Network Working Group   R. Austein
Internet Draft: IMAP4 Disconnected Access   Epilogue Technology
Document: internet-drafts/draft-ietf-imap-disc-00.txt   June 1994

SYNCHRONIZATION OPERATIONS FOR DISCONNECTED IMAP4 CLIENTS

Status of this Memo

This is a draft document of the IETF IMAP Working Group. A revised version of this draft document will be submitted to the RFC editor as an Informational RFC for the Internet Community. Discussion and suggestions for improvement are requested, and should be sent to imap@CAC.Washington.EDU. This document will expire before 31 November 1994.

This note attempts to address some of the issues involved in building a disconnected IMAP4 client. In particular, it deals with the issues of what might be called the "driver" portion of the synchronization tool: the portion of the code responsible for issuing the correct set of IMAP4 commands to synchronize the disconnected client in the way that is most likely to make the human who uses the disconnected client happy.

1. DESIGN PRINCIPLES

All mailbox state or content information stored on the disconnected client should be viewed strictly as a cache of the state of the server. The "master" state remains on the server, just as it would with an interactive IMAP4 client. The one exception to this rule is that information about the state of the disconnected client's cache remains on the disconnected client: that is, unlike the equivilent case in the DMSP protocol, the IMAP4 server is not responsible for remembering the state of the disconnected IMAP4 client.

We assume that a disconnected client is a client that, for whatever reason, wants to minimize the length of time that it is "on the phone" to the IMAP4 server. Often this will be because the client is using a dialup connection, possibly with very low bandwidth, but sometimes it might just be that the human is in a hurry to catch an airplane, or some other event beyond our control. Whatever the reason, we assume that we must make efficient use of the network connection, both in the usual sense (not generating spurious traffic) and in the sense that we would prefer not to have the connection sitting idle while the client and/or the server is performing strictly local computation or I/O. Another, perhaps simpler way of stating this is that we assume that network connections are "expensive".

Practical experience with existing disconnected mail systems (PCMAIL/DMSP) has shown that there is no single synchronization strategy that is appropriate for all cases. Different humans have different preferences, and the same human's preference will vary depending both on external circumstance (how much of a hurry the human is in today) and on the value that the human places on the messages being transfered. The point here is that there is no way that the synchronization program can guess exactly what the human wants to do, so the human will have to provide some guidance.

Taken together, the preceeding two principles lead to the conclusion that the synchronization program must make its decisions based on some kind of configuration file provided by the human, but almost certainly should not pause for I/O with the human during the middle of the synchronization process. The human will almost certainly have several different configurations for the synchronization program, for different circumstances.

Automated support for helping naive humans write better configuration files would be a good thing, but writing such tools is outside the scope of this discussion.

Since a disconnected client has no way of knowing what changes might

have occured to the mailbox while it was disconnected, message numbers are not useful to a disconnected client. All disconnected client operations should be performed using UIDs, so that the client can be sure that it and the server are talking about the same messages during the synchronization process. It is permissible, but probably not useful, for a disconnected client to use message numbers once it has obtained a valid current mapping between UIDs and message numbers.

2. OVERALL PICTURE OF SYNCHRONIZATION

The basic strategy for "normal" synchronization is pretty simple, and closely follows the strategy used by a disconnected DMSP client (terminology explained below):

a) Process any "actions" that were pending on the client;

b) Fetch the current list of "interesting" mailboxes;

c) For each mailbox, fetch the current "descriptors";

d) For each mailbox, fetch the bodies of any "interesting" messages that the client doesn't already have.

Explanation:

a) "Actions" are queued requests that were made by the human to the client's MUA software while the client was disconnected. Expected requests are commands like COPY, STORE, EXPUNGE, CREATE. FETCH commands may also show up as actions, if the MUA allows the human to explicitly fetch the body of a particular message that was skipped by the normal synchronization process. In general, any IMAP4 command can show up as an action, as can a few things that are not IMAP4 commands at all, such as requests to send a newly composed message via SMTP.

The list of actions should be ordered. Eg, if the human deletes message A1 in mailbox A, then expunges mailbox A, then deletes message A2 in mailbox A, the human will expect that message A1 is gone and that message A2 is still present but is now deleted.

By processing all the actions before proceeding with synchronization, we avoid having to compensate for the local MUA's changes to the server's state. That is, once we have processed all the pending actions, the steps that the client must take to synchronize itself will be the same no matter where the changes to

the server's state originated.

b) The set of "interesting" mailboxes pretty much has to be determined by the human. What mailboxes belong to this set may vary between different IMAP4 sessions with the same server, client, and human.

c) "Descriptors" is a DMSP term, borrowed here because it's both easier to use and more precise than a specific list of IMAP4 FETCH data items. Conceptually, a message's descriptor is that set of information that allows the synchronization program to decide what protocol actions are necessary to bring the local cache to the desired state for this message; since this decision is really up to the human, this information probably includes a at least a few header fields intended for human consumption. Exactly what will constitute a descriptor depends on the client implementation. At a minimum, the descriptor contains the message's UID and FLAGS. Other likely candidates are the RFC822.SIZE and BODYSTRUCTURE data items and the RFC-822 From:, To:, Date:, Subject:, and Message-ID: header lines.

Note that this step is also where the client finds out about changes to the flags of messages that the client already has in its local cache, as well as finding out about messages in the local cache that no longer exist on the server (ie, messages that have been expunged).

d) "interesting" messages are those messages that the synchronization program thinks the human wants to have cached locally, based on the configuration file and the data retrieved in step (c).

The rest of this discussion will focus primarily on the synchronization issues for a single mailbox.

3. DETAILS OF "NORMAL" SYNCHRONIZATION OF A SINGLE MAILBOX

The most common form of synchronization is where the human trusts the integrity of the client's copy of the state of a particular mailbox, and simply wants to bring the client's cache up to date so that it accurately reflects the mailbox's current state on the server.

Let <lastseen> represent the highest UID that the client knows about in this mailbox. Since UIDs are allocated in strictly ascending order, this is simply the UID of the last message in the mailbox that

the client knows about. Let <lastseen+1> represent <lastseen>'s UID plus one. Let <descriptors> represent a list consisting of all the FETCH data item items that the implementation considers to be part of the descriptor; at a minimum this is just the FLAGS data item.

With no further information, the client can issue issue the following two commands:

```
tag1 UID FETCH <lastseen+1>:* <descriptors>
tag2 UID FETCH 1:<lastseen> FLAGS
```

The order here is significant. We want the server to start returning the list of new message descriptors as fast as it can, so that the client can start issuing more FETCH commands, so we start out by asking for the descriptors of all the messages we know the client cannot possibly have cached yet. The second command fetches the information we need to determine what changes may have occurred to messages that the client already has cached. Once the client has issued these two commands, there's nothing more the client can do with this mailbox until the responses to the first command start arriving. A clever synchronization program might use this time to fetch its local cache state from disk, or start the process of synchronizing another mailbox.

Once the descriptors start arriving, the client can start issuing appropriate FETCH commands for "interesting" messages or bodyparts thereof. The decision on what is an "interesting" message is up to the client software and the human. One easy criterion that should probably be implemented in any client is whether the message is "too big" for automatic retrieval, where "too big" is a parameter defined in the client's configuration file.

It is important to note that fetching a message into the disconnected client's local cache does NOT imply that the human has (or even will) read the message. Thus, the synchronization program for a disconnected client should always be careful to use the .PEEK variants of the FETCH data items that implicitly set the \Seen flag.

Once the last descriptor has arrived and the last FETCH command has been issued, the client simply needs to process the incoming fetch items, using them to update the local message cache.

In order to avoid deadlock problems, the client must give processing of received messages priority over issuing new FETCH commands during this synchronization process. This may necessitate temporary local queuing of FETCH requests that cannot be issued without causing a deadlock. In order to achive the best use of the "expensive" network connection, the client will almost certainly need to pay careful attention to any flow-control information that it can obtain from the underlying transport connection (usually a TCP connection).

4. SPECIAL CASE: DESCRIPTOR-ONLY SYNCHRONIZATION

For some mailboxes, fetching the descriptors might be the entire synchronization step. Practical experience with DMSP has shown that a certain class of mailboxes (eg, "archival" mailboxes) are used primarily for long-term storage of important messages that the human wants to have instantly available on demand but does not want cluttering up the disconnected client's cache at any other time. Messages in this kind of mailbox would be fetched exclusively by explicit actions queued by the local MUA. Thus, the only synchronization that is necessary for a mailbox of this kind is fetching the descriptor information that the human will use to identify messages that should be explicitly fetched.

Special mailboxes that receive traffic from a high volume, low priority mailing list might also be in this catagory, at least when the human is in a hurry.

5. SPECIAL CASE: FAST NEW-ONLY SYNCHRONIZATION

In some cases the human might be in such a hurry that s/he doesn't care about changes to old messages, just about new messages. In this case, the client can skip the UID FETCH command that obtains the flags and UIDs for old messages (1:<lastseen>).

6. SPECIAL CASE: BLIND FETCH

In some cases the human may know (for whatever reason) that s/he always wants to fetch any new messages in a particular mailbox, unconditionally. In this case, the client can just fetch the messages themselves, rather than just the descriptors, by using a command like:

```
   tag1 UID FETCH <lastseen+1>:* (FLAGS RFC822.PEEK)
```

7. SPECIAL CASE: OPTIMIZING "MOVE" OPERATIONS

Practical experience with IMAP, DMSP, and other mailbox access protocols that support multiple mailboxes suggests that moving a message from one mailbox to another is an extremely common operation. In IMAP4 a "move" operation is really a combination of a COPY operation and a STORE +FLAGS (\Deleted) operation. This makes good protocol sense for IMAP, but it leaves a simple-minded disconnected client in the silly position of deleting and possibly expunging its cached copy of a message, then fetching an identical copy via the network.

Fortunately, there is a relatively easy way around this problem. By including the Message-ID: header and the INTERNALDATE data item as part of the descriptor, the client can check the descriptor of a "new" message against messages that are already in its cache, and avoid fetching the extra copy. Of course, it's possible that the cost of checking to see if the message is already in the local cache may exceed the cost of just fetching it, so this technique should not be used blindly. If the MUA implements a "move" command, it make special provisions to use this technique when it knows that a copy/delete sequence is the result of a "move" command.

Since it's theoretically possible for this algorithm to find the wrong message (given sufficiently malignant Message-ID headers), implementors should provide a way to disable this optimization, both permanently and on a message-by-message basis.

Security Considerations

Security considerations ane not discussed in this memo.

Author's Address:

Rob Austein
Epilogue Technology Corporation
268 Main Street, Suite 283
North Reading, MA 01864

Phone: (617) 245-0804
FAX: (617) 245-8122

Email: sra@epilogue.com