A Comprehensive Architecture for Continuous Media E-Mail on the Internet

David A. Turner and Keith W. Ross

Institut Eurécom

Sophia Antipolis, France

{turner,ross}@eurecom.fr

March 10, 2000

Abstract

Continuous media (CM) email is email that contains media that is rendered over time, such as audio and video, and sender-stored delivery of CM email is when the sender stores the CM message content and streams it to the recipient upon request. In this paper we propose solutions to the key problems resulting from sender-stored delivery of CM email. We begin by reviewing the sender-stored delivery model and its benefits; then we identify its weaknesses and propose solutions to them. First, there is the QoS problem, which we address by proposing a combination of sender-side and recipient-side storage. Second, there is the issue of managing the sender's storage, which now contains CM data that the recipient may wish to access at unknown points in time. We outline several solutions to the problem of message deletion. Third, new approaches are needed for forwarding and replying when sender-side storage is used—we propose a new set of techniques and discuss their advantages.

1 Introduction

Although CM may not completely replace text in email, there are many situations where the sole or additional use of audio or video in messages is more desirable than only text. However, there are inherent problems in the existing Internet email storage and delivery model for supporting CM. We review these problems and explain how they are surmountable with current Web technology.

Given user interest in CM email, and the existing Web technology to provide it, we predict the rapid emergence of CM as a popular alternative to traditional text email. However, a new Web-based storage and streaming delivery model for email raises a number of issues related to QoS, message deletion, forwarding and replying. We identify these problems and provide solutions to them. Before we begin this analysis, we first motivate our work by identifying the most important benefits of adding CM to the present email system.

Adding CM to email solves several accessibility problems. Email with audio content is particularly appropriate for the seeing-impaired. It also provides increased accessibility to persons suffering from ailments that inhibit their use of a keyboard, such as paralysis of the limbs, carpal tunnel syndrome, etc. Many other potential users of asynchronous messaging, such as young children and other persons who can not read or write their native language, would gain access to email if audio and video were more widely supported.

CM email is well suited for small portable devices, because these devices lack adequate space for a keyboard, while the space required for a microphone or video capture device is relatively small. Certainly, in environments where the user has limited use of his or her hands, CM email would be easier to compose and render than text email. For all users, CM email reduces eyestrain resulting from prolonged exposure to a monitor. Additionally, the shift from keyboard input to a more natural speaking mode would provide the user greater freedom of movement, and reduce physical stress resulting from keyboard usage.

Another advantage to CM messaging is that audio and video messages are inherently more personal than text messages. Incorporating this personal effect in email is certainly desirable for communication between family and friends, and also in many business correspondences. Additionally, audio and video messages are inherently easier to comprehend than plain text. Compare watching television to reading a book—most would agree that watching TV requires less effort.

In face-to-face communication, people use their bodies and the sound of their voices to communicate more than what they can accomplish with only the literal content of their words. For this reason, it is easier and more natural for people to communicate with CM messages. Additionally, people speak at a rate of about 180 words per minute, whereas the average person types less than 30 words per minute. Thus, it is easier and more efficient to create messages with CM content rather than text.

If the benefits of CM email are so numerous and compelling, then why is it not more widely used?

One of the reasons has been the absence of audio and video capture and playback hardware at user terminals. But this hardware barrier is starting to disappear; most computer systems now come equipped with sound cards, speakers and microphones. Also, video capture hardware is available for about $100 US, which is well within the budget of the average user. Another reason for the absence of CM email is the lack of audio and video capture and playback functionality within the user agents (mail readers). However, a more fundamental barrier to the development of CM email is the manner in which Internet email is currently stored and delivered, which we describe in the next section.

The remainder of this paper is organized as follows. In Sec. 2, we review the findings in [2], which identify the aspects of the current delivery model that act as barriers to the development of CM email. Sec. 3 reviews the sender-stored delivery model proposed in [2] that solves these problems, and describes a Web-based solution that can be implemented incrementally in individual sender systems. We conclude Sec. 3 by identifying the benefits and problems with sender-stored delivery. The final sections of the paper present our solutions to these problems. Sec. 4 describes our integrated recipient/sender-stored delivery solution to the QoS problem. Sec. 5 details our proposals for the problem of message deletion. In Sec. 6 we provide methods for message forwarding and replying. We conclude the paper in Sec. 7.

2 Barriers to the Development of CM Email

There are four main problems in the underlying infrastructure of Internet email:

- Message delivery is not universally possible, because of storage limitations in the message store of the recipient system.
- There are long delays when retrieving CM messages in low bandwidth environments.
- Email suffers from a faulty cost model.
- Resources are wasted for the storage and transport of non-rendered media.

To understand these problems, let's first review the basic delivery model of Internet email with the aid of the illustration in Fig. 1. First, user A creates an email message in her[1] user agent (UA).

[1] Throughout this paper we assume that the sender is a "she," and the recipient a "he." In the examples, we often call the sender Alice and the recipient Bob.

UA A transfers the message data by Simple Mail Transport Protocol (SMTP) to an outgoing mail transfer agent (MTA) in A's mail system. This MTA then transfers the message to the recipient's incoming MTA, which stores the message data in it's message store. The message remains in the store until the user connects to it and requests retrieval of the message.

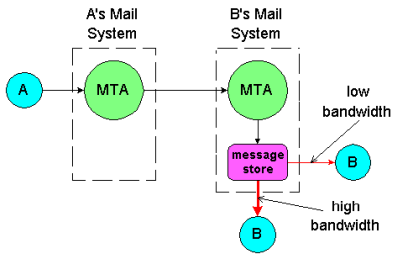

Figure 1: The Current Email Delivery Model

Incoming MTAs generally limit the size of incoming messages that they allow into their message stores. Frequently, a user is allocated a fixed amount of storage, which in general is adequate to store incoming text messages, but inadequate for the storage of messages with CM content, which tends to be quite large in comparison to text and images. For this reason, it is frequently not possible to deliver email that contains CM. Typically, the incoming MTA cuts off receipt of incoming message data, and responds with a notice that the message size has either exceeded the maximum allowable size or has exceeded the capacity of the recipient's inbox storage.

Even if the email with CM has been accepted by the recipient mail system, there is the problem of bandwidth between the recipient and the message store. If the recipient is behind a slow network connection (either because he is always behind a slow connection, or because he intermittently accesses his mail from behind a slow connection), then he will be subjected to a long wait before the entire CM message is transferred from his mail system to his UA. For example, a short video message could easily consume 1 MB of storage. With a 28.8 kbps modem, this would take more than half an hour to download. The problem is that the video data is being treated as if it were a loss-intolerant static object, such as text, rather a loss-tolerant, adaptive continuous video stream.

Mail service providers are under little incentive to begin allowing large-sized audio and video objects into their message stores, because this would increase storage and bandwidth costs, which would be translated into higher usage fees for their clients. At this point, we see that the delivery model of Internet email is contrary to the message delivery model one would expect, because the recipient is actually responsible for paying more for message delivery than the sender. Both sender

and recipient pay approximately equal amounts for bandwidth, but the recipient has the extra expense of providing temporary storage for messages in the message store.

As a final rebuke to the current Internet email delivery model, observe the waste encountered when sending a single message to multiple recipients. For large distribution lists, typically most of the message content will go unread, or partially read. When transporting CM, this would amount to a great waste of network resources for the transport of non-rendered content.

Now that we have established the basic flaws in the current Internet email delivery model, which is designed for static media, we will explain in the next section how the Web's content storage and delivery model, in combination with the adaptive streaming media capabilities of servers and browsers, solves these fundamental problems, and thus provides a way for the development of CM email.

3 The Sender-Stored Delivery Model

To resolve the infrastructural inadequacies identified in the previous section, we proposed a storage and delivery model called *sender-stored email* [2] that relies on current Web technology, and that takes into account the needs and heterogeneity of CM email users while only requiring incremental changes in the existing email infrastructure. In this scheme, the mail system is responsible for the storage of the CM content of its outgoing messages, which are streamed to recipient UAs (or media players) in the moment message rendering is desired. The entire data stream need not be sent: only those portions of the stream that are requested by the recipient, which he controls through the playback controls of his interface. To accommodate different access rates, the CM server should have the ability to transmit a compressed version of the CM data that matches the available bandwidth.

We use the term *streaming* to describe a client-server system of CM delivery in which the client initiates rendering of a stream of CM data while it is in the process of retrieving the stream from the server. Streaming delivery of CM reduces start up latency, and allows the user to make temporal jumps within the stream and change the stream's playback rate. *Adaptive streaming* is a form of streaming in which the server adjusts the compression rate of the CM data to match the bandwidth available between server and client. Adaptive streaming is used to deliver CM with the highest possible quality under existing bandwidth constraints. In this paper, we always mean adaptive streaming whenever we use the word streaming.

Under the sender-stored delivery model for CM email, the CM portion of the message is placed in a CM server in the sender's mail system, and a *base message* containing a reference to this data is sent to the recipient in the form of a small text message. (This is different from the traditional delivery mechanism, which we refer to as *bulk delivery*, in which the data comprising the entire message is sent in a single transaction.) The recipient's UA uses the base message to instantiate an appropriate media player to stream the CM from the sender's CM media server.

Fig. 2 illustrates the sender-stored delivery process. To better understand this process, suppose Alice (A) sends Bob (B) a video message. Alice's UA transfers the message in bulk to her outgoing mail transfer agent (MTA). This MTA then places the CM data with its CM server, and constructs a referencing base message, which it transfers to Bob's incoming MTA. Bob retrieves the base message from the message store of his mail system, and uses it to stream the video message from the CM server of Alice's mail system.

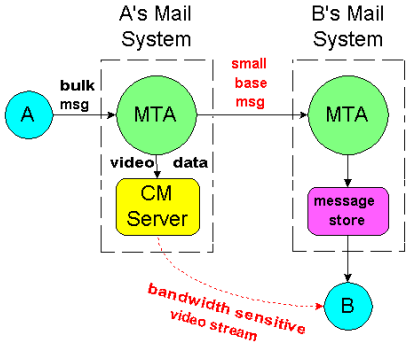

Figure 2: Sender-Stored CM email

Several other research teams have also considered forms of distributed storage for CM email similar to our sender-stored scheme. The multimedia messaging system prototypes of the BERKOM Multimedia Mail Teleservice [6], VistaMail [5], and MHEGAM [7] have all considered some form of a distributed storage architecture to solve the problem of large media files. However, these earlier contributions either propose architectures for non-Internet mail (such as X.400) or propose entirely new mail systems. Our contribution is to frame the concept of a distributed storage architecture in terms of current Internet email systems. Rather than designing a new system of conformant mail agents, we are proposing an evolutionary step that can be applied in a piecemeal manner to individual systems as they exist in today's environment. Our sender-side proposal

is backward compatible with existing recipient systems, because it requires only changes at the sender. Furthermore, our sender-stored email proposals explicitly address CM message forwarding, replying and annotation.

In recent years, email user agents have become MIME [9] compatible, and are therefore capable to a limited degree of creating and rendering email messages with multimedia content. Paralleling the MIME developments, there has been increased integration of the email user agent and the Web browser. For example, mail readers now parse text messages for URLs, and render them as hyperlinks which, when clicked upon, launch a companion Web browser that retrieves and displays the referenced page. These developments allow a form of sender-stored delivery of CM email in which the sender delivers a hyperlink to remotely stored CM message data. When the hyperlink is clicked upon, the recipient's UA delegates processing to the browser. The browser in turn delegates rendering to a media player when it recognizes the multimedia content. In fact, such schemes are already appearing in the marketplace ([12] and [13]). As CM messaging becomes more widely used, we expect that UAs will be able to render sender-stored CM messages within their own window in order to present the message as an integrated part of the user's messaging system.

3.1 Implementation of Sender-Stored CM email

We now describe by way of example how sender-stored CM email can be implemented by requiring changes only to the sender's system. In the example, Alice is sending a video message to Bob. After Alice creates the message and issues the command to send, her UA transfers the message data to her outgoing MTA. The MTA then stores the CM portion of the message with its CM server, and formats a base message with a reference to this file, which it sends along the normal route taken by a traditional email message during the push phase of email transport. When the recipient checks for new messages in his mailbox (his allocated portion of the message store), the pull phase of transport for this message begins. His UA will build a list of messages that are in his mailbox, comprised of senders' names, subject headings, message dates, etc. When he selects the new message, he will have the option to render the CM.

Presently, in order to deliver sender-stored CM email with adaptive streaming to an arbitrary recipient, the base message needs to be processed by a media player outside the recipient's UA. With current UAs, the only way to accomplish this is by formatting the base message as an HTML document that links to a secondary object in the sender's mail system. When the recipient clicks

on the link, his UA will instantiate its companion Web browser and have it process the hyperlink request to the secondary object, which ultimately leads to the instantiation of a media player and the commencement of the media stream.

There are several methods to implement streaming playback for the recipient; we will describe two examples that illustrate two different approaches. In the first example, the recipient is using Internet Explorer to process the hyperlink to the secondary object. The secondary object is an HTML file that contains a reference to an ActiveX control that functions as the media player. If the control is not already in the client's system, it is downloaded automatically. The control is then initialized to stream the CM stored with the sender's CM server. In the second example, the recipient is using Netscape to process the hyperlink to the secondary object, which is also an HTML document. But rather than containing a reference to an ActiveX control, this HTML file contains a reference to a Java applet. The applet is loaded by the browser and run in its Java virtual machine. The applet contacts the sender's CM server to stream and render the CM.

In both examples, the base message would look something like that in Fig. 3 below.

```
From: "Alice Adams" <alice@aaa.com>
To: "Bob Brown" <bob@bbb.com>
Subject: meeting announcement
Date: Tue, 9 Feb 1999 13:18:45 +0100
MIME-Version: 1.0
Content-Type: text/plain;
     charset="iso-8859-1"
Content-Transfer-Encoding: 7bit

Video message: http://mail.aaa.com/12345.html
```

Figure 3: Base Message Referencing Secondary Object

Most likely, Bob's UA will display the URL in Fig. 3 as a hyperlink, which he can activate to instantiate his browser to retrieve the secondary object (12345.html). If his mail reader does not support this function, then Bob would have to manually start his browser and point it to the secondary object. When Alice's Web server receives the request for the secondary object, it detects the operating system and browser that Bob is using by reading the appropriate headers from the browser's HTTP request. With this information, Alice's system returns an appropriate secondary object. For instance, if Bob's browser were a version of Netscape that provides a Java API that supports the streaming playback of Alice's video, then the Web server would return an HTML

document with a reference to the Java applet, including a PARAM tag to initialize the applet with the URL that locates the streamable video data. Fig. 4 is one such possibility for a response.

```
<HTML><BODY>
<APPLET code="cmail.class">
<PARAM name="URL">rtsp://mail.aaa.com/12345.mpg</PARAM>
</APPLET>
</BODY></HTML>
```

Figure 4: Secondary Object with Applet

If Bob's browser doesn't support the Java environment needed by the applet, he may be prompted to allow the automatic installation of the enabling software.

In this example, we used short URLs such as 12345.html and 12345.mpg. But in a real implementation, these URLs would need to be long strings of random sequences of characters. Such a scheme would provide a level of privacy equivalent to sending passwords in plain text, and would allow recipients to access their messages from arbitrary hosts at arbitrary IP addresses. Additional security would require the use of encryption and certificate-based recipient identification.

3.2 Benefits of Sender-Stored Delivery of CM email

Sender-side storage combined with streaming solve the four basic problems summarized in Sec. 2. First, note that the problem of universal message delivery is solved. Because the sender stores the CM data, and only sends a small referencing base message, it is unlikely that the base message will be rejected due to storage limitations in the recipient's message store. Additionally, most mail readers today delegate processing of hyperlinks to a companion Web browser, so when the base message is in the form of an HTML document or contains a hyperlink, the UA will pass control to the browser. In turn, the browser—by virtue of its Java API, plug-in architecture or ActiveX control support—is configurable to support the streaming delivery of the remotely stored media.

Because the CM is adaptively streamed, senders can deliver CM messages that render without significant startup delays, regardless of the recipients' access rates. Recipients behind slow network connections are not encumbered with excessive retrieval delays, because the streaming mechanism will increase the compression rate of the media stream to match the available bandwidth. Additionally, streaming message content from a remote store enables a thin client with relatively small local memory to render CM messages.

Sender-stored email delivery follows a more intelligent economic model, because the sender now bears the cost of message storage, and the recipient only pays for the bandwidth used in transmitting that portion of the message data that he chooses to render.

Finally, sender-stored email conserves bandwidth in the case when recipients do not choose to render the message, or choose only to render part of the message.

With sender-storage mechanisms in place, recipient systems may opt to accept only small-sized messages into their message stores, thereby forcing senders to resort to sender-stored delivery. The primary motivation for mail service providers to do so will be to reduce their storage and bandwidth costs, especially those related to audio and video spam.

3.3 Problems with Sender-Stored Delivery of CM email

Sender-stored email is a major paradigm shift for existing email, and thus engenders several new problems. First, there is a QoS problem, which results from the streaming delivery of CM across a bandwidth-limited network path. We address this problem in the next section on *integrated recipient/sender-stored email*, where the sender first attempts to deliver the CM data to the recipient's MTA in the usual manner, and then the recipient streams the CM data from his mail system, which is presumably closer to him than the sender's media server.

Another problem arising from the use of sender-side storage includes deciding when to delete CM message data from the sender's storage, which the recipient may wish to access at an unknown point in time. (We will sometimes refer to this storage as the sender's outbox.) Deletion of CM from the sender's outbox storage can be done manually or automatically. In *manual deletion*, the sender is responsible for managing the contents of her outbox in the same manner in which she manages the contents of her inbox. In *automatic deletion*, we propose several solutions in increasing degrees of complexity. For both deletion approaches, we show how the use of CM access statistics can be used to avoid both the premature deletion of message data and the retention of stale data.

Also, new approaches are needed for forwarding when sender-side storage is used. When forwarding, one must decide between two types of forwarding, which we refer to as *reliable forwarding* and *unreliable forwarding*. In unreliable forwarding, the forwarder sends a copy of his referencing message, so that the forward recipient will stream the CM data from the origin sender's CM server. We consider the problems this engenders, and then describe how these problems can be avoided with the more expensive reliable forwarding procedure, where the forwarder copies the CM data

into the storage of his own CM server, and sends a reference to this copy rather than a reference to the CM that resides in the origin sender's system.

Replying to sender-stored email also introduces new complexities. When replying, a person frequently sends an annotated response, that is, a response that contains the whole or pieces of the sender's original message. In the case of sender-stored email, the annotated data already resides in the original sender's storage, and so it doesn't need to be delivered; instead, a reference to it is used.

4 Integrated Recipient/Sender-Stored email

4.1 Pure Recipient-Stored Delivery

Streaming CM message content from the sender's CM delivery system raises a QoS issue, because the network path between the sender's CM server and the recipient may be congested. Under congestion, the server will only be able to transmit a highly compressed version of the CM, or may be forced to introduce rendering delays to build up a large playback buffer. Therefore, we would like to move the message data closer to the recipient to improve quality. To solve this problem, we introduce *recipient-stored* delivery of CM email, where the CM is transferred into the recipient system's message store in the normal manner using SMTP, but streamed to the recipient from his message store in the moment that he chooses to render the message. After we describe this concept in more detail, we then propose integrated recipient/sender-stored email as a more flexible mechanism, which will better serve the interests of the majority of email users.

The process of recipient-stored delivery is depicted in Fig. 5, where the message is transferred in bulk from Alice's UA to her MTA, which then transfers the bulk message to the recipient's MTA. Once the message data arrives at the recipient's MTA, the CM data can be extracted from the message and given to a CM server under the control of the recipient's mail system. The message that is retrieved by the recipient's UA via POP, IMAP or HTTP will be the base message referencing the separately stored CM data. By moving the message data into the recipient's storage, the media can be streamed to the recipient from a location that is most likely closer to him, and thus there will be more available bandwidth that can be used to provide higher quality playback. An additional benefit is that message deletion is now under the control of the recipient.

Our proposed design for pure recipient-stored delivery introduces a filter within the recipient

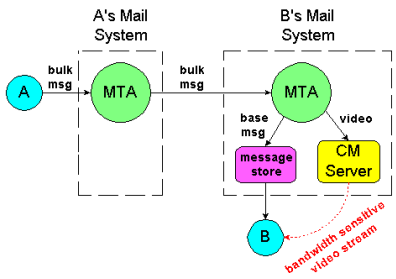

Figure 5: Recipient-Stored CM email

MTA that modifies the push phase of message transport. All incoming messages are first passed through the filter before entering the message store. When the filter finds a MIME body part that contains CM, it extracts the CM data, removes any base-64 encoding, and then stores the resulting CM data in storage accessible by the system's CM server. To create the base message, the filter replaces the CM data in the original message with a reference to the CM data. The MTA then places this base message in the recipient's mailbox within the message store for retrieval by the recipient's UA. When the recipient chooses to render the CM message, the message is streamed from the CM server of the recipient's mail system to a media player on the recipient's machine.

Recipient-stored delivery can be wholly implemented within the recipient's mail system, without making changes at the sender or at the recipient UA. Thus a mail service provider can implement the system without requiring any changes to client software, except possibly the automatic installation of media player software, as mentioned in the section on sender-stored delivery.

In addition to improving the QoS, recipient-stored delivery also solves the deletion problem by placing the CM data in storage under the control of the recipient.

4.2 Integrated Recipient/Sender-Stored Delivery

Although QoS can in general be improved with recipient-stored delivery, there are three situations in which it is desirable to use a pure sender-stored delivery approach. The first situation is when the recipient's mail system is not capable of streaming CM content to the recipient. In this case, if the recipient of a large message is behind a slow connection, he will suffer a long delay when his UA retrieves the entire message before commencing playback. The second situation is when mailing to a large distribution list, where most recipients are expected to render very little of the CM content.

In this case, a large amount of bandwidth is conserved by streaming only data that is requested, rather than pushing all of the data into the message store of each recipient system. The third situation is when a message is addressed to multiple recipients within the sender's mail system. If recipient-stored delivery is used, a separate copy of the message data will be placed in the allocated storage (mailboxes) of each of the recipients. Since the message is not subject to modification by the recipients—it is *read only*—there will be needless duplication of message data. Furthermore, the recipients will retrieve the message data from the mail system of the sender, and thus there is no QoS improvement with recipient-stored delivery.

We propose and advocate the following mail delivery strategy for an individual message with a single recipient. If the recipient is local, i.e., if he shares the same mail system as the sender, then use sender-stored delivery. Otherwise, query the recipient's mail system to see if it is *CM-aware*, that is, if it will stream the CM data to the recipient from its message store. (This will be explained in the next paragraph.) If the response is affirmative, then deliver the message to that system in bulk. If the response is negative, then use sender-stored delivery to insure adaptive streaming delivery.

A mail system can be queried to see if it is CM-aware by using Extended SMTP [10]. Alice's MTA could have queried Bob's MTA with the ESMTP protocol exchange shown in Fig. 6. Bob's MTA responds that it supports the extended functionality identified by the keyword CMAWARE.

```
MTA A: (initiates TCP connection to MTA B through port 25)
MTA B: 220 bbb.com mail server ready
MTA A: EHLO mail.aaa.com
MTA B: 250-bbb.com
MTA B: 250-SIZE
MTA B: 250 CMAWARE
```

Figure 6: ESMTP Server Announces that it is CM-Aware

For a message with multiple recipients, we propose a slightly more complicated delivery strategy. When a group of recipients share a common domain name in their email addresses, it means that they use the same MTA and that a message addressed to all of them can be delivered within a single SMTP message transfer. Thus the bandwidth cost of sending a large message to many recipients that share the same mail system equals the cost of sending the message to one of them. For this reason, we group the recipients of the message by the domain name appearing in their email addresses. For the recipients who are local, use sender-stored delivery. If the number of non-local

recipient mail systems exceed some threshold, then use sender-stored delivery. Otherwise, query each mail system to see if it is CM-aware. If the response is affirmative, then deliver the message to that system in bulk. If the response is negative, then use sender-stored delivery to insure adaptive streaming delivery.

So that recipient-stored delivery doesn't support the faulty cost model described in section 2, users should be able to specify those sources of email from which they are willing to accept large messages, and set a message size limit for all other senders. In this way, they can filter out potential video and audio spam, yet allow large CM messages from known senders to pass into their allocated storage within the message store.

5 Message Deletion

In sender-stored email, the sender (or the sender's system) makes the decision regarding when to delete message content from storage. The recipient would prefer that the CM data referenced by his received base message be available until the time he deletes the base message. However, Internet email does not currently support a form of storage negotiation between sender and recipient systems that could be used to avoid premature deletion of sender-stored message content. Thus, sender-stored email systems must rely on non-deterministic methods for deletion of sender-stored CM data.

Even if the sender were to know how many outstanding references existed to a CM object in her storage, she may still opt to delete it, because she may be unwilling to service all requests for the object in the case that many external references to it have been created by repeated forwarding of the base message.

When recipient-stored delivery of CM email is used—and the entire content of the message is delivered into the recipient's storage—then these problems don't exist, because the recipient decides how long to keep messages in storage and when to delete them to make space available for new messages. This works fine for IMAP-based and Web-based UAs, because the remote store can delete CM when its referencing base message is deleted. But it does not work for POP-based systems, because base messages are kept in local storage rather than in remote storage with the CM. When a message in local storage is deleted by the user, the UA does not inform the remote store of the deletion. Therefore, non-deterministic methods of CM deletion are also relevant for recipient-stored

messages that are accessed through POP.

When CM messages are sent to recipients within the same mail system as the sender, then the system has knowledge of whether or not the recipient's referencing base messages have been deleted. In this case, CM can be protected from deletion until all known referencing base messages have been deleted. However, if one of the recipients has forwarded the base message outside the mail system, the system will loose the ability to track the number of outstanding base messages referencing the CM. Thus, whenever a copy of a base message leaves the mail system, a non-deterministic policy of CM deletion must be used.

In the following subsections, we identify a number of different possible message store management strategies for sender-stored email systems, and examine how they would behave under various scenarios.

5.1 Manual Deletion

The simplest scheme of storage management is similar to the ordinary manual management of one's mailbox (the user's allocated portion of the message store). Inside the UA, the sender views her messages arranged into a tree of folders. These messages include messages that she has received from other users, messages she has sent and retained a copy of, and in particular, base messages she has sent that refer to CM she has created and makes available to recipients through her system's CM server. When she deletes a base message from her mailbox, her mail system also deletes the CM to which it refers, and so she controls at what point the recipient will no longer be able to stream the message data from her CM server.

For manual deletion, the UA should provide the user with information about the capacity of her mailbox storage and the amount of storage being used by the CM stored in it, which could be presented as a pie chart showing the percentage of consumed versus available storage. To make room for new outgoing CM (and incoming messages) the user deletes from her mailbox those messages she considers expendable. The UA should respond by deleting both the base message and the CM to which the message refers. Message sizes should be indicated in the display, so that the user knows the impact of each message on her storage allocation.

One drawback to this approach is that the user must suffer the inconvenience of managing the available space. Prior to adoption of a sender-stored system, the user only had to make decisions regarding preservation or deletion of her received messages; now she must additionally manage

messages that could still be rendered by other users. But users are already faced with the responsibility of managing their finite storage resources, and so the added responsibility of deciding which outgoing messages should be saved and which should be deleted may not be perceived as excessively inconvenient.

In addition to the senders, the recipients of sender-stored messages must also be aware of the issue of CM lifetime. If a recipient of sender-stored CM wishes to be able to access the CM of a message at an arbitrary point in the future, and does not believe the sender will provide it to him at that time, he must copy the CM into storage that is under his control. If the recipient desires to move the CM into his own storage, then the sender ought to provide a lossless mechanism of CM transport, so that the recipient can obtain a high quality copy of the message.

5.2 FIFO Deletion

One approach to message deletion is a simple first-in/first-out (FIFO) queue of CM data. Under this approach, the sender has a fixed amount of storage reserved for her outgoing CM content. Messages are retained for as long as possible, but when room is needed for new content, they are deleted in the order of oldest first until there is enough space for the new content.

The advantage of this approach is that it is automatic; the user is relieved of the burden of deciding which messages to delete. The problem is that the FIFO approach to message deletion makes it possible for non-rendered messages to be deleted before rendered messages. Suppose Alice sends Bob a video message on Monday, then sends a different video message to Claire on Tuesday. Bob has been home with the flu, and so has not been to the office to check his mail. Claire, on the other hand, viewed Alice's video message the day it arrived, and quickly deleted the base message from her mailbox. On Thursday, pressed for new space to hold new outgoing messages, Alice's mail system deletes Bob's non-rendered video data, while uselessly retaining Claire's video data. Bob arrives at work on Friday, selects Alice's message, issues the command to play it, and receives nothing. Therefore, FIFO is insensitive to whether a message has been read or not.

5.3 Expiration Date Deletion

Another problem with FIFO is that some messages are intended to be more short-lived than others. For example, suppose Alice sends Bob a reminder to bring a certain report with him to the meeting they will have at 2:30 later in the day. Clearly, such a message has a very short lifetime. As

a contrasting example, suppose Alice sends Bob a description of a product she thinks would be interesting to his company. Alice may want the CM content to remain available for a relatively long period of time to ensure the delivery of her sales message in the event it is requested at a later time.

To accommodate messages with different lifetime expectancies, expiration dates can be used to override the FIFO order of automatic message deletion. This can be implemented as two queues, as shown in Fig. 7. Messages initially enter an expiration queue. When their expiration date is reached, they are moved to an expendable (FIFO) queue. The system keeps messages in the expendable queue for as long as possible; but when space is needed for new content, CM is retired from the expendable queue in the order of oldest first.

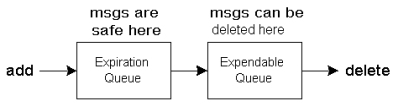

Figure 7: Message Lifetime Queues

Expiration information can be supplied in both human and machine-readable form. The human-readable expiration date allows recipients to manually copy CM message content into their own storage when they desire to retain it beyond its expiration date. The machine-readable form of the expiration date allows recipient mail systems to automatically pre-fetch CM that has not yet been rendered—or CM that is referenced by undeleted messages—just prior to its approaching expiration date, or to implement other messaging policies that utilize expiration date information.

Expiration information regarding referenced CM can be added as headers in the base message. For example, if Alice's video data is due to expire on 18-Feb-2000, the following header can be added to the base message in Fig. 3:

```
Link-expiration="http://mailhost.aaa.com/12345.smil 18 Jan 99 1430 GMT"
```

The expiration header includes two components. The first component identifies the link that points to the secondary object, which is needed to distinguish ordinary links that may be a part of the message from links to sender-stored CM. The second component of the header is the time the CM expires.

5.4 CM Access Statistics

Both manual and automatic deletion mechanisms can be enhanced with the use of *CM access statistics*. When a recipient accesses the CM of a referencing message, the sender's mail system can make a record of this access. Because CM can be retrieved in parts, the record keeping of access statistics could become complex. The most detailed form of record keeping would entail a record of each streaming event, including the time that a particular byte range within the CM data file was streamed. A less detailed mechanism might simply mark whether or not any part of the CM data was streamed.

In the manual deletion approach, the UA can display the access statistics for CM attached to messages in one's list of sent messages. One would use these access records in deciding whether to delete or maintain a particular CM message. For example, after Alice's CM server streams her video to Bob, it makes a record of the event, which includes the byte range delivered and the time of delivery. When Alice runs her UA, she opens the folder containing this message and selects the message she sent to Bob. The access statistics for the message show that it was streamed in its entirety last week. Because of the nature of the message, she decides that it has already served its purpose and can be deleted. If she is using an IMAP or Web-based UA, the command to delete the message is processed by the remote mail system, which can thus delete both the base message and its referenced CM. But if she is using a POP-based system, her UA would only be able to delete the base message from its local storage; to delete the CM from the remote storage of the CM server, a new protocol between UA and mail system would be required.

In the automatic deletion system, the access statistics can be used to order the CM in the expendable queue, so that the oldest CM is not necessarily the first to be deleted. CM that has not yet been retrieved could be given higher priority for retention over newer CM that has already passed through a phase of being accessed. Automatic deletion is especially appropriate for POP-based systems, because no additional protocol exchange mechanism needs to be developed to delete the remotely stored CM.

One problem with implementing a storage policy that tracts recipient access statistics is identifying which recipient is retrieving the message data when a message has been sent to multiple recipients or has been forwarded. The IP address of the recipient's mail server will most likely not be the same IP address of the system at which the recipient renders the message, so when a recipient retrieves message data, the sender's message delivery system can not determine which recipient on

its list of message recipients is actually accessing the message.

One solution is to use different URLs inside the base messages that are delivered to the different recipients. These base messages reference different secondary objects, which contain different URL references to the CM. The CM server then maps each of these different URLs to the same CM file that represents the contents of the message. For example, suppose Alice sends a video message to Bob and Claire. Her system constructs base messages and secondary objects so that Bob's player requests the file 12345b.mpg and Claire's player requests 12345c.mpg. Alice's CM server will map requests for these files to the same object, 12345.mpg. When Bob reads his message, Alice's CM server would receive a request for 12345b.mpg. It would stream the file 12345.mpg, but record the event as Bob's access. When Alice checks the access statistics for this message, she will see that only Bob has accessed the message content, and that Claire has not.

The solution of unique URLs fails in the case that a message is forwarded, because now two different recipients will have base messages with identical URLs. The CM server can not distinguish whether an incoming request is from the intended recipient or a forwarded party. However, this problem will not occur if the forwarder makes a copy of the CM in his own storage and reconstructs the base message to point to this copy. We call this approach *reliable forwarding*, which we discuss in the next section.

A large percentage of email in a corporate environment is directed toward recipients within the corporation, and thus are not transferred beyond the corporate mail server. In this case, message deletion can be made completely reliable, because the central mail system can track the number of undeleted references to a particular CM file, and retain the CM data until the last reference to it is deleted. This strategy can be implemented if IMAP or HTTP is used as the method of mailbox access, because message data remains in the domain of the central mail system rather than being transferred to the user's local storage as is done under POP. Such a system would reduce network traffic and conserve disk space by reducing the amount of redundant static data in the system.

6 Forwarding and Replying

6.1 Forwarding

A sender-side storage architecture represents a fundamental paradigm shift in email distribution. Consequently, the operations of forwarding and replying must be completely rethought. For ex-

ample, suppose that Bob has a base message from Alice, and that he has just viewed its video by streaming it from Alice's CM server. If he wants to forward the message to Claire, Bob (or his system) must decide between two different types of forwarding, which we refer to as *unreliable* and *reliable*.

In unreliable forwarding the user's system simply sends a copy of the base message to the forward recipient. The approach is unreliable in the sense that the forwarder has no control over the existence of the referenced CM data; it is possible that the original author of the message deletes the CM before the forward recipient has a chance to render it. Fig. 8 illustrates unreliable forwarding. In this example, Alice first sends Bob a sender-stored CM message. When Bob instructs his UA to forward the base message to Claire (C), Bob's MTA transfers the base message to Claire's MTA. Claire retrieves the forwarded base message from her MTA through her UA, and uses the base message to stream the video message from Alice's mail system.

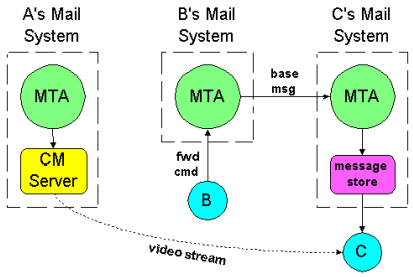

Figure 8: Unreliable Forwarding of a CM Message

Alternatively, with reliable forwarding the forwarder's MTA first retrieves a copy of the CM, then sends it to the forward recipient using the delivery strategy described in Section 4. The approach is reliable in the sense that the forwarder has control over the lifetime of the CM data. Fig. 9 illustrates the scenario were Bob's system (MTA B) copies the video data into the message store of his system, and delivers a referencing base message to Claire. Claire will retrieve the base message from her mailbox and stream the video from Bob's CM server. Alternately, Bob's MTA could have delivered the forwarded message in bulk to Claire's MTA, so that Claire would then stream the message data from her own mail system.

It is possible for the user to decide the method of forwarding on a per message basis. Under this manual approach, Bob estimates the likelihood that the audio data becomes inaccessible before Claire tries to render it. His estimate is influenced by the nature of the message, by the expiration

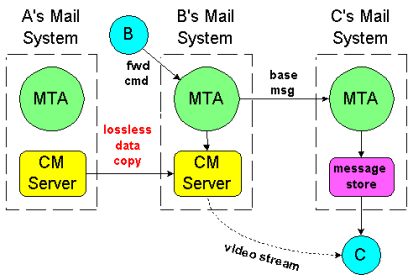

Figure 9: Reliable Forwarding of a CM Message

date Alice may have specified in the base message, and whether Bob wants to attempt bulk delivery of the message to Claire in order to improve the quality of playback.

If an automatic approach is used, Bob's mail service decides the type of forwarding by comparing the expiration date placed in the base message by Alice with the expiration date specified by Bob. If Bob's expiration date is earlier than Alice's date, then Bob's mail service simply forwards the original base message to Claire's mail system, with its reference to the CM stored in Alice's mail system. On the other hand, if Bob's expiration date comes after Alice's date, then Bob's mail system will copy the video data into its storage and deliver it from there to Claire.

Whether the result of user action or automatic mechanism, if Bob's mail service copies the video data into its message store, it should do so over a reliable TCP-based protocol, such as FTP or HTTP, in order to avoid the problem of compounding streaming loss. Additionally, it is desirable for the sender to provide a reliable transport mechanism for their outgoing CM to allow recipients to make lossless copies of it into their own storage. For example, suppose that after streaming the video content from Alice's CM server, Bob wants to retain a lossless copy of the message for an indefinite period. He believes that Alice will eventually delete the video data, so he instructs his UA to copy the CM referenced within the base message into his system's message store. When Bob's UA issues this instruction, his mail service retrieves the message content from Alice's storage using a reliable protocol such as FTP. Bob's mail service then modifies the base message so that it now resolves to the local copy of the video. Note that the message is still delivered to Bob's UA as a base message, only it now resolves to CM data as it is stored in his mail system.

Adding new functionality that involves communication between UA and mail systems, as just described, is much easier to accomplish with a Web-based UA system, because the user's mail service provides the UA interface through HTML documents or mobile code. Thus, changes in the behavior

of the remote mail service and the UA interface can be accomplished together. Implementing this new functionality in a standalone UA system is more difficult, because it requires concurrent changes in two separate applications: the remote MTA and the local UA.

6.2 Replying

Frequently, recipients include the original message, or pieces of the original message, in their replies. It's possible to do the same with CM email. Let's look at an example. Suppose that Bob plays Alice's video message by streaming it from her CM server. He wishes to comment on something specific that Alice has said. He constructs a video reply in which he begins by speaking, then inserts into his message that section of Alice's video message he wishes to quote, and then ends the message with some additional speaking of his own. To do this, he uses the repositioning controls available to him, such as slider bar, rewind and fast forward buttons. Once he locates the point within Alice's video that he desires to quote, he clicks on a *start copy* button to begin capturing the video into the system clipboard. When the video reaches the end of what he wishes to quote, he clicks on stop copy, and now the clipboard contains the interval of Alice's video that he wishes to playback within his reply message. (In the case that his UA has cached Alice's video message from the first time he streamed it, replaying parts of her message as just described does not generate any additional network traffic.)

Now, Bob clicks the record button in his message reply window, and begins speaking. When he gets to the point where he wishes to insert Alice's comment, he clicks on the insert command to insert the contents of the clipboard into his video, and then speaks the rest of his message. He now has a video message with a video quote embedded within it. Because there is no need to deliver data that is already in Alice's storage, a reference to the video data in her mail system is used, rather than storing a second copy of Alice's data in Bob's mail system.

When Bob issues the command to send his annotated message, his mail system constructs a base message and secondary object with the SMIL file depicted in Fig. 10. Notice that the link pointing to the embedded video data is a reference into the storage of Alice's mail system.

```
<smil><body>
<video src="rtsp://mailhost.bbb.com/67890.mpg" clip-end="10s"/>
<video src="rtsp://mailhost.aaa.com/12345.mpg"
     clip-begin="25s" clip-end="42s"/>
<video src="rtsp://mailhost.bbb.com/67890.mpg" clip-begin="10s"/>
<body><smil>
```

Figure 10: The SMIL Document of a Sender-Stored Annotated Reply

7 Conclusion

We have identified the major weakness of Internet email that obstruct the development of CM messaging. These include recipient storage limitations that make message delivery impossible, excessive delays that result from inappropriate data retrieval mechanisms, a faulty cost model in which the recipient bears much of the cost of email delivery, the duplication of message data within mail systems when email is sent to multiple recipients, and the wasteful delivery of non-rendered message data.

We solve these problems with a sender-stored message delivery architecture, which can be implemented and incrementally deployed with current Web technology. As this represents a major paradigm shift in existing email practice, it naturally engenders new problems, which include reduced QoS, deletion of stale message data, and the intricacies of forwarding and replying.

In order to improve QoS in sender-stored delivery, we propose the combined use of both sender and recipient-stored delivery. In recipient-stored delivery, the CM is still delivered in streaming mode to the recipient in order to minimize start up latency, but it is done so from the recipient's mail system, which will provide better QoS when closer to the recipient. To solve the problem of message deletion, we proposed and examined several solutions, including both manual and automatic message deletion, and the use of recipient access statistics and expiration dates. We identified two methods of forwarding: reliable and unreliable, and discussed situations in which one is more appropriate than the other. We identified the main intricacy with replying as enabling annotation, and we described a SMIL-based method to support annotated replies.

References

[1] L. Huges. Internet E-mail: Protocols, Standards, and Implementation. Artech House, Norwood, MA, 1998.

[2] D. Turner and K. Ross. Continuous Media E-mail on the Internet: Infrastructure Inadequacies and a Sender-Side Solution. Submitted to IEEE Network Magazine.

[3] J. Reynolds, J. Postel, A. Katz, G. Finn, and A. DeSchon. The DARPA Experimental Multimedia Mail System. IEEE Computer, Oct, 1985.

[4] D. Turner and K. Ross. Asynchronous Audio Conferencing on the Web. International Symposium on Intelligent Media and Distance Education, Baden-Baden, Germany, Aug, 1999.

[5] C. Hess, D. Lin, and K. Nahrstedt. VistaMail: An Integrated Multimedia Mailing System. IEEE Multimedia, Oct-Dec, 1998.

[6] G. Schürmann. Multimedia Mail. Multimedia Systems, ACM Press, Oct, 1996.

[7] V. Gay and B. Dervella. MHEGAM: A Multimedia Messaging System. IEEE Multimedia, Oct-Dec, 1997.

[8] S. Carrier and N. Georganas. Practical Multimedia Electronic Mail on X.400. IEEE Multimedia, winter, 1995.

[9] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. Internet Engineering Task Force, Network Working Group, RFC 2045, Nov, 1996.

[10] J. Klensin, N. Freed and K. Moore. SMTP Service Extensions for Message Size Declaration. Internet Engineering Task Force, RFC 1870, Nov, 1995.

[11] P. Hoschka, ed. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. Synchronized Multimedia Working Group, W3C Recommendation, Jun, 1998.

[12] Wimba.com. Internet startup specializing in voice-enabled newsgroups and messaging. (http://www.wimba.com)

[13] Onebox.com. Internet startup providing voice-mail, e-mail and fax. (http://www.onebox.com)