle any response at any time, actually making use of this feature should not impose any extra requirements on their implementors. The only acceptable reason for not using IDLE is, in my opinion, a technical deficiency in the platform's TCP stack and the related layers. If the total power consumption of the IDLE command exceed periodic polling, clients shall refrain from calling IDLE.

ID

The ID command provides useful diagnostics about the other end of the IMAP channel. Similar features exist in other Internet protocols. Hiding the name of one's implementation is a weak form of security measure — existing efforts [148] are able to recognize an IMAP server by its CAPABILITY response, a set of standard human-readable texts and through additional criteria and it is conceivable that an alternative for clients is feasible as well.

BINARY

The BINARY extension can delegate decoding of the content-transfer-encoding to the server side. Given that servers compliant with the baseline IMAP protocol already have to include a full MIME decoder, support for the BINARY extension is a firm candidate for the basic level of the extension taxonomy in spite of the fact that the most widely deployed IMAP server does not support it in its production releases yet.

UIDPLUS

Without the UIDPLUS extension, it is hard for clients to tell the UID of a message they have just appended to the mailbox. On servers which support persistent storage of UIDs, support for this extension does not add any further requirements.

CHILDREN, LIST-EXTENDED and LIST-STATUS

With the CHILDREN LIST extension, GUI clients do not have to explicitly issue one more LIST command for each mailbox shown in the GUI to tell whether said mailbox shall be drawn with a visual cue indicating its potentially expandable state. The LIST-EXTENDED and LIST-STATUS allow for bundling of additional metadata with the LIST responses and invoking the STATUS command, respectively.

ESEARCH

The ESEARCH response does not add any requirements on the IMAP servers besides the fact that they must send the whole response at once. I believe that this is a reasonable trade-off considering the benefits the compact `uidset` syntax can provide when working with large mailboxes.

COMPRESS=DEFLATE

Given the ubiquitous presence of the ZLib library, clients shall make sure that either the TLS compression is active or the COMPRESS=DEFLATE command is issued. IMAP is a textual protocol and as such compresses extremely well.

5.3.2 Useful Extensions

The second group of extensions consists of those which are very valuable for the increased efficiency of the protocol exchange, yet their implementation imposes a set of requirements on the server vendors, CPU resources or generally requires considerable effort to "get right".

CONDSTORE and QRESYNC

The CONDSTORE extension enables clients to skip downloading of all flags upon each mailbox reconnect. As such, it is highly desirable. It is not placed in the basic category due to its implications on the server side, though.

With server requirements not significantly different to the CONDSTORE, QRESYNC reduces the amount of data transferred when resynchronizing a mailbox after new messages have arrived and others have been expunged. Because it allows clients to skip the UID map rebuild, significant bandwidth savings can be obtained. As such, clients are highly recommended to support this extension.

ENABLE

The ENABLE extension is a requirement for QRESYNC. It does not add any additional requirements on clients or servers, so it shall belong to the same category.

MULTIAPPEND

The MULTIAPPEND command improves performance in clients upload many messages at once — a scenario common in applications which support batched import of existing e-mails. Clients which do not need this feature can safely ignore this extension. MULTIAPPEND also mandates that the whole operation is atomic, that is, either all of the messages are appended, or none for them are. For this reason, this extension is not put into the basic level.

SENDMAIL

Servers implementing the SENDMAIL extension as proposed in this thesis free their clients from having to speak the ESMTP protocol at all, a benefit bringing along streamlined end-user configuration and reducing support calls. However, due to a certain political pressure against the adoption of any extension allowing message submission via IMAP, I have decided to put SENDMAIL to the middle category.

CATENATE

If the client can make use of the `CATENATE` extension, it can be leveraged for tasks like stripping out an unwanted attachment from a message otherwise considered valuable. Its usefulness increases in combination with `SENDMAIL` or `BURL` where it allows for full support for the "forward without download" functionality and significant bandwidth reduction.

5.3.3 The Most Advanced Extensions

The last group contains those extensions which are either something *special*, exotic, or generally hard to implement correctly. Clients can benefit from many extensions from this group, yet one cannot expect to have them available on many IMAP servers due to their inherent complexity or increased amount of requirements which they put on the server.

SORT, SORT=DISPLAY and THREAD

These extensions present functionality which is very desirable from a client's point of view, yet requires the server to build an index of the whole mailbox contents at once. In absence of this extension, clients wishing to offer threading support or those that need to present the results to their users in a particular order have no choice but to download additional data for every single message in the mailbox.

INCTHREAD, CONTEXT=SEARCH and CONTEXT=SORT

This group builds upon the functionality provided by `SORT` and `THREAD` by enabling live updates to the results the client is interested in. In their absence, clients either have to repeat the server-side operation once again whenever a new message arrive, or defer to a purely client-side sorting or threading.

SEARCH=FUZZY

Adding support for "fuzzy search", this extension mandates servers to build an equivalent of a fulltext search index for the current mailbox, placing a significant burden on the IMAP server.

URLAUTH and BURL

This subgroup of extensions is known as the "Lemonade trio". Together, they allow for efficient operations like forward-without download, drastically reducing the amount of data to send when working with big messages. Due to the requirement for an explicit trust path between the IMAP server and the ESMTP submission service, clients have to anticipate problems when they try to utilize this functionality.

SPECIAL-USE and CREATE-SPECIAL-USE

A relatively new extension which allows clients to automatically obtain data about what mailbox serves a "special purpose" — for example, containing *all messages* from any other mailbox, or being designated as a "sent folder" by the system or account configuration. This could become a very useful extension, but it is too early to mandate its support now.

CONVERT

No servers that I am aware of deploy the CONVERT extension despite the time it has been around. No matter how useful I find this extension, this trait alone puts it firmly into the "speciality" group.

NOTIFY and MULTISEARCH

The NOTIFY builds where IDLE stopped, adding support for asynchronous notification about state of many mailboxes without a need of keeping concurrent connections opened. The MULTISEARCH command adds support for searches spanning several mailboxes, improving the traditional model where clients are required to "hop" mailboxes periodically. Without much support on the server side, these extensions probably cannot be relied upon in 2012.

Chapter 6

Trojitá's Architecture

This chapter provides a brief introduction to the architecture of Trojitá from a programmer's point of view. Additional information is provided in the documentation shipped as a part of the source tree.

6.1 Overview of Components

Trojitá makes heavy use of certain idioms common in Qt programming and in object-oriented software development in general.

At the highest layer lies the GUI, graphical code managing the actual user interaction. This code contains no knowledge of IMAP or any other e-mail protocol; it is simply a graphical presentation layer specific to the desktop version of Trojitá. In the releases intended for mobile platforms, the traditional `QWidget`-based GUI is replaced by a variant built on top of Qt's QML [149], a framework especially suited for touch-centric devices.

Any interaction with IMAP is initiated through the model-view framework [150] and related utilities. A core class encapsulating a representation of a single IMAP server, the `Model` class, is accompanied by various proxy models and other helpers to segregate and transform the data into a better shape suitable for the upper layers.

Any action which shall take effect is, however, not performed by any of the model-related classes. Trojitá utilizes the concept of *tasks*, a set of single-purpose classes each serving a distinct role. Examples of such tasks are "obtain a connection", "synchronize a mailbox", "download a message" or "update message flags".

One layer below the Tasks, the Parser is located. This class along with its support tools converts data between a byte stream arriving from or destined to the network and higher-level commands and responses which are utilized by the upper layers. Actual network I/O operations are handled through a thin wrapper around Qt's own `QIODevice` called `Streams`. [1]

6.1.1 Handling Input/Output

The raw byte streams provided by the network abstraction classes are intercepted by the `Parser` class. It takes any incoming octet sequence and with the help of a `LowLevelParser` instantiates the concrete subclasses of the `AbstractResponse` class. Actual parsing of the responses is typically deferred to the corresponding `Response`

[1] The `QIODevice` is wrapped to allow for transparent compression using the deflate algorithm. Due to the historic reasons, the `Stream` subclasses use the *has-a* instead of the *is-a* approach; this was required back when Trojitá shipped I/O implementations not based on the `QIODevice` class for evaluation purposes. It also helps to reduce the amount of functions each I/O implementation has to implement.

constructors which will tokenize the incoming data through LowLevelParser's methods and act on their contents according to the corresponding grammar rules.

The response parsing had to be substantially relaxed from its original state due to observed interoperability issues. Even the most popular IMAP implementations struggled with following the ABNF-defined [151] syntax to the letter; the most iconic example is Google's service which has prevented the IMAP clients talking to it from accessing forwarded messages for many years [11]. [2]

For some time, Trojitá's Parser was implemented in a separate thread in an attempt to improve performance. However, profiling showed that the amount of time spent in parsing was negligible compared to the rest of the application with the only exception being the THREAD response which essentially requires a complex transformation of nested lists. As such, the whole of Trojitá is now a single threaded application. [3]

6.1.2 The Concept of Tasks

The Tasks are designed to collaborate with each other, and there is a network of dependencies on how they can be used so that each task is responsible for only a subset of the overall work required to achieve a particular goal. For example, when a message shall be marked as read, an UpdateFlagTask is created. Because "marking message as read" is implemented by manipulating IMAP message flags, an action which can only happen while a mailbox is selected, the UpdateFlagsTask asks the Model to return an instance of a KeepMailboxOpenTask, a "manager" object which is responsible for keeping the mailbox state synchronized between the server and the client. [4] If the mailbox is already opened, an existing instance of the KeepMailboxOpenTask is returned; if that is not the case, a new instance is returned. The UpdateFlagsTask's execution is blocked and will commence only when (and if) the acquisition of a synchronized status succeeds. Similarly, this "maintaining" task (the KeepMailboxOpenTask itself deals just with incremental *updates* to the mailbox state and is activated only when the mailbox is opened. The actual mailbox synchronization is delegated to the ObtainSynchronizedMailboxTask. This synchronizer obtains the underlying connection handle through consultation with the Model to decide whether a new connection shall be established or an existing one re-purposed. This policy decision is completely contained in the Model and is not visible to other layers (or the tasks) at all.

Similar divisions of responsibility exist during other scenarios; the supporting infrastructure makes sure that no actions are started unless their prerequisites have succeeded.

The whole hierarchy is presented to the user in various shapes; the most basic and rudimentary one is a "busy indicator" showing whether any activity is taking place. A more detailed view showing an overview of all active tasks in a tree-like manner illustrating their dependencies is available (see the TaskPresentationModel class in Trojitá's sources).

The introduction of Tasks to Trojitá in 2010 proved to be an excellent decision allowing for much increased development pace and contributed to the much improved robustness and code clarity. Thanks to a proper use of git's branches, the transition was undertaken over a longer time pried without disturbing the ongoing application development.

[2]As of mid-2012, this issue remains unfixed despite my bug reports raised through both official and unofficial channels.

[3]The WebKit engine used in HTML rendering creates threads for its individual purposes; the similar behavior is observed in the QML engine used in the mobile version. These threads are not counted here, for they are considered to come from the "system libraries".

[4]The KeepMailboxOpenTask also serves as a dispatcher ensuring that the selected mailbox is switched only when any tasks which needed a particular mailbox open will have finished.

The Tasks are instantiated by an auxiliary factory class in order to better facilitate automated unit testing with mock objects.

6.1.3 Routing Responses

The Tasks introduced in the previous section are also used for proper response processing. At all times, the `Model` can access a list of "active tasks" which are assigned to a particular IMAP connection. When a response arrives, it is dispatched to these tasks in a well-defined order until either of the tasks declares the response as "handled". If no task claims responsibility for a particular response, the `Model` itself takes action. This action might be either regular processing, or, in case the `Model` cannot safely take care of said response, an error being raised and the connection sewered to prevent possible data loss due to suspected bug in the IMAP implementation.

The responses themselves are communicated to the `Model` (and, through it, to the responsible Tasks) through a queue of responses using Qt's own signal-and-slot mechanism. The same way of passing data is used for additional "meta information" like informing about connection errors, parser exceptions or the low-level SSL/TLS certificate properties. Having a unified queue of the incoming data made error handling much more elegant. [5] Care has been taken to optimize this parsed data passing to minimize the amount of copying and — in general — Qt's `QMetaObject` invocations while balancing the GUI responsiveness. The IMAP handlers now process the incoming responses in batches, pausing every so often to allow the GUI to keep up with other incoming events to prevent an appearance of the whole application "freezing" temporarily. The selected approach works well even on cell phones with limited resources.

Any responses passing through the queue are also logged into an in-memory ring buffer. Should an error occur, this buffer is used as a "black box" device which is shown to the user to give her a better context of what went wrong. This debug logging proved to be extremely valuable in debugging interoperability issues with non-compliant servers as well as when fixing bugs in Trojitá.

6.1.4 Models and Proxies

Historically, much of the IMAP logic in Trojitá has been concentrated in the `Model` class. Over the time, I've refactored that code to separate modules; however, even today, the `Model` handles both data publication through the Qt's model-view framework as well as IMAP response dispatch to the relevant Tasks. With little less than two thousands of physical lines of code, the `Model` remains the second-largest source file in the code base (the biggest file is a unit tests related to various modes of mailbox synchronization).

The `Model` itself exports a view containing anything available from a particular IMAP server account through the Qt's model-view API. The actual data is stored in a tree formed by various `TreeItem` subclasses.

As showing a single, rich-structured tree containing *everything* from mailboxes to individual message parts would not be particularly user-friendly, a number of so called proxy models were created. These models usually operate on Qt's `QModelIndex` level, but where profiling showed that a compelling speed increase would result from bypassing the `QVariant` abstraction, direct access through the underlying (and Trojitá-specific) `TreeItem` classes was used instead. This has notably affected the design of the `ThreadingMsgListModel` which is actually the biggest consumer of the CPU time when Trojitá opens a huge mailbox and activates message threading for the first time.

[5] Previously, matters were complicated by the way how QWidget-based UIs tend to deal with errors through dialog boxes which trigger nested event loops.

Proxies exist for performing various transformations and filtering of the available data; some of them (like the `MailboxModel` and `MsgListModel`) are generic enough and used in all of the desktop client, the mobile application and the single-purpose batch synchronizer (see Appendix B.1.2, p. 94) while others (like the `PrettyMsgListModel`) are exclusive to the traditional desktop GUI.

6.1.5 Lazy Loading and Cache

The model-view separation strictly followed throughout Trojitá proved to be very useful when leveraging the full potential of many advanced IMAP features. Because the individual message parts are accessible separately, Trojitá's native mode of operation supported the often-mentioned use case of "only download a tiny text accompanying the big photo attachment and then ask user whether to fetch the photo" without any effort. At the same time, this separation made certain tasks which would typically be considered trivial a bit more demanding — e.g. when forwarding a message, Trojitá has to explicitly retrieve two independent body parts, one for the headers and one for the message payload, and explicitly join them together. On the other hand, in most of the situations this separation brought benefits visible to the end user which trumped the minor, uncommon complications.

Any data once downloaded are kept in a persistent cache. This feature allows Trojitá to work extremely well under unfavorable network conditions. It also allows its users to access any data which were already known previously in an offline mode.

Several caching backends are shipped in Trojitá; some of them store data exclusively in memory and therefore are not *persistent* per se, others use SQLite [152] for actual data storage. Another mode which offloads "big data" storage to additional on-disk files is used by default. [6]

6.2 The Mobile Version

Trojitá also ships with a special version targeted for use on cell phones with touch screen such as Nokia's N9. The release was tested on the Nokia N950, a developer device which I obtained through Nokia's Community Device Programme (Appendix B.1.3, p. 94).

The touch-optimized version of Trojitá shares most of the code with the desktop version; in particular, none of the underlying IMAP code had to be modified. The changes were concentrated solely in the GUI layer which was completely rewritten using QML [149], Qt's declarative language for fluid user interface creation. A certain amount of C++ code was also required, mainly for seamless integration of the underlying data providers with the JavaScript-based QML data presentation.

I also had to extend some of the existing Qt classes with proper functions required for Trojitá. One of them was replacing QML's native `QDeclarativeWebView` component, an QML item encapsulating WebKit, the HTML renderer. This replacement was required because the stock version from upstream Qt did not support specifying a `QNetworkAccessManager` instance to use on a per-item basis [153]. In QML, the whole QML scene shares a single instance of the `QNetworkAccessManager` which is used for all sorts of data fetching. In Trojitá, such a behavior is unacceptable for security reasons because the actual message data are transferred through said manager, and the manager therefore has to implement a proper security policy denying

[6] Work on the structured cache backends was sponsored by the KWest GbmH. (Appendix B.1.1, p. 94).

any requests for external elements. [7] Another reason for this separation is to make sure that each QNetworkAccessManager can only access data originating from a single e-mail message, an enforcement crucial to prevent phishing attempts and maintain the confidentiality of the user's messages. There were also additional technical reasons for this extension, some of which were related to a documented requirement of QML's global instance adding provisions for future multithreaded access, a requirement which Trojitá's implementation explicitly does not guarantee for performance reasons.

Apart from the mentioned issues, the porting proved to be very easy, to the extent when the most limiting factor were the author's lack of familiarity with QML and this technology's relative immaturity and lack of existing components when compared to the "regular" Qt applications. [8] No design decisions of the application were hit as obstacles during the porting process.

Trojitá's asynchronous mode of operation where any data transfers are performed in the background, without blocking the user interface, was paramount in achieving a decent performance and smooth UI updates. The batched data processing mentioned in previous sections of this chapter is an example of a performance optimization which contributed to the excellent responsiveness of the cell phone version.

The mobile version is still not as mature as the regular desktop release; most importantly, it does not support sending e-mails for now. Proper platform integration should also be done in future. For these reasons, the application was submitted to the Ovi store, Nokia's "app store" product, as a *technical preview*. Even at its present state, however, the application is very useful for accessing user's e-mail on the go and for quick checking of whether something important has happened. Trojitá's support for many IMAP extensions which reduce the overall bandwidth consumption was also tremendously useful — many of them, like the CONDSTORE and QRESYNC, were designed with exactly this use case in mind and are also reasonably wide deployed among the publicly available IMAP server implementations.

6.3 Regression Testing

The IMAP protocol implementation developed as a part of Trojitá is covered by an extensive test suite verifying its behavior under varying conditions. All extensions supported by Trojitá which relate to e-mail retrieval are covered by the test suite.

Most of the testing works by replacing the connection to the IMAP server by a mock object capable of verifying that the transmitted data matches the developer's expectations and returning the sample responses back. This approach is self-contained, does not require any other software to test and therefore increases the chances of regressions getting caught before they made their way to a production release.

The following is an example on how an actual test might look like:

```
/** @short Test QRESYNC reporting changed flags */
void ImapModelObtainSynchronizedMailboxTest::testQresyncChangedFlags()
{
```

[7]If the e-mail rendered was able to access external data, e.g. fetch images from the Internet at will, the resulting HTTP transfers would compromise the user's privacy and reveal confidential information like presence and schedule of a real person accessing a particular e-mail account. Such tracing can be trivially implemented by including a unique identifier in the image URL, which is the reason why any decent MUA rejects such network requests by default.

[8]An example is the lack of a QSettings equivalent in QML; users are suggested to use raw SQL access through openDatabaseSync method which is indeed much more flexible, but also located several layers below the QSettings on an index of the out-of-box usability — clearly, writing custom SQL queries is more demanding than issuing a call to int i = QSettings().value("i").toInt().

```
// Trick the IMAP code into believing that the server supports QRESYNC
// The test environment can do that without the burden of explicitly
// faking the CAPABILITY response.
FakeCapabilitiesInjector injector(model);
injector.injectCapability("QRESYNC");

// Simulate the previous state of the mailbox
Imap::Mailbox::SyncState sync;
sync.setExists(3);
sync.setUidValidity(666);
sync.setUidNext(15);
sync.setHighestModSeq(33);
QList<uint> uidMap;
uidMap << 6 << 9 << 10;

// Store the simulated state in the persistent cache
model->cache()->setMailboxSyncState("a", sync);
model->cache()->setUidMapping("a", uidMap);
model->cache()->setMsgFlags("a", 6, QStringList() << "x");
model->cache()->setMsgFlags("a", 9, QStringList() << "y");
model->cache()->setMsgFlags("a", 10, QStringList() << "z");

// At this point, we can proceed with actual testing

// Initiate the activity
model->resyncMailbox(idxA);
// Check the I/O communication
cClient(t.mk("SELECT a (QRESYNC (666 33 (2 9)))\r\n"));
cServer("* 3 EXISTS\r\n"
        "* OK [UIDVALIDITY 666] .\r\n"
        "* OK [UIDNEXT 15] .\r\n"
        "* OK [HIGHESTMODSEQ 36] .\r\n"
        "* 2 FETCH (UID 9 FLAGS (x2 \\Seen))\r\n"
        );
cServer(t.last("OK selected\r\n"));
// Make sure that nothing else was transferred "over the network"
cEmpty();

// Verify that the persistent cache had been updated properly
sync.setHighestModSeq(36);
QCOMPARE(model->cache()->mailboxSyncState("a"), sync);
QCOMPARE(static_cast<int>(model->cache()->mailboxSyncState("a").exists()),
        uidMap.size());
QCOMPARE(model->cache()->uidMapping("a"), uidMap);
QCOMPARE(model->cache()->msgFlags("a", 6),
        QStringList() << "x");
QCOMPARE(model->cache()->msgFlags("a", 9),
        QStringList() << "\\Seen" << "x2");
QCOMPARE(model->cache()->msgFlags("a", 10),
        QStringList() << "z");
```

```
    // Make sure that we've picked up the changed flag
    QCOMPARE(idxA.data(Imap::Mailbox::RoleUnreadMessageCount).toInt(), 2);

    // Nothing else should be running
    justKeepTask();
}
```

Trojitá includes many of these individual test cases covering both regular operation as well as several pathological scenarios which would be tricky to hit in the real world. In addition to that, the automated tests also verify speed of mailbox synchronization with folders containing hundreds of thousands of messages, a scenario which might be rare in practice [9] but extremely useful for catching programmer mistakes like proposing an $O(n^2)$ algorithm [10] over an $O(n \log n)$ one — in spite of the fact that the overhead of the testing framework, especially when working with huge data transfers, as is indeed the case when synchronizing such a mailbox, is comparable to the actual code being profiled.

6.3.1 Scalability

I'm regularly testing Trojitá on mailboxes with hundreds of thousands of messages. Trojitá's unique design which guarantees that only the *required* data is transferred is paramount in achieving (and retaining) reasonable performance under varying workloads.

Trojitá respects all relevant recommendations about incremental retrieval of data. Unless in the so called "expensive mode" where Trojitá tries to save bandwidth at the cost of increased waiting time, an intelligent preload is implemented which will seamlessly ask for data which is likely to be required in future in a background preload operation.

6.3.2 Command Pipelining

Whenever possible, Trojitá issues the IMAP commands in parallel. [11] Unfortunately, many expensive operations invoked on the server side often requires mailbox-wide locks, at least in the current server's implementations. An example of such behavior are Dovecot's adaptive indexes [154] which are created on-demand, in response to actual client's requests. When Trojitá asks for threading on a big mailbox for the first time, Dovecot will block handling the threading request and building the required index, not responding to a query for metadata of messages, even though Trojitá has already issued a command requesting their delivery.

Under normal circumstances, though, pipelining is crucial for decent performance in presence of excess network round trip times, and this parallel delivery of commands works extremely well even in case where the servers are unable to fulfill the requests in parallel.

[9] The *Seznam.cz*, the most popular Czech free e-mail provider, imposes a limit of just 30,000 messages per account as of early 2012.

[10] It was necessary to locate a message in the mailbox by its UID. A conventional binary search algorithm was not applicable because the search could encounter "message placeholders" whose UID was not know, and therefore could not be compared to the reference value. In the end, measurements have shown that starting with binary search and falling back on a linear scan later in the process upon encountering the first undefined value works fast enough in all tested circumstances.

[11] The parallelization imposes a configurable limit on the number of concurrently active tasks so as not to overload the server.

6.3.3 Low-level optimization

Performance measurements were undertaken using Valgrind's [155] `callgrind` [156] tool while memory usage was tracked using Valgrind's `massif` [157]. It is hard to compare existing clients based on the "consumed memory" alone, especially when considering that certain desktop clients like KDE's Akonadi-based [158] products are split into several processes, many of which are handling both IMAP-related and unrelated operations. The memory profiling was therefore concentrated on making Trojitá's memory usage "reasonable" and for checking of obvious regressions.

Some of the results I have observed when profiling the code were initially rather surprising — for example, replacing a `QStringList` implementation by a `QSet<QString>` one actually *increased* the memory usage from 296 MB in a synthetic benchmark [12] to 385 MB, i.e. by roughly one third — a result caused by the `QSet`'s hash table memory overhead.

In the end, the total use in the mentioned benchmark was reduced to just 186 MB by explicit use of `QString`'s implicit sharing where the actual text data are saved in memory only once, using `QString`'s native reference counting. The results could be likely improved by moving the data sharing one level up to the level of the whole `QStringList` instead of the underlying elements.

More drastic reductions could be obtained by actively removing unused data from memory and reducing the amount of empty placeholders being initialized to "null" values in the `TreeItemMessage` instances, or, potentially, deferring their instantiations indefinitely till the corresponding messages come to the widget's viewport. However, no compelling reason to do so on the target platforms was observed so far.

[12]The test used was `ImapModelObtainSynchronizedMailboxTest::testFlagReSyncBenchmark` as of commit `cf92f5f9f8e929e1427877c0db470d8c59651d3b` (March 2012) with the number of messages in mailbox being increased to half a million. Measurements were performed on an `x86_64` Gentoo Linux machine running Qt 4.7.1.

Chapter 7

Conclusion

The goal of this thesis was to investigate the existing IMAP extensions, evaluate how well they contribute to the operation of a mobile e-mail client, and find out what areas could benefit from further optimization. All of these goals were fulfilled.

The "mobile client" I often speak about throughout this thesis is today a very powerful device, and one that is very, very different from what was considered a "mobile client" just a few years ago. An ordinary phone that people use on a daily basis has typically a 1 GHz CPU with a gigabyte of RAM and can transfer data at the rate of several tens of megabits per second. However, should these powerful resources be fully utilized, the battery life would drop to mere hours at best. With great power comes great responsibility, and in the context of mobile applications this responsibility directly translates to a need of eliminating data transfers and CPU-heavy client-side operations as much as possible. Thankfully, the IMAP protocol and its rich extension family provide ample opportunities for delegating a fair amount of processing to the server.

The available IMAP extensions range from simple tweaks of the protocol behavior and experimental facilities adding completely different features to IMAP all the way to the extensions which are hard to implement, but provide the protocol speakers with features or processes which would otherwise not be achievable, or prove to be needlessly hard to attain. Certain features, formerly designed to accommodate hardware designs which are ancient through today's optics (like the support for command pipelining which was meant to allow the computer operators to physically locate a tape in their archive and mount that on the user's behalf), are today crucial for sustaining high performance over cellular networks with rather big round-trip times.

I believe that one cannot really evaluate a proposal without getting a perfect grasp of the changes it introduces. This is why I implemented most of the available extensions in Trojitá, my free software project dedicated to building a usable, fast and lightweight IMAP e-mail client. Functions which remained cumbersome or still required substantial data transfers even after the support for the existing extensions was added proved to be excellent candidates for proposing concrete enhancements.

I have selected three areas in which the IMAP protocol can be still improved. These areas do not have much in common. It was my intention to demonstrate that even more than twenty years after the IMAP protocol was conceived, it can be still built upon and modified to better suit today's needs, and that these opportunities can be discovered on multiple fronts.

As an example of a rather low-hanging fruit, I have proposed an extension which fixes a possible race condition in the QRESYNC extension. This bug illustrates the process of the protocol design pretty well — despite a concerted effort of many people involved in drafting and reviewing of the proposal, bugs can still creep in.

A second extension which I have included in this thesis optimizes the behavior of an online client. By building on top of three separate extensions which have been around for years, it allows an IMAP client to obtain threading information for messages in an incremental manner, preventing downloads of huge amount of data over and over again.

My last proposal deals with a relatively hot topic in the IMAP sphere — it adds a support for message submission to happen over IMAP. The question about whether this approach is the correct one has historically divided the protocol experts and software vendors into two hostile camps, one with cheering proponents, the other holding vigorous critics. People have tried to achieve a similar result by various means, but so far nothing as simple as the proposed `UID SENDMAIL` has ever got enough traction.

All of the extensions which I have designed were submitted for review to the expert group through mailing lists related to the IMAP protocol and its extensions. Especially with the last proposal, an interesting discussion has been sprouted. It will be interesting to watch the fate of the proposals as they enter the IETF standardization process. With luck and enough patience, they might even become an RFC one day.

The Trojitá application has received a substantial amount of attention throughout my work on this thesis. The project has grown almost three times in its overall size since my bachelor thesis on this subject. I was happy to receive contributions from people all over the world. Two companies expressed their interest in basing their commercial offering on top of Trojitá; I was hired to perform a contracted work on Trojitá on their behalf. A completely new version of the user interface optimized for touch-controlled cell phones has been developed and — thanks to a generous donation from Nokia — ported to the MeeGo line of smartphones. Interoperability testing revealed quite a few bugs, both in Trojitá as well as in production releases of IMAP servers from many software vendors, opensource and proprietary variants alike.

7.1 Future Work

This thesis represents the result of my ongoing involvement with IMAP and e-mail related topics in general. It is my intention to carry on this work in future.

The documents I proposed will have to be maintained. People known in the IMAP world have already raised their questions and provided valuable comments on the functionality of the extensions, some of them expressed explicit interest and willingness to implement them in their own products. I will do my best to shepherd the drafts through their long journey towards possible acceptance. Even if this process fails and no RFC gets directly produced, the approach will still have been documented and available for future protocol engineers and developers.

The Trojitá as a free software project will certainly benefit from its increased visibility. There is a long list of features which might be added in future, be it support for signed or encrypted e-mails or pushing harder for implementation of more extensions among the IMAP servers. The mobile version could certainly be ported to other phone platforms as well. Targeting Android will definitely be a welcome improvement.

Some of the extensions which are mentioned in this thesis were not implemented in Trojitá for various reasons, typically due to the fact that I was not able to find any servers announcing their availability. This should change in future — some of the IMAP server vendors have already expressed their interest in implementing quite a few of these advanced features. Thanks to its internal structure, I am confident that Trojitá will be able to accommodate these new proposals as they arrive.

Appendix A

Proposed Internet Drafts

This chapter contains full text of the Internet Drafts which I have submitted for community consideration.

The Internet Drafts are written in a custom document format which is the reason why these documents are presented in a dedicated section rather than being included as regular chapters of this thesis.

A.1 draft-imap-qresync-arrived

Next seven pages contain a copy of `draft-imap-qresync-arrived`. This extension is introduced in section section 4.1 on page 37.

```
Internet Engineering Task Force                               J. Kundrat
Internet-Draft                                             July 25, 2012
Intended status: Standards Track
Expires: January 24, 2013
```

IMAP QRESYNC-ARRIVED Extension
draft-imap-qresync-arrived-01

Abstract

This document updates the QRESYNC extension of the IMAP protocol to use a new untagged response, the ARRIVED one, to inform about the UIDs of newly arriving messages. Deprecating the EXISTS response, this extension prevents a possible race condition where clients can lose synchronization of message UIDs in a selected mailbox when new arrivals are immediately expunged.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The QRESYNC extension [RFC5162] introduces the VANISHED response, a UID-based supplement to the sequence-number-based EXPUNGED [RFC3501]. The VANISHED response is also explicitly allowed to refer to non-existing message UIDs. Such a situation present a possible race condition where clients could lose track of the sequence -> UID mapping where new arrivals are removed in a parallel session.

This document updates the QRESYNC extension with a new response to be used instead of EXISTS, the ARRIVED one. It also adds a mechanism for activating this extension and explains backward compatibility concerns.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Race Condition in QRESYNC

This section illustrates an example where a fully-compliant QRESYNC client loses track of UIDs of some messages in its mailbox.

Suppose a client has opened a mailbox using the SELECT ... QRESYNC command. The mailbox contains a single message with UID 5; UIDNEXT is 11. The client is fully synchronized with the server. In this situation, the server informs client about new arrival:

```
                          S: * 3 EXISTS
```

Upon receiving the EXISTS reponse, IMAP client would typically proceed towards discovering various message metadata like theirs UIDs and flags, either through the UID SEARCH ALL command, or via UID FETCH (FLAGS). However, when the new arrivals are immediately expunged (perhaps as a result of an activity in a parallel session), the server can possibly process the request for additinal metadata only after the expunge has been reported to the client. The VANISHED response actually complicates matter in this situation:

```
S: * VANISHED 12:20
```

This response informs about permanent removal of messages with UIDs between 12 and 20, inclusively. Unfortunately, as the VANISHED response is explicitly allowed to reference non-existing UIDs, the client doesn't know whether this VANISHED response affects two newly arriving messages. Immediately after having received these two responses, the client cannot make any assumptions about the rest of the mailbox (i.e. messages with UIDs > 6). Specifically, any of the following can be true:

- The mailbox now contains only one message with UID 5
- The mailbox contains two messages; first of them has UID 5 and nothing is known about the other one
- The mailbox contains three messages; first of them has UID 5 and nothing is known about the rest of them

In absence of the QRESYNC-ARRIVED extension or when talking to servers not offering that, clients MUST be prepared to work around the described race condition. An example of such a behavior is issuing an explicit UID SEARCH command to rediscover the unclear portion of the sequence -> UID mapping:

```
S: * 3 EXISTS
S: * VANISHED 12:20
C: x UID SEARCH UID 11:*
S: * SEARCH
S: x OK Search completed
```

A single search operation is enough to recover the complete UID mapping, but it induces unnecessary round trip before a full mapping can be established again. In addition, clients would have to ignore any unsolicited FETCH responses not containing UID of the target message:

```
S: * 3 EXISTS
S: * 2 FETCH (FLAGS (foo))
S: * 3 FETCH (FLAGS (bar))
S: * VANISHED 12:20
C: x UID SEARCH UID 11:*
S: * SEARCH
S: x OK Search completed
```

Clients have no chance but to ignore any FETCH response which affects messages whose UID is not yet known.

This race condition can be avoided by switching to UID-based equivalent of EXISTS, the ARRIVED response. Alternatively, the same goal would be achieved if the QRESYNC specification mandated that VANISHED responses MUST NOT refer to UIDs which were not present in the mailbox at the time immediately before the VANISHED response was generated.

3. IMAP Protocol Changes

3.1. QRESYNC-ARRIVED Parameter to SELECT/EXAMINE

The QRESYNC-ARRIVED parameter to SELECT/EXAMINE commands shares syntax with the QRESYNC parameter defined in [RFC5162]; the only exception is that it is identified by the "QRESYNC-ARRIVED" atom.

For backward compatibility, IMAP servers supporting this extension MUST support QRESYNC as defined by [RFC5162]. In addition, such servers SHOULD NOT sent VANISHED responses containing UIDs of messages which were expunged before, and SHOULD NOT send the untagged EXPUNGED instead of VANISHED.

3.2. ARRIVED Response

Contents: ordered list of UIDs

The ARRIVED response reports that the specified UIDs have been delivered to the mailbox. It is similar in functionality to the EXISTS response [RFC3501]; however, it can return information about multiple messages, and it returns UIDs instead of message numbers. The first benefit saves bandwidth, while the second elliminates the potential of client losing track of assigned UIDs in a mailbox.

If the client has opened the mailbox with the QRESYNC-ARRIVED parameter to SELECT/EXAMINE command, the EXISTS response MUST be still sent during the initial mailbox synchronization, i.e. between having received the SELECT or EXAMINE command and the corresponding tagged response, as in [RFC3501]. Servers MAY send the EXISTS response several times, but they SHOULD send it only once to conserve resources. The number given in the EXISTS response MUST NOT shrink. The clients MUST treat the last occurence as the final data.

Servers MUST NOT issue ARRIVED prior to issuing a tagged OK for the SELECT/EXAMINE command to prevent uncertanities about the HIGHESTMODSEQ value. New arrivals since the last time the mailbox has been selected are reported using FETCH responses with embedded UID, as in regular QRESYNC.

After the mailbox has been selected, all further updates about new arrivals MUST use the ARRIVED response. If servers were allowed to fall back to untagged EXISTS, the possibility of race conditions would return.

UIDs sent in the ARRIVED response must be always presented in a sorted order, lowest UID first. The sequence MUST NOT contain duplicate UIDs. Servers SHOULD take advantage of the compression capabilities of the sequence-set syntax and use the range syntax if possible.

The ARRIVED response MUST NOT be sent unless the client has passed the QRESYNC-ARRIVED option to the last SELECT or EXAMINE command.

4. Acknowledgements

This text builds upon the QRESYNC IMAP extension [RFC5162].

5. IANA Considerations

IMAP4 capabilities are registered by publishing a standards track or IESG approved experimental RFC. The registry is currently located at:

http://www.iana.org/assignments/imap4-capabilities

This document defines the QRESYNC-ARRIVED IMAP capability. IANA will be asked to add this capability to the registry.

6. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234].

Non-terminals referenced but not defined below are as defined by [RFC3501], [RFC5162], or [RFC5234].

```
capability          =/ "QRESYNC-ARRIVED"

select-param        =/ "QRESYNC-ARRIVED" SP "(" uidvalidity SP
                       mod-sequence-value [SP known-uids]
                       [SP seq-match-data] ")"
                       ;; conforms to the generic select-param
                       ;; syntax defined in [IMAPABNF]

message-data        =/ arrived-resp

arrived-resp        = "ARRIVED" SP sorted-sequence-set

sorted-sequence-set = sequence-set
                    ;; sequence of UIDs, "*" is not allowed
                    ;; UIDs must be sorted, lowest first, no duplicates
```

7. Security Considerations

This extensions adds no functionality on top of what is already defined in the QRESYNC extension, and therefore we believe that no additional security implications have to be considered.

8. References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC5162] Melnikov, A., Cridland, D. and C. Wilson, "IMAP4 Extensions for Quick Mailbox Resynchronization", RFC 5162, March 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

Appendix A. Changelog

Appendix A.1. Changes in 00 since private-02

o Fixed ABNF syntax for the select-param

o Better abstract

o Clearer formatting

Appendix A.2. Changes in private-02 since private-01

o ENABLE is still required, otherwise clients have no way of activating VANISHED

Appendix A.3. Changes in private-01 sice private-00

o Clarified that UIDs presented in the ARRIVED response MUST be always presented in a sorted order

o Used a special non-terminal in the ABNF grammar

o Used correct format for FETCH responses

o Clarify that EXISTS must be sent at least once and SHOULD be sent only once

o Clarify that ARRIVED MUST NOT occur before the tagged response to SELECT/EXAMINE

o Clarify that arrivals since the last time are to be reported through usual UID FETCH, as in old QRESYNC

Author's Address

Jan Kundrat
Eledrova 558
Prague 181 00
CZ

Email: jkt@flaska.net

A.2 draft-imap-incthread

Next ten pages contain a copy of the `draft-imap-incthread`. The background of this extension is discussed in section section 4.2 on page 39.

Internet Engineering Task Force   J. Kundrat
Internet-Draft   July 25, 2012
Intended status: Standards Track
Expires: January 24, 2013

IMAP Extension for Incremental Threading (INCTHREAD)
draft-imap-incthread-00

Abstract

This document describes the INCTHREAD IMAP extension which enables clients to retrieve incremental updates of the mailbox threading. The extension repurposes the ESEARCH response for passing along the threading information and builds on top of Arnt Gulbrandsen's work on the INTHREAD search key. The UID THREAD command is also extended to allow activating this extension. Together, these changes make it possible for clients not to fetch the complete mailbox threading any time a new message arrives.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Online IMAP clients which want to conserve the required bandwidth and also show messages in threads for their users have an option to delegate the message threading to the IMAP server through mechanisms outlines in [RFC5256]. Using the UID SEARCH command, clients do not have to download the message headers (like the Message-Id, References and In-Reply-To), fetching a complete thread mapping of all messages in a mailbox instead.

Unfortunately, the savings in transferred data is significantly reduced when clients have to fetch the thread mapping over and over again, which is the case when a new message arrives. Even if the clients obtained knowledge of the relevant headers of new arrivals, these data alone are not sufficient to determine a proper place where to insert the newly arriving message. Furthermore, a single newly arriving message could potentially affect placement of many messages or even all of them in a pathological case due to joining of adjacent threads together. This issue prevents using approach similar to the CONTEXT=SEARCH and CONTEXT=SORT extensions [RFC5267] where only a position of the new arrival is communicated in an incremental manner.

This extension builds upon Arnt Gulbrandsen's work [I-D.ietf-morg-inthread] and reuses the INTHREAD search key defined in said draft. This search key is used to inform the server that the search conditions are to refer to all threads containing any messages which match the original search criteria. However, the untagged THREAD response does not contain any data about the position of the affected thread among other threads in the mailbox. Support for INTHREAD alone therefore does not relief the clients from performing additional operations due to missing information. If the affected threads were always placed at the logical end of the mailbox, copying older messages would yield results different from the complete THREAD command. Similarly, attempting to reuse the original thread position would possibly limit the usefulness of the REFS algorithm [I-D.ietf-morg-inthread] which sorts threads with "fresh messages" at the end of the view.

Because the THREAD response cannot transmit the position of the resulting thread relative to other threads in the mailbox, the ESEARCH response [RFC4731] is used and the UID THREAD command is extended to allow for specifying the return options in manner consistent with how SEARCH and UID SEARCH were modified. Finally, two new ESEARCH return option, the THREAD and the INCTHREAD, are defined.

These modifications together allow clients to delegate the threading operation completely to the server side without significantly increasing network traffic even on busy mailboxes.

1.1. Drawbacks and Alternatives

This extension can still transfer excessive amounts of data because it transfers complete threads instead of incremental difference updates. However, this approach allows for reusing the clients' and servers' existing facilities, both for parsing and response processing. In addition, unless the protocol mandated history tracking of the threading tree, a much intrusive and resource-demanding feature, the incremental updates would be only possible in case where the mailbox is currently selected. This extension affirms the decision about threading requests to remain on the client side, letting it use its policies about when to request full threading information and when to use the incremental updates.

No support for automated updates of the threading data in the sense of the CONTEXT extension [RFC5267] are defined at this point. This might change based on feedback from other server and client vendors.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. IMAP Protocol Changes

2.1. New SEARCH Keys

This document reuses the INTHREAD search key from [I-D.ietf-morg-inthread] with no changes in meaning or semantics. Please note the difference between the INTHREAD and INCTHREAD ESEARCH return options.

2.2. Modified IMAP Commands

2.2.1. Modified UID THREAD Command

The UID THREAD command is extended with the following two return options:

THREAD Return the threading information through the ESEARCH response using syntax similar to the untagged THREAD response.

INCTHREAD Return a dedicated INCTHREAD section for each thread found in the result set.

The THREAD and INCTHREAD return options are mutually exclusive. Servers MUST return a tagged BAD response if a client specifies both return options in a single UID THREAD command.

Servers MUST use the ESEARCH response instead of untagged THREAD response when responding to the extended form of the UID THREAD command.

The THREAD and UID THREAD are two distinct commands. Because clients are not expected to rely on transient identifiers like the message sequence numbers for threading retrieval and storage and because of the requirement of using UIDs in the INCTHREAD response, no modification to the THREAD command is defined. This might change in future iterations of this draft if client authors expects sufficient interest.

2.3. ESEARCH Extension

Servers announcing the INCTHREAD capability support two new search return options:

THREAD Method for conveying the threading information in form similar to the THREAD untagged response.

INCTHREAD Response contains the threading information along with the specification of a UID of the previous thread root.

2.3.1. The THREAD ESEARCH Return Value

The THREAD return value uses the same format as the threading data in the untagged THREAD responses [RFC5256]. This return value is defined to allow clients to easily match data received over network with the tag of the command which caused them, as per the usual ESEARCH rules.

2.3.2. The INCTHREAD ESEARCH Return Value

The INCTHREAD return value consists of "previous thread root UID" followed by "threading response containing a single thread". The UID of the previous thread root MUST be zero if the thread sorts first in the resulting list of threads.

A dedicated INCTHREAD record MUST be present for each thread contained in the result set.

2.3.2.1. Processing Incremental Threading Updates

At first, the client removes any messages referenced in the received INCTHREAD response from its thread mapping. This step is crucial to allow new arrivals joining previously independent threads together.

In the second step, the client extracts the threading information from the received INCTHREAD response. The threading data is parsed as in [RFC5256] and the newly formed thread is inserted just behind a thread whose root message has UID specified in the "uid" argument of the INCTHREAD response.

If the client supports incremental threading, the INCTHREAD responses MUST be processed in order. In particular, inserting the newly formed threads to a proper location shall happen immediately as the new thread is created as the subsequent responses COULD refer to the root message of the just-inserted thread. If the UID is said to be zero (0), it is interpreted specially to mean that the newly formed thread is the very first one among the list of threads in the mailbox.

If valid, the UID of the previous thread root message MUST refer to the previous thread in a mapping which contains all messages from a mailbox. In particular, no matter what additional searching criteria the client has used and therefore what messages are present in the mailbox, the previous thread MUST always be identified without any search criteria being applied. In future, this might be changed to allow for a subsequent search specification to accommodate clients working on a subset of messages.

Servers MUST NOT send a number which does not refer to any thread root UID unless the number is 0 to indicate the very first thread.

Clients MUST deal with servers sending a UID which does not refer to any thread root or any message in the mailbox. Is is implementation-defined at which position such thread shall be inserted, but the thread MUST appear in the list of threads.

2.4. New Capabilities

This document adds two new IMAP capabilities, the ETHREAD and INCTHREAD.

Servers announcing the ETHREAD capability support the extended UID THREAD command syntax and the THREAD return option.

Servers supporting the INCTHREAD capability MUST support and announce the ETHREAD capability as well.

3. Examples

This section contains a few examples to illustrate how the INCTHREAD extension operates.

3.1. General Mode of Operation

Using the proposed extension, a typical communication between two compliant IMAP protocol speakers might look like the following:

```
S: * 666 EXISTS
C: x1 UID FETCH 665:* (FLAGS)
S: * 666 FETCH (UID 1666 FLAGS ())
S: x1 OK fetched
C: x2 UID THREAD RETURN (INCTHREAD) REFS utf-8
      INTHREAD REFS 666
S: * ESEARCH (TAG "x2") UID INCTHREAD 400
      (600 601 (640 666)(602 603))
S: x2 OK sent
```

The actual resulting message threading looks like the following:

```
   ...
    |
    | (All preceding threads are simply skipped.)
    |
    +-- 400
    |    +-- ...
    | (No data for the previous thread besides its root node is sent.)
    |
    +-- 600
    |    +-- 601
    |         +-- 640
    |         |    +-- 666  <-- the new arrival
    |         +-- 602
    |              +-- 603
    |
   ... (No data about any subsequent threads is included in the response.)
```

3.2. Inserting a Single Message

Consider the following threading for a mailbox:

```
         C: x1 UID THREAD (RETURN THREAD) REFS utf-8 ALL
         S: * ESEARCH (TAG "x2") UID THREAD (1)(2)(3)(4)
         S: x1 OK Threading sent
```

Such a response corresponds to the following threading:

```
                              +-- 1
                              +-- 2
                              +-- 3
                              +-- 4
```

A new message arrives and the client asks for the reading information:

```
   S: * 5 EXISTS
   S: * 5 FETCH (UID 5)
   C: x2 UID THREAD (RETURN INCTHREAD) REFS utf-8 INTHREAD REFS UID 5
   S: * ESEARCH (TAG "x2") UID INCTHREAD 2 (3 5)
   S: x2 OK Threading sent
```

The updated threading information should look like the following:

```
                            +-- 1
                            +-- 2
                            +-- 3
                            |   +-- 5
                            +-- 4
```

In this case, the thread with thread root with UID 4 is still considered "fresher" by the selected thread algorithm.

3.3. Joining Threads

The following example shows a more complicated scenario where independent threads are joined together. This illustrates the need for clients to remove the referenced messages from their thread mapping:

```
         C: x1 UID THREAD (RETURN THREAD) REFS utf-8 ALL
         S: * ESEARCH (TAG "x2") UID THREAD (1 2)(3 4)(5)
         S: x1 OK Threading sent
```

Such a response corresponds to the following threading:

```
                             +-- 1
                             |   +-- 2
                             +-- 3
                             |   +-- 4
                             +-- 5
```

A new response arrives, joining the first two threads in the mailbox together:

```
S: * 6 EXISTS
S: * 6 FETCH (UID 6)
C: x2 UID THREAD (RETURN INCTHREAD) REFS utf-8 INTHREAD REFS UID 6
S: * ESEARCH (TAG "x2") UID INCTHREAD 0 (6 (1 2)(3 4))
S: x2 OK Threading sent
```

The newly formed thread remains at the beginning of a mailbox:

```
 +-- 6           <-- new arrival
 |   +-- 1       <-- previous thread #1
 |   |   +-- 2
 |   +-- 3       <-- previous thread #2
 |       +-- 4
 +-- 5           <-- previous thread #3 remains a standalone thread
```

4. Acknowledgements

This extension builds upon the SEARCH=INTHREAD extension [I-D.ietf-morg-inthread] and the THREAD extension [RFC5256].

5. IANA Considerations

IMAP4 capabilities are registered by publishing a standards track or IESG approved experimental RFC. The registry is currently located at:

http://www.iana.org/assignments/imap4-capabilities

This document defines the ETHREAD and INCTHREAD IMAP capabilities. IANA will be asked to add these capability to the registry.

6. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234].

Non-terminals referenced but not defined below are as defined by [RFC3501], [RFC4466], [RFC4731], or [RFC5256].

```
  capability          =/ "ETHREAD" / "INCTHREAD"
      ;; <capability> from [RFC3501]

  modifier-ethread    = "THREAD"

  modifier-incthread  = "INCTHREAD"

  thread-return-opt   = modifier-thread / modifier-incthread
      ;; Similar to <search-return-opt> from [RFC4466]

  ret-data-thread     = "THREAD" [SP 1*thread-list]
      ;; <thread-list> from [RFC5256]

  ret-data-incthread  = "INCTHREAD" SP uid SP thread-list
      ;; <uid> from [RFC3501]
      ;; <thread-list> from [RFC5256]

  search-return-data  =/ ret-data-thread / ret-data-incthread
      ;; <search-return-data> from [RFC4466]

  thread              =/ "UID" SP "THREAD" [thread-return-opts]
                         SP thread-alg SP search-criteria
      ;; <thread> and <thread-alg> from [RFC5256]
      ;; <search-criteria> from [RFC3501] as amended by
      ;;   [I-D.ietf-morg-inthread]
      ;;
      ;; The thread-return-opts MUST contain exactly one of
      ;;   modifier-thread or modifier-incthread

  thread-return-opts  = SP "RETURN" SP "(" [thread-return-opt
                        *(SP thread-return-opt)] ")"
      ;; similar to the <search-return-opts> from [RFC4466]
```

7. Security Considerations

This document is believed to not have any security implications besides those already implied by [RFC5256] and [I-D.ietf-morg-inthread].

8. References

8.1. Normative References

[I-D.ietf-morg-inthread]

Gulbrandsen, A. and a. them, "The IMAP SEARCH=INTHREAD and THREAD=REFS Extensions", Internet-Draft draft-ietf-morg-inthread-01, July 2010.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC4466] Melnikov, A. and C. Daboo, "Collected Extensions to IMAP4 ABNF", RFC 4466, April 2006.

[RFC4731] Melnikov, A. and D. Cridland, "IMAP4 Extension to SEARCH Command for Controlling What Kind of Information Is Returned", RFC 4731, November 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5256] Crispin, M. and K. Murchison, "Internet Message Access Protocol - SORT and THREAD Extensions", RFC 5256, June 2008.

8.2. Informative References

[RFC5267] Cridland, D. and C. King, "Contexts for IMAP4", RFC 5267, July 2008.

Author's Address

Jan Kundrat
Eledrova 558
Prague 181 00
CZ

Email: jkt@flaska.net

A.3 draft-imap-sendmail

Next ten pages contain a copy of `draft-imap-sendmail`. This extension is described in depth in section section 4.3 on page 41.

Internet Engineering Task Force   J. Kundrat
Internet-Draft   July 22, 2012
Intended status: Standards Track
Expires: January 21, 2013

IMAP SENDMAIL Extension
draft-imap-sendmail-01

Abstract

This document extends the IMAP protocol with a feature to submit e-mail messages for delivery.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Many proposals exist which aim at providing the forward-without-download feature where user can reuse parts of her message for composing and sending a new one. The CATENATE extension [RFC4469] adds a method for constructing an IMAP message from parts to be obtained purely on the server side; the LEMONADE extension [RFC5550] mandates full support for BURL [RFC4468] and URLAUTH [RFC4467]. Together with a properly configured IMAP and SMTP servers and with enough client support, these extensions allow for possibility to forward messages without a prior download.

This functionality puts a certain burden on clients -- each of them must be properly configured with two accounts instead of one. It is not enough to configure user's client for access to IMAP, one must also provide all details for a proper SMTP server if it's required to be able to send e-mails. In addition, both SMTP and IMAP servers have to be properly configured to allow an additional mode of authentication and authorization.

This extension drastically simplifies the client operation and minimizes configuration requirements on the server side. If combined with the existing CATENATE extension [RFC4469], it works at least as effectively as the Lemonade trio.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Mode of Operation

The SENDMAIL extension adds the UID SENDMAIL IMAP command which instructs the IMAP server to arrange for delivery of an already existing IMAP message. How this message is composed is outside of scope of this extension, but it is assumed that clients will often use the APPEND or APPEND ... CATENATE commands.

Upon receiving the SENDMAIL command, the IMAP server is asked for arranging the message submission. Clients MAY pass additional data in form of various options of the SENDMAIL command. The server checks the passed data and submission options, optionally performs sanity checks on the message contents, verifies against a local policy that the user is authorized for message submission, and if none of these checks fails, the server passes the message for final delivery. The delivery method is outside of scope of this document, but typical methods would be invoking the `sendmail` binary or passing the message to an ESMTP gateway.

3. IMAP Protocol Changes

This extension introduces one new IMAP command, a few capabilities and related response codes.

3.1. New IMAP Capabilities

3.1.1. The SENDMAIL Capability

Servers implementing this extension announce its presence through the SENDMAIL capability. If the server supports this extension but message submission is unconditionally disabled by a security policy or service configuration, this capability MUST NOT be announced.

3.1.2. The SENDMAIL= Capabilities Family

The SENDMAIL command MAY contain submission options. Servers supporting voluntary features MUST indicate so by including the appropriate strings in the CAPABILITY responses. All capabilities used for these purposes use the SENDMAIL= prefix.

3.1.2.1. SENDMAIL=DSN

If the server supports user control of generating the Delivery Status Notifications (DSN), it MUST announce the SENDMAIL=DSN capability. Clients MUST NOT attempt to control DSN options through the DSN submission option unless the server announced SENDMAIL=DSN.

3.2. Additional Response Codes

The following response codes are defined for communicating the reason why submission failed in a machine-readable way.

3.2.1. The POLICYDENIED Response Code

The POLICYDENIED response code SHOULD be used if the server rejects message submission as a result of a policy based decision which MAY take the message content, user's behavior and transaction history into account.

3.2.2. The SUBMISSIONRACE Response Code

The SUBMISSIONRACE response code MUST be sent in the tagged response if the client asks for submission of a message that is either not marked with the $SubmitPending keyword or marked with the $Submitted keyword.

3.3. UID SENDMAIL command

The UID SENDMAIL command submits a message for delivery.

Arguments:

- UID of message to be sent
- optional list of submission options

Responses: FETCH response with updated message flags

Result:

OK Message submitted for delivery

NO Submission failed

BAD Invalid commands or options

This command is only valid in the selected state.

The server MUST check its local policy configuration and verify that the authenticated user is allowed to submit messages. The decision MAY be based on the user's credentials, the message contents, past history of the user, or any other criteria the server deems relevant. The server SHOULD take into account any other local policies before it proceeds with further submission.

Clients MUST NOT submit a message which is either not marked with the $SubmitPending keyword from [RFC5550], or which is marked with the $Submitted keyword. Servers MUST reject such a command with a tagged NO bearing the SUBMISSIONRACE response code.

In the course of submission, servers SHOULD atomically add the $Submitted flag to the message, as described in LEMONADE [RFC5550]. This transition MAY be hidden from any IMAP session or it MAY be visible in all of them.

If the command succeeded, the message MUST be marked with the $Submitted keyword, the $SubmitPending keyword MUST be cleared and a FETCH response containing the message UID and its new flags MUST be sent.

If the command failed, the server MUST clear both the $Submitted or $SubmitPending keywords.

If the server supports CONDSTORE [RFC4551] or QRESYNC [RFC5172] extensions, any flag changes MUST obey the usual MODSEQ invariants.

Clients MUST be prepared to handle failing submission at any time. Servers MAY reject message submission for any reason.

The server MUST process all specified submission options. The server MUST respond with a tagged BAD if the client used unrecognized or unannounced submission option. If the server cannot honor a recognized and announced submission option, it MUST respond with a tagged NO with the POLICYDENIED response code and the message MUST NOT be submitted.

Servers MAY alter the message payload of the outgoing message in conformance with best current practice about Internet mail. Individual recipients MAY receive different versions of the message. In particular, servers MUST change message headers so that the identity of addresses present in the Bcc headers is not revealed to other recipients. This mode of operation is described in [RFC5321] and [RFC5322]. The copy stored on the IMAP server MUST NOT be altered by these modifications.

3.3.1. Submission options

The following submission options are defined by this extension:

3.3.1.1. FROM Submission Option

Syntax: one e-mail address

The FROM submission option corresponds to the FROM field of the SMTP envelope. If not present, its value MUST be inferred from the message payload.

It is an error if the FROM submission option is present more than once.

3.3.1.2. DSN Submission Option

Syntax: delivery status notice specification

The DSN submission option controls generating of delivery status notifications related to the currently submitted message. When not given, an implementation-defined default value MUST be used.

It is an error if the DSN submission option is present multiple times.

Clients MUST NOT specify the DSN submission option unless the server announces the SENDMAIL=DSN capability. Support for the SENDMAIL=DSN submission option is OPTIONAL.

The DSN specification is either NIL to deactivate DSNs altogether, or a parenthesized list of any of the following options:

SUCCESS requests generating DSNs upon successful delivery of a message

DELAY activates generating DSNs when delivery is delayed

FAILURE requests generating DSNs when the delivery fails

The order of DSN requests is not significant.

3.3.1.3. RECIPIENT Submission Option

Syntax: one e-mail address

The RECIPIENT submission option corresponds to the TO field of the SMTP envelope.

The RECIPIENT submission option MAY be present more than once. Servers MAY impose a limit on the number of recipients of a single message.

If the RECIPIENT submission option is present, servers MUST ignore any To, Cc and Bcc headers in the message payload when determining the list of recipients of this message. That is, the final list of recipients of the message MUST consist exactly of those recipients specified in the RECIPIENT submission options.

If the RECIPIENT submission option is missing, servers MUST infer its value from the message payload. For example, each address present in any of To, Cc and Bcc message headers SHOULD be present in the recipient list.

Servers MAY impose a local policy decision about always sending a copy of message to a certain address. This operation MUST NOT affect the user-specified list of recipients passed through the RECIPIENTS submission options.

Message submission MUST be atomic -- message is either submitted for delivery to all recipients, or it MUST NOT be submitted for delivery to anyone.

4. Example

This section contains an example of how message submission over IMAP works.

The following example shows how client should submit a message with UID 123 in the current mailbox for delivery. If the message is passed through SMTP, its From address in the SMTP envelope MUST be set to foo@example.org and it MUST be submitted for delivery to two recipients, the a@example.org and b@example.org. The DSN options are set to report about excess delays failures in message delivery.

```
          C: x UID SENDMAIL 123 (FROM "foo@example.org"
                  FROM "bar@example.org"
                  RECIPIENT "a@example.org"
                  RECIPIENT "b@example.org"
                  DSN (delay failure)
              )
          S: * 10 FETCH (UID 123 FLAGS ($Submitted))
          S: x OK Message passed to the sendmail binary
```

In the following example, a message is delivered to addresses specified in the message payload. No submission options are given, and therefore the From and Sender envelope items are inferred from the actual payload. The DSN options, if supported, are set to an implementation-defined default value.

```
          C: x UID SENDMAIL 123
          S: * 10 FETCH (UID 123 FLAGS ($Submitted))
          S: x OK Message passed to the sendmail binary
```

5. Acknowledgements

FIXME

6. IANA Considerations

IMAP4 capabilities are registered by publishing a standards track or IESG approved experimental RFC. The registry is currently located at:

http://www.iana.org/assignments/imap4-capabilities

This document defines the following list of IMAP capabilities. IANA will be asked to add them to the registry:

- o SENDMAIL

- o SENDMAIL=DSN

FIXME: response codes

7. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234].

Non-terminals referenced but not defined below are as defined by [RFC3501], or [RFC5234].

```
capability          =/ "SENDMAIL" / "SENDMAIL=DSN"

uid                 =/ "UID" SP sendmail

sendmail            = "SENDMAIL" SP uniqueid [SP submission-options]

submission-options  = "(" submission-option *( SP submission-option ) ")"

submission-option   = sub-option-from / sub-option-recipient
                      / sub-option-dsn

sub-option-from     = "FROM" SP one-email-addr
                      ;; MUST NOT be present more than once

sub-option-recipient= "RECIPIENT" SP one-email-addr
                      ;; MAY be present more than once

sub-option-dsn      = "DSN" SP ( NIL / dsn-spec )
                      ;; MUST NOT be present more than once

dsn-spec            = "(" dsn-spec-item *( SP dsn-spec-item ) ")"
                      ;; an individual dsn-spec-item MUST NOT
                      ;; be present more than once

dsn-spec-item       = "DELAY" / "FAILURE" / "SUCCESS"

one-email-addr      = string
                      ;; valid e-mail address as per [RFC5321]
```

8. Security Considerations

This extension introduces a way of allowing authenticated users to submit Internet mail. Servers supporting this extension SHOULD implement the same security measures as other SUBMISSION [RFC4409] servers open to users.

The redirect command from SIEVE [RFC5228] already requires some types of IMAP message stores to be able to generate outgoing mail. Security considerations for this extension are similar.

9. References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC4409] Gellens, R. and J. Klensin, "Message Submission for Mail", RFC 4409, April 2006.

[RFC4467] Crispin, M., "Internet Message Access Protocol (IMAP) - URLAUTH Extension", RFC 4467, May 2006.

[RFC4468] Newman, C., "Message Submission BURL Extension", RFC 4468, May 2006.

[RFC4469] Resnick, P., "Internet Message Access Protocol (IMAP) CATENATE Extension", RFC 4469, April 2006.

[RFC4551] Melnikov, A. and S. Hole, "IMAP Extension for Conditional STORE Operation or Quick Flag Changes Resynchronization", RFC 4551, June 2006.

[RFC5172] Varada, S., "Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol", RFC 5172, March 2008.

[RFC5228] Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language", RFC 5228, January 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

[RFC5550] Cridland, D., Melnikov, A. and S. Maes, "The Internet Email to Support Diverse Service Environments (Lemonade) Profile", RFC 5550, August 2009.

Appendix A. FIXME Items

What's got higher priority, SUBMISSIONRACE or POLICYDENIED? :)

IANA and the response codes

Appendix B. Changelog

Appendix B.1. Changes in 01 since 00

o Removed the superfluous SENDER submission option

o Mandating Bcc removal in conformance with RFC 5321 / RFC 5322

Author's Address

Jan Kundrat
Eledrova 558
Prague 181 00
CZ

Email: jkt@flaska.net

Appendix B

Acknowledgement

This appendix provides full reference about third party contributions to the Trojitá code base. It also mentions the commercial ecosystem built around this open-source project.

B.1 Commercial Applications

B.1.1 Partnership with KWest GmbH.

In early 2010, I was contracted [159] by a German embedded systems vendor, the KWest GmbH (lately acquired by Blaupunkt). That company was tasked by a German ISP for developing a tablet and were looking for an e-mail client to ship. Even though the tablet was not finished for reasons unrelated to Trojitá, but this collaboration significantly improved the feature set of Trojitá.

B.1.2 Collaboration with OpenMFG LLC, dba xTuple

Later in 2010, an American CRM [1] vendor, the OpenMFG LLC, dba xTuple, were on a search for a solution integrating customers' e-mail correspondence into their ERP database. I was contracted to add the required features to Trojitá. The project successfully concluded in early 2011 and shipped on time.

More details about the whole architecture are available on xTuple's product website [160].

B.1.3 Nokia Developer Participation

In 2011, my application for Nokia's Community Device Program was accepted. Nokia was kind enough to provide their MeeGo smartphone, the N950, on loan. I have used this opportunity to build a version of Trojitá optimized for this platform [161].

This application was also presented at the OpenMobility conference in Prague [162].

[1]Customer Resource Management

B.2 Third-party Contributions

Although I'm the principal author of the vast majority of code in Trojitá, the project is run in an open environment as a free software. This model has attracted quite a few developers — both individuals and from established software vendors — over the course of the project history. This section presents a complete overview of all of their contributions. It is sorted in a chronological order.

Justin J contributed improvements to the GUI.

Benson Tsai started the effort of providing an optimized user interface suitable for portable devices. Portions of his work remained unmerged due to technical issues in compatibility with the desktop version, but his patches were most inspiring.

Gil Moskowitz from *OpenMFG LLC, dba xTuple* contributed patches towards better PostgreSQL integration in the xTuple e-mail synchronizer shipped as part of Trojitá.

John Rogelstad from *OpenMFG LLC, dba xTuple* improved PostgreSQL interoperability through use of xTuple's own `XSqlQuery` classes and fixed a build failure on Windows.

Jiří Helebrant contributed GUI improvements and the Trojitá's logo and application icon. He is also the author of the web site design.

Jun Yang submitted a patch fixing a build failure under Visual Studio 2008. He also reported an interoperability problem with `STATUS` response parsing with servers not conforming to RFC 3501.

Andrew Brouwers is the author of the `.desktop` file which ships with Trojitá and of the initial version of the `.spec` file used for building RPMs.

Tomáš Kouba cleaned up the C++ code. He also reported build failures with older versions of Qt.

Mariusz Fik of *OpenSuSE* improved the `.spec` file. He also started using OpenSuSE's Open Build Service for building Trojitá.

Thomas Gahr from *Ludwig-Maximilians-Universität München* implemented a simple address book, improved the GUI and made sure that new arrivals are properly reported.

Shanti Bouchez added support for tagging e-mails with arbitrary keywords. She also improved SMTP interoperability and enabled SSL/TLS support in there.

Chase Douglas from *Canonical Ltd.* added a feature for hiding of already read messages from the message listing.

Wim Lewis fixed encoding of human-readable names in outgoing messages.

Thomas Lübking contributed a feature for raw IMAP searching, fixed quite a few QWidget issues and submitted many GUI improvements in general.

I'd like to use this opportunity to also extend my gratitude to all users who reported bugs or encouraged further development of Trojitá.

B.3 Use of Existing Libraries

Trojitá makes use of the following third party libraries:

The Qt framework is used throughout the code as Trojitá is a Qt application.

The Qt Messaging Framework provided code for wrapping the deflate compression algorithm in a Qt API. It is also used for low-level character set conversions and MIME encoding/decoding.

ZLib is used as a backend for actual deflate compression and decompression.

The KDE project's PIM libraries was used for low-level string manipulation, character set conversion and related operations.

The QwwSmtpClient library from Witold Wysota is used for speaking the SMTP protocol. Several fixes were applied on top of the original release.

ModelTest is a testing tool for verifying `QAbstractItemModel` invariants. It is shipped as part of the source tree for technical reasons.

Appendix C

The Attached CD

C.1 Contents of the CD

The attached CD is an integral part of this thesis. The contents of the top-level `trojita` directory is the following:

internet-drafts The Internet-Draft manuscripts with extensions proposed as a part of this thesis

src A compressed tarball containing the source code of the Trojitá IMAP client

thesis A printable PDF file with the text of this thesis

C.2 Build Instructions

In order to build Trojitá, please follow these instructions:

- Make sure that Qt 4.6 (or newer), including its development headers, is present on the machine
- If compiling on a Unix host, make sure that the `pkgconfig` and `zlib` are available as well
- Enter the directory with Trojitá's source code
- Execute `mkdir _build; cd _build`
- Run `qmake CONFIG+=debug ../trojita.pro` to configure the package
- Execute `make -j4` in order to build the application

After following these instructions, Trojitá can be launched by running: `./src/Gui/trojita`.

Bibliography

[1] Heidar Bernhardsson. IMAP is evil. Comment on a blog post (online), April 2012. URL http://blog.haraldkraft.de/2012/04/imap-is-evil/#comment-2156.

[2] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard), March 2003. URL http://www.ietf.org/rfc/rfc3501.txt. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186.

[3] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (Standard), May 1996. URL http://www.ietf.org/rfc/rfc1939.txt. Updated by RFCs 1957, 2449, 6186.

[4] Jan Kundrát. Trojita: A Qt IMAP Client. Bachelor thesis, May 2009. URL http://trojita.flaska.net/thesis.pdf.

[5] Mark Crispin. IMAP MOVE extension. Comments on the imap-protocol mailing list, June 2010. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2010-June/001144.html.

[6] Mark Crispin. IMAP MOVE extension. Comments on the imap-protocol mailing list, June 2010. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2010-June/001156.html.

[7] Bill Shannon. Reporting/detecting expunged messages. Comments on the imap-protocol mailing list, September 2006. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2006-September/000261.html.

[8] Mark Crispin. Childless noselect mailboxes. Comments on the imap-protocol mailing list, December 2009. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2009-December/001043.html.

[9] Jorma Kilpi and Pasi Lassila. Statistical analysis of RTT variability in GPRS and UMTS networks. Technical report, November 2005. URL http://www.netlab.tkk.fi/tutkimus/pannet/publ/rtt-report.pdf.

[10] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045 (Draft Standard), November 1996. URL http://www.ietf.org/rfc/rfc2045.txt. Updated by RFCs 2184, 2231, 5335, 6532.

[11] Jan Kundrát. GMail IMAP: returning BODYSTRUCTURE for embedded messages. Comments on the imap-protocol mailing list, May 2011. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2011-May/001413.html.

[12] P. Resnick. Internet Message Format. RFC 2822 (Proposed Standard), April 2001. URL http://www.ietf.org/rfc/rfc2822.txt. Obsoleted by RFC 5322, updated by RFCs 5335, 5336.

[13] K. Moore. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text. RFC 2047 (Draft Standard), November 1996. URL http://www.ietf.org/rfc/rfc2047.txt. Updated by RFCs 2184, 2231.

[14] J. Myers. IMAP4 non-synchronizing literals. RFC 2088 (Proposed Standard), January 1997. URL http://www.ietf.org/rfc/rfc2088.txt. Updated by RFC 4466.

[15] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), May 1996. URL http://www.ietf.org/rfc/rfc1951.txt.

[16] A. Gulbrandsen. The IMAP COMPRESS Extension. RFC 4978 (Proposed Standard), August 2007. URL http://www.ietf.org/rfc/rfc4978.txt.

[17] C. Newman. Using TLS with IMAP, POP3 and ACAP. RFC 2595 (Proposed Standard), June 1999. URL http://www.ietf.org/rfc/rfc2595.txt. Updated by RFC 4616.

[18] Jiří Peterka. Kauza DigiNotar, aneb: když certifikační autorita ztratí důvěru, September 2011. URL http://www.lupa.cz/clanky/kauza-diginotar-aneb-kdyz-certifikacni-autorita-ztrati-duveru/.

[19] Adam Langley. Public key pinning, May 2011. URL http://www.imperialviolet.org/2011/05/04/pinning.html.

[20] B. Leiba. IMAP4 IDLE command. RFC 2177 (Proposed Standard), June 1997. URL http://www.ietf.org/rfc/rfc2177.txt.

[21] Mark Crispin. IMAP Idle. Comments on the imap-protocol mailing list, March 2008. URL http://mailman2.u.washington.edu/pipermail/imap-uw/2008-March/001959.html.

[22] Timo Sirainen. iOS IMAP IDLE (Standard "Push Email") Deficiency, Explanation? Comments on the imap-protocol mailing list, October 2010. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2010-October/001311.html.

[23] Issue 23971: IMAP idle (PUSH email) is not supported. Android bug tracker. URL http://code.google.com/p/android/issues/detail?id=23971.

[24] Dave Cridland. RE: UID SEARCH responses. Comments on the IETF's imapext mailing list, August 2007. URL http://www.ietf.org/mail-archive/web/imapext/current/msg00458.html.

[25] A. Melnikov and D. Cridland. IMAP4 Extension to SEARCH Command for Controlling What Kind of Information Is Returned. RFC 4731 (Proposed Standard), November 2006. URL http://www.ietf.org/rfc/rfc4731.txt.

[26] Mark Crispin. UID SEARCH responses. Comments on the IETF's imapext mailing list, August 2007. URL http://www.ietf.org/mail-archive/web/imapext/current/msg00477.html.

[27] Alexey Melnikov. UID SEARCH responses. Comments on the IETF's imapext mailing list, August 2007. URL http://www.ietf.org/mail-archive/web/imapext/current/msg00472.html.

[28] Dave Cridland. UID SEARCH responses. Comments on the IETF's imapext mailing list, August 2007. URL http://www.ietf.org/mail-archive/web/imapext/current/msg00482.html.

[29] A. Melnikov and S. Hole. IMAP Extension for Conditional STORE Operation or Quick Flag Changes Resynchronization. RFC 4551 (Proposed Standard), June 2006. URL http://www.ietf.org/rfc/rfc4551.txt.

[30] A. Melnikov, D. Cridland, and C. Wilson. IMAP4 Extensions for Quick Mailbox Resynchronization. RFC 5162 (Proposed Standard), March 2008. URL http://www.ietf.org/rfc/rfc5162.txt.

[31] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Best Current Practice), March 1997. URL http://www.ietf.org/rfc/rfc2119.txt.

[32] Jan Kundrát. QRESYNC and new arrivals which get deleted immediately through VANISHED. Comments on the imap-protocol mailing list, June 2012. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2012-June/001781.html.

[33] A. Gulbrandsen, C. King, and A. Melnikov. The IMAP NOTIFY Extension. RFC 5465 (Proposed Standard), February 2009. URL http://www.ietf.org/rfc/rfc5465.txt.

[34] A. Gulbrandsen and A. Melnikov. The IMAP ENABLE Extension. RFC 5161 (Proposed Standard), March 2008. URL http://www.ietf.org/rfc/rfc5161.txt.

[35] Alfred Hoenes. Errata #1365 for RFC5162, March 2008. URL http://www.rfc-editor.org/errata_search.php?rfc=5162.

[36] Alexey Melnikov. QRESYNC and new arrivals which get deleted immediately through VANISHED. Comments on the imap-protocol mailing list, June 2012. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2012-June/001784.html.

[37] L. Nerenberg. IMAP4 Binary Content Extension. RFC 3516 (Proposed Standard), April 2003. URL http://www.ietf.org/rfc/rfc3516.txt. Updated by RFC 4466.

[38] A. Melnikov and P. Coates. Internet Message Access Protocol - CONVERT Extension. RFC 5259 (Proposed Standard), July 2008. URL http://www.ietf.org/rfc/rfc5259.txt.

[39] P. Resnick and C. Newman. IMAP Support for UTF-8. RFC 5738 (Experimental), March 2010. URL http://www.ietf.org/rfc/rfc5738.txt.

[40] M. Crispin and K. Murchison. Internet Message Access Protocol - SORT and THREAD Extensions. RFC 5256 (Proposed Standard), June 2008. URL http://www.ietf.org/rfc/rfc5256.txt. Updated by RFC 5957.

[41] D. Karp. Display-Based Address Sorting for the IMAP4 SORT Extension. RFC 5957 (Proposed Standard), July 2010. URL http://www.ietf.org/rfc/rfc5957.txt.

[42] Arnt Gulbrandsen. The IMAP SEARCH=INTHREAD and THREAD=REFS Extensions. Published IETF Internet Draft, July 2010. URL http://tools.ietf.org/html/draft-ietf-morg-inthread-01.

[43] D. Cridland and C. King. Contexts for IMAP4. RFC 5267 (Proposed Standard), July 2008. URL http://www.ietf.org/rfc/rfc5267.txt. Updated by RFC 5465.

[44] Does Gmail support all IMAP features? GMail's Support website, . URL http://support.google.com/mail/bin/answer.py?hl=en&answer=78761.

[45] Extension of the SEARCH command: X-GM-RAW. Google Developers — Google App Platform website, . URL https://developers.google.com/google-apps/gmail/imap_extensions#extension_of_the_search_command_x-gm-raw.

[46] T. Sirainen. IMAP4 Extension for Fuzzy Search. RFC 6203 (Proposed Standard), March 2011. URL http://www.ietf.org/rfc/rfc6203.txt.

[47] R. Gellens. IMAP Regular Expressions SEARCH Extension. Published IETF Internet Draft, March 2000. URL http://tools.ietf.org/html/draft-ietf-imapext-regex-00.

[48] Mark Crispin. "SCAN" capability ? Comments on the imap-protocol mailing list, May 2007. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2007-May/000535.html.

[49] B. Leiba and A. Melnikov. IMAP4 Multimailbox SEARCH Extension. RFC 6237 (Experimental), May 2011. URL http://www.ietf.org/rfc/rfc6237.txt.

[50] M. Gahrns and R. Cheng. The Internet Message Action Protocol (IMAP4) Child Mailbox Extension. RFC 3348 (Informational), July 2002. URL http://www.ietf.org/rfc/rfc3348.txt.

[51] A. Melnikov and T. Sirainen. IMAP4 Extension for Returning STATUS Information in Extended LIST. RFC 5819 (Proposed Standard), March 2010. URL http://www.ietf.org/rfc/rfc5819.txt.

[52] Michael J. Wener. IMAP Extension for CLEARIDLE. Published IETF Internet Draft, February 2005. URL http://tools.ietf.org/html/draft-wener-lemonade-clearidle-02.

[53] Arnt Gulbrandsen. The IMAP NOSTORE Extension. Published IETF Internet Draft, February 2006. URL http://tools.ietf.org/html/draft-gulbrandsen-imap-nostore-00.

[54] K. K. Tibanne and Mark Karpeles. IMAP4 IDLEPLUS extension. Published IETF Internet Draft, June 2010. URL http://tools.ietf.org/html/draft-magicaltux-imap4-idleplus-01.

[55] Arnt Gulbrandsen. On good and bad RFCs. Personal blog, March 2012. URL http://rant.gulbrandsen.priv.no/good-bad-rfc.

[56] Timo Sirainen. IMAP NOTIFY extension. The Dovecot mailing list, February 2012. URL http://dovecot.org/list/dovecot/2012-February/064031.html.

[57] Dovecot's source code in Mercurial, the dovecot-2.2-notify branch. Source code repository, July 2012. URL http://hg.dovecot.org/dovecot-2.2-notify/rev/201b097d5b58.

[58] M. Crispin. Internet Message Access Protocol (IMAP) - MULTIAPPEND Extension. RFC 3502 (Proposed Standard), March 2003. URL http://www.ietf.org/rfc/rfc3502.txt. Updated by RFCs 4466, 4469.

[59] P. Resnick. Internet Message Access Protocol (IMAP) CATENATE Extension. RFC 4469 (Proposed Standard), April 2006. URL http://www.ietf.org/rfc/rfc4469.txt. Updated by RFC 5550.

[60] R. Gellens and J. Klensin. Message Submission for Mail. RFC 6409 (Standard), November 2011. URL http://www.ietf.org/rfc/rfc6409.txt.

[61] C. Newman. Message Submission BURL Extension. RFC 4468 (Proposed Standard), May 2006. URL http://www.ietf.org/rfc/rfc4468.txt. Updated by RFC 5248.

[62] M. Crispin. Internet Message Access Protocol (IMAP) - URLAUTH Extension. RFC 4467 (Proposed Standard), May 2006. URL http://www.ietf.org/rfc/rfc4467.txt. Updated by RFCs 5092, 5550.

[63] Thiago Macieira and Don Sanders. [QTMOBILITY-1911] Sending any email fails after BURL command is sent. The Qt Bug Tracker, October 2011. URL https://bugreports.qt-project.org/browse/QTMOBILITY-1911.

[64] Mark Crispin. Re: [imap5] Where is IMAP5 ? Comments on the IETF's imap5 mailing list, July 2011. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00162.html.

[65] Dave Cridland. Re: [imap5] Where is IMAP5 ? Comments on the IETF's imap5 mailing list, July 2011. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00176.html.

[66] S. H. Maes, C. Kuang, R. Lima, R. Cromwell, E. Chiu, J. Day, R. Ahad, Wook-Hyun Jeong, Gustaf Rosell, J. Sini, Sung-Mu Son, Fan Xiaohui, Zhao Lijun, and D. Bennett. Push Extensions to the IMAP Protocol (P-IMAP). Published IETF Internet Draft, March 2006. URL http://tools.ietf.org/html/draft-maes-lemonade-p-imap-12.

[67] Bron Gondwana. Re: [imap5] Where is IMAP5 ? Comments on the IETF's imap5 mailing list, July 2011. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00163.html.

[68] Mark Crispin. Re: [imap5] Where is IMAP5 ? Comments on the IETF's imap5 mailing list, July 2011. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00177.html.

[69] A. Melnikov. IMAP4 POSTADDRESS extension. Published IETF Internet Draft, September 2010. URL http://tools.ietf.org/html/draft-melnikov-imap-postaddress-07.

[70] T. Showalter. IMAP4 ID extension. RFC 2971 (Proposed Standard), October 2000. URL http://www.ietf.org/rfc/rfc2971.txt.

[71] C. Newman, A. Gulbrandsen, and A. Melnikov. Internet Message Access Protocol Internationalization. RFC 5255 (Proposed Standard), June 2008. URL http://www.ietf.org/rfc/rfc5255.txt.

[72] A. Gulbrandsen. IMAP Response Codes. RFC 5530 (Proposed Standard), May 2009. URL http://www.ietf.org/rfc/rfc5530.txt.

[73] C. Newman, M. Duerst, and A. Gulbrandsen. Internet Application Protocol Collation Registry. RFC 4790 (Proposed Standard), March 2007. URL http://www.ietf.org/rfc/rfc4790.txt.

[74] M. Crispin. i;unicode-casemap - Simple Unicode Collation Algorithm. RFC 5051 (Proposed Standard), October 2007. URL http://www.ietf.org/rfc/rfc5051.txt.

[75] P. Resnick, C. Newman, and S. Shen. IMAP Support for UTF-8. Published IETF Internet Draft, June 2012. URL http://tools.ietf.org/html/draft-ietf-eai-5738bis-04.

[76] Jan Kundrát. Working with QDateTime's timezone information. The Qt-interest mailing list, June 2012. URL http://thread.gmane.org/gmane.comp.lib.qt.user/2225.

[77] M. Crispin. Distributed Electronic Mail Models in IMAP4. RFC 1733 (Informational), December 1994. URL http://www.ietf.org/rfc/rfc1733.txt.

[78] M. Gahrns. IMAP4 Multi-Accessed Mailbox Practice. RFC 2180 (Informational), July 1997. URL http://www.ietf.org/rfc/rfc2180.txt.

[79] B. Leiba. IMAP4 Implementation Recommendations. RFC 2683 (Informational), September 1999. URL http://www.ietf.org/rfc/rfc2683.txt.

[80] Mark Crispin. Ten Commandments of How to Write an IMAP client. URL http://www.washington.edu/imap/documentation/commndmt.txt.html.

[81] Timo Sirainen and Dave Cridland. IMAP Client Coding HOWTO. URL http://dovecot.org/imap-client-coding-howto.html.

[82] Best Practices for Implementing an IMAP Client. The IMAP Protocol Wiki. URL http://www.imapwiki.org/ClientImplementation.

[83] A. Melnikov. Internet Message Access Protocol (IMAP) UNSELECT command. RFC 3691 (Proposed Standard), February 2004. URL http://www.ietf.org/rfc/rfc3691.txt.

[84] M. Crispin. Internet Message Access Protocol (IMAP) - UIDPLUS extension. RFC 4315 (Proposed Standard), December 2005. URL http://www.ietf.org/rfc/rfc4315.txt.

[85] Jan Kundrát. [imapext] I-D ACTION:draft-ietf-imapmove-command-00.txt. Comments on the IETF's imapext mailing list, June 2012. URL http://www.ietf.org/mail-archive/web/imapext/current/msg04464.html.

[86] A. Melnikov and C. Daboo. Collected Extensions to IMAP4 ABNF. RFC 4466 (Proposed Standard), April 2006. URL http://www.ietf.org/rfc/rfc4466.txt. Updated by RFC 6237.

[87] A. Melnikov and D. Cridland. IMAP4 Keyword Registry. RFC 5788 (Proposed Standard), March 2010. URL http://www.ietf.org/rfc/rfc5788.txt.

[88] B. Leiba and A. Melnikov. Internet Message Access Protocol version 4 - LIST Command Extensions. RFC 5258 (Proposed Standard), June 2008. URL http://www.ietf.org/rfc/rfc5258.txt.

[89] D. Cridland, A. Melnikov, and S. Maes. The Internet Email to Support Diverse Service Environments (Lemonade) Profile. RFC 5550 (Proposed Standard), August 2009. URL http://www.ietf.org/rfc/rfc5550.txt.

[90] S. Maes and A. Melnikov. Internet Email to Support Diverse Service Environments (Lemonade) Profile. RFC 4550 (Proposed Standard), June 2006. URL http://www.ietf.org/rfc/rfc4550.txt. Obsoleted by RFC 5550.

[91] M. Gahrns. IMAP4 Login Referrals. RFC 2221 (Proposed Standard), October 1997. URL http://www.ietf.org/rfc/rfc2221.txt.

[92] M. Gahrns. IMAP4 Mailbox Referrals. RFC 2193 (Proposed Standard), September 1997. URL http://www.ietf.org/rfc/rfc2193.txt.

[93] Bug 59704 - support for IMAP Referrals (RFC2221, RFC2193), when server advertises MAILBOX-REFERRALS we should use RLIST and RLSUB (will help with MS Exchange and others). Mozilla Thuderbird bug tracker. URL https://bugzilla.mozilla.org/show_bug.cgi?id=59704.

[94] Rob Siemborski. A few MAILBOX-REFERRALS and NAMESPACE questions. Comments on the imap-protocol mailing list, January 2007. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2007-January/000360.html.

[95] M. Gahrns and C. Newman. IMAP4 Namespace. RFC 2342 (Proposed Standard), May 1998. URL http://www.ietf.org/rfc/rfc2342.txt. Updated by RFC 4466.

[96] A. Melnikov. IMAP4 Access Control List (ACL) Extension. RFC 4314 (Proposed Standard), December 2005. URL http://www.ietf.org/rfc/rfc4314.txt.

[97] J. Myers. IMAP4 ACL extension. RFC 2086 (Proposed Standard), January 1997. URL http://www.ietf.org/rfc/rfc2086.txt. Obsoleted by RFC 4314.

[98] J. Myers. IMAP4 QUOTA extension. RFC 2087 (Proposed Standard), January 1997. URL http://www.ietf.org/rfc/rfc2087.txt.

[99] J. Myers. IMAP4 Authentication Mechanisms. RFC 1731 (Proposed Standard), December 1994. URL http://www.ietf.org/rfc/rfc1731.txt.

[100] R. Siemborski and A. Gulbrandsen. IMAP Extension for Simple Authentication and Security Layer (SASL) Initial Client Response. RFC 4959 (Proposed Standard), September 2007. URL http://www.ietf.org/rfc/rfc4959.txt.

[101] C. Daboo. Use of SRV Records for Locating Email Submission/Access Services. RFC 6186 (Proposed Standard), March 2011. URL http://www.ietf.org/rfc/rfc6186.txt.

[102] C. Daboo and R. Gellens. Internet Message Access Protocol - ANNOTATE Extension. RFC 5257 (Experimental), June 2008. URL http://www.ietf.org/rfc/rfc5257.txt.

[103] C. Daboo. The IMAP METADATA Extension. RFC 5464 (Proposed Standard), February 2009. URL http://www.ietf.org/rfc/rfc5464.txt.

[104] E. Burger. WITHIN Search Extension to the IMAP Protocol. RFC 5032 (Proposed Standard), September 2007. URL http://www.ietf.org/rfc/rfc5032.txt.

[105] A. Melnikov. IMAP Extension for Referencing the Last SEARCH Result. RFC 5182 (Proposed Standard), March 2008. URL http://www.ietf.org/rfc/rfc5182.txt.

[106] A. Melnikov and C. King. IMAP4 Extension for Named Searches (Filters). RFC 5466 (Proposed Standard), February 2009. URL http://www.ietf.org/rfc/rfc5466.txt.

[107] A. Melnikov. Message Disposition Notification (MDN) profile for Internet Message Access Protocol (IMAP). RFC 3503 (Proposed Standard), March 2003. URL http://www.ietf.org/rfc/rfc3503.txt.

[108] N. Cook. Streaming Internet Messaging Attachments. RFC 5616 (Informational), August 2009. URL http://www.ietf.org/rfc/rfc5616.txt.

[109] M.R. Crispin. Interactive Mail Access Protocol: Version 2. RFC 1064, July 1988. URL http://www.ietf.org/rfc/rfc1064.txt. Obsoleted by RFCs 1176, 1203.

[110] M.R. Crispin. Interactive Mail Access Protocol: Version 2. RFC 1176 (Experimental), August 1990. URL http://www.ietf.org/rfc/rfc1176.txt.

[111] J. Rice. Interactive Mail Access Protocol: Version 3. RFC 1203 (Historic), February 1991. URL http://www.ietf.org/rfc/rfc1203.txt.

[112] M. Crispin. Internet Message Access Protocol - Version 4. RFC 1730 (Proposed Standard), December 1994. URL http://www.ietf.org/rfc/rfc1730.txt. Obsoleted by RFCs 2060, 2061.

[113] M. Crispin. IMAP4 Compatibility with IMAP2 and IMAP2bis. RFC 1732 (Informational), December 1994. URL http://www.ietf.org/rfc/rfc1732.txt.

[114] M. Crispin. Internet Message Access Protocol - Version 4rev1. RFC 2060 (Proposed Standard), December 1996. URL http://www.ietf.org/rfc/rfc2060.txt. Obsoleted by RFC 3501.

[115] M. Crispin. IMAP4 Compatibility with IMAP2bis. RFC 2061 (Informational), December 1996. URL http://www.ietf.org/rfc/rfc2061.txt.

[116] M. Crispin. Internet Message Access Protocol - Obsolete Syntax. RFC 2062 (Informational), December 1996. URL http://www.ietf.org/rfc/rfc2062.txt.

[117] J. Myers. IMAP4 UIDPLUS extension. RFC 2359 (Proposed Standard), June 1998. URL http://www.ietf.org/rfc/rfc2359.txt. Obsoleted by RFC 4315.

[118] The RFC Editor. Formatting RFCs. The RFC Editor homepage, November 2010. URL http://www.rfc-editor.org/formatting.html.

[119] S. Sherry and G. Meyer. Protocol Analysis for Triggered RIP. RFC 2092 (Informational), January 1997. URL http://www.ietf.org/rfc/rfc2092.txt.

[120] Jan Kundrát. IMAP QRESYNC-ARRIVED Extension. Comments on the imap-protocol mailing list, July 2012. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2012-July/001799.html.

[121] Jan Kundrát. [imapext] Draft for server-side incremental threading. Comments on the IETF's imapext mailing list, July 2012. URL http://www.ietf.org/mail-archive/web/imapext/current/msg04555.html.

[122] J. Klensin. Simple Mail Transfer Protocol. RFC 5321 (Draft Standard), October 2008. URL http://www.ietf.org/rfc/rfc5321.txt.

[123] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001. URL http://www.ietf.org/rfc/rfc2821.txt. Obsoleted by RFC 5321, updated by RFC 5336.

[124] Mozilla et al. Thunderbird:Autoconfiguration. Mozilla's wiki, online, May 2011. URL https://wiki.mozilla.org/Thunderbird:Autoconfiguration.

[125] Dave Crocker. [imap5] Beep. Comments on the IETF's imap5 mailing list, May 2012. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00488.html.

[126] Adrien de Croy. Re: [imap5] Designing a new replacement protocol for IMAP. Comments on the IETF's imap5 mailing list, February 2012. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00301.html.

[127] Giovanni Panozzo. Re: [imap5] Where is IMAP5 ? Comments on the IETF's imap5 mailing list, July 2011. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00192.html.

[128] R. Gellens. IMAP Submit Without Download. Published IETF Internet Draft, October 2003. URL http://tools.ietf.org/html/draft-ietf-lemonade-submit-01.

[129] N. Freed. SMTP Service Extension for Command Pipelining. RFC 2920 (Standard), September 2000. URL http://www.ietf.org/rfc/rfc2920.txt.

[130] Arnt Gulbrandsen. Re: Requesting comments on draft-melnikov-imap-postaddress-05.txt. Comments on the IETF's imapext mailing list, November 2006. URL http://www.ietf.org/mail-archive/web/imapext/current/msg00828.html.

[131] Bron Gondwana. Re: [imap5] Where is IMAP5 ? Comments on the IETF's imap5 mailing list, July 2011. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00148.html.

[132] Mark Crispin. Re: [imap5] Where is IMAP5 ?, July 2011. URL http://www.ietf.org/mail-archive/web/imap5/current/msg00164.html.

[133] Jan Kundrát. [imapext] Mail submission over IMAP. Comments on the IETF's imapext mailing list, July 2012. URL http://www.ietf.org/mail-archive/web/imapext/current/msg04518.html.

[134] Mike Abbott. Patch: support BURL. Apple's Opensource site (online), April 2010. URL http://www.opensource.apple.com/source/postfix/postfix-229/patches/burl.patch.

[135] Timo Sirainen. [IMAP Server Support Matrix] Specifications. The IMAP Protocol Wiki, June 2012. URL http://imapwiki.org/Specs.

[136] Isode Ltd. IMAP Enhancements in iPhone OS Update (iOS 4). Online, July 2010. URL http://isode.com/company/wordpress/imap-enhancements-in-iphone-os-update-ios-4/.

[137] Steve Jobs. Re: iOS IMAP IDLE (Standard P̈ush Email)̈ Deficiency, Explanation? Private message forwarded to the imap-protocol mailing list, October 2010. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/attachments/20101004/e96c54cf/iOSIMAPIDLEStandardPushEmailDeficiencyExplanation.eml.

[138] Henry Haverinen, Jonne Siren, and Pasi Eronen. Energy Consumption of Always-On Applications in WCDMA Networks. 2007. ISSN 1550-2252. doi: 10.1109/VETECS.2007.207. URL http://www.pasieronen.com/publications/haverinen_siren_eronen_vtc2007.pdf.

[139] Dave Cridland. [Imap-protocol] Fwd: iOS IMAP IDLE (Standard "Push Email") Deficiency, Explanation? Comments on the imap-protocol mailing list, October 2010. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2010-October/001303.html.

[140] Mark Crispin. [Imap-protocol] Fwd: iOS IMAP IDLE (Standard "Push Email") Deficiency, Explanation? Comments on the imap-protocol mailing list, October 2010. URL http://mailman2.u.washington.edu/pipermail/imap-protocol/2010-October/001298.html.

[141] k9mail. (Online), 2012. URL http://code.google.com/p/k9mail/.

[142] Tinymail. (Online), 2012. URL https://gitorious.org/tinymail#more.

[143] The Qt Messaging Framework. (Online), 2012. URL https://qt.gitorious.org/qt-labs/messagingframework.

[144] A. Melnikov and C. Newman. IMAP URL Scheme. RFC 5092 (Proposed Standard), November 2007. URL http://www.ietf.org/rfc/rfc5092.txt. Updated by RFC 5593.

[145] Extension of the list command: Xlist. Google Developers — Google App Platform website, . URL https://developers.google.com/google-apps/gmail/imap_extensions#extension_of_the_list_command_xlist.

[146] Jan Kundrát. Trojitá, a Qt IMAP e-mail client. (Online), 2012. URL http://trojita.flaska.net/.

[147] Isode Ltd. LEMONADE Profile: The Key Standard for Mobile Messaging. Online, June 2006. URL http://www.isode.com/whitepapers/lemonade-profile.html.

[148] Timo Sirainen. Open email survey. (Online), 2012. URL http://openemailsurvey.org/.

[149] The Qt Project. Introduction to the QML Language. Qt Documentation (online), 2012. URL http://qt-project.org/doc/qt-4.8/qdeclarativeintroduction.html.

[150] The Qt Project. The Interview Framework. Qt Documentation (online), 2012. URL http://qt-project.org/doc/qt-4.8/qt4-interview.html.

[151] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 5234 (Standard), January 2008. URL http://www.ietf.org/rfc/rfc5234.txt.

[152] The SQLite Developers. SQLite Home Page. Online, 2012. URL http://www.sqlite.org/.

[153] Jan Kundrát. Using different QNetworkAccessManager instances for multiple QML's WebView items. The Qt-interest mailing list, 2012. URL http://thread.gmane.org/gmane.comp.lib.qt.user/1807.

[154] Timo Sirainen. Dovecot's index files. Dovecot Wiki (online), March 2009. URL http://wiki2.dovecot.org/Design/Indexes.

[155] Valgrind Developers. Valgrind home. Online, 2012. URL http://valgrind.org/.

[156] Valgrind Developers. Callgrind: a call-graph generating cache and branch prediction profiler. Online, 2012. URL http://valgrind.org/docs/manual/cl-manual.html.

[157] Valgrind Developers. Massif: a heap profiler. Online, 2012. URL http://valgrind.org/docs/manual/ms-manual.html.

[158] The KDE Community. KDE PIM/Akonadi. The KDE Community wiki (online), 2012. URL http://community.kde.org/KDE_PIM/Akonadi.

[159] Jan Kundrát. KWest GmbH to Sponsor Trojita's Development. Personal blog, April 2010. URL http://jkt.flaska.net/blog/KWest_GmbH_to_Sponsor_Trojita_s_Development.html.

[160] OpenMFG LLC, dBa xTuple. xTuple Connect Email Integration. xTuple's Website, 2011. URL http://www.xtuple.org/emailIntegration/beta.

[161] Jan Kundrát. [Announce] Trojitá, a fast IMAP e-mail client. The MeeGo Forum, April 2012. URL http://forum.meego.com/showthread.php?t=5837.

[162] OpenMobility. Vyhlášení výsledků. Online, April 2012. URL http://www.openmobility.eu/konference/om2012/soutez/vyhlaseni-vysledku/.