A Comprehensive Architecture for Continuous Media Email

David A. Turner and Keith W. Ross
Eurecom Institute, France

Continuous media email has numerous advantages over plain text email. We propose solutions to the key problems resulting from sender-stored delivery of continuous media email that have prevented implementation. To address quality of service, message deletion, and forwarding and replying problems, we advocate an integrated sender- and recipient-stored delivery approach.

Many situations exist where using audio or video in email messages might be more desirable than text alone. The benefits of continuous media (CM) email are numerous and compelling (see the sidebar "Benefits of Continuous Media Email"), but it still isn't widely used. Unfortunately, inherent problems exist in the current Internet email storage and delivery model for supporting continuous media.

The absence of audio and video capture and playback hardware at user terminals is one problem, 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 problem is the lack of audio and video capture and playback functionality within the user agents (mail readers). Currently, a more fundamental barrier to CM email development is how we store and deliver Internet email.

These problems 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 quality of service (QoS), message deletion, forwarding, and replying.

Barriers to CM email development

The underlying infrastructure of Internet email has four main problems:

- Message delivery isn't universally possible, because of storage limitations in the recipient system's message store.
- In low bandwidth environments, it takes a long time to retrieve CM messages.
- Email suffers from a faulty cost model.
- Storing and transporting nonrendered media wastes resources.

To understand these problems, let's first review Internet email's basic delivery model (see Figure 1). First, Alice creates an email message in her user agent. (Throughout this paper we assume that the sender is a she, and the recipient a he. In the examples, we call the sender Alice and the recipient Bob.) Alice's user agent transfers the message data by simple mail transport protocol (SMTP) to an outgoing mail transfer agent in Alice's mail system. This MTA then transfers the message to the recipient's incoming MTA, which stores the message data in its message store. The message remains in the store until the user connects to it and retrieves the message.

MTAs generally limit the size of incoming messages that they allow into their message stores. Frequently, they allocate a fixed amount of storage for users. That amount is generally adequate to store incoming text messages but inadequate for CM content, which tends to be large compared to text and images. For this reason, it's frequently impossible to deliver email containing 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 recipient mail system accepts the email with CM, a bandwidth problem between the recipient and the message store still exists. If the recipient is behind a slow network connection (either because he's 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 transfers from his mail system to his user agent. For example, a short video message could easily consume 1 Mbyte of storage. With a 28.8-

1070-986X/01/$10.00 © 2001 IEEE

Kbps modem, it would take more than half an hour to download. The MTA is treating the video data as if it were a loss-intolerant static object, such as text, rather than 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 that would translate into higher usage fees for their clients. In the delivery model of Internet email, the recipient pays more for message delivery than the sender, contrary to what we would expect. Both sender and recipient pay approximately equal amounts for bandwidth, but the recipient must support the extra expense of providing temporary message storage.

Lastly, consider the waste we encounter when sending a single message to multiple recipients. For large distribution lists, 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 nonrendered content.

Sender-stored delivery model

To resolve these infrastructural inadequacies, we proposed a storage and delivery model called sender-stored email[1] that relies on current Web technology. It considers the needs and heterogeneity of CM email users while only requiring incremental changes in the existing email infrastructure. In this scheme, the mail system stores the CM content of its outgoing messages, which it streams to recipient media players the moment the user desires message rendering. It doesn't need to send the entire data stream: it only sends those portions that the recipient requests, which he controls through the playback controls of his interface. To accommodate different access rates, the CM server should be able to transmit a compressed version of the CM data that matches the available bandwidth.

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

Under the sender-stored delivery model for CM email, the sender's mail system places the message's CM portion with a CM server and delivers by SMTP a base message—which is a small text message that references the CM—to the recipient. (This differs 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 user agent uses the base message to

Benefits of Continuous Media Email

Adding continuous media to email solves several accessibility problems. Email with audio content is particularly appropriate for the visually impaired. It also provides increased accessibility for people suffering from ailments that inhibit their use of a keyboard, such as limb paralysis and carpal tunnel syndrome. Many other potential users of asynchronous messaging, such as young children and other people who can't read or write their native language, would gain access to email if industry systems widely supported audio and video.

CM email suits small, portable devices well. These devices lack adequate space for a keyboard, but a microphone or video capture device needs little space. Certainly, in environments where users have limited use of their 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 speech would provide the user greater freedom of movement and reduce physical stress resulting from keyboard use.

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 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 with just text. For this reason, it's 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's easier and more efficient to create messages with CM content rather than text.

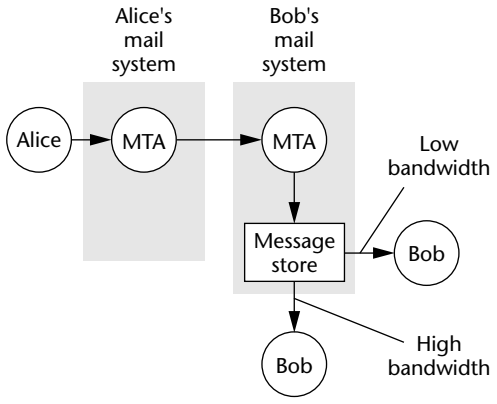

Figure 1. Current email delivery model.

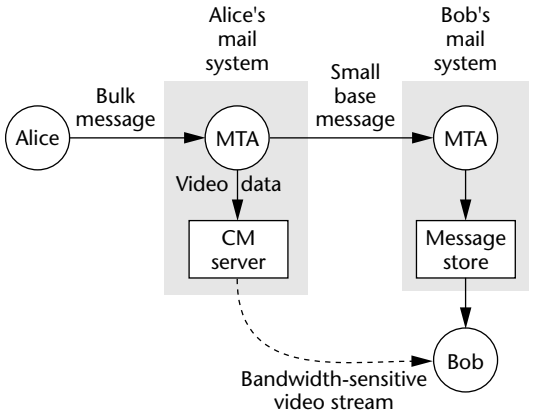

Figure 2. Sender-stored CM email.

instantiate an appropriate media player to stream the CM from the sender's CM media server.

Figure 2 illustrates the sender-stored delivery process. To better understand this process, suppose Alice sends Bob a video message. Alice's user agent transfers the message in bulk to her outgoing MTA. Her 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.

Several other research teams have 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,[2] VistaMail,[3] and MHEGAM[4] are all some form of a distributed storage architecture that solve the large media file problem. However, these earlier contributions either propose architectures for non-Internet mail (such as X.400) or entirely new mail systems. Our contribution frames 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 propose an evolutionary step that industry can apply 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 only requires changes at the sender. Furthermore, our sender-stored email proposal explicitly addresses CM message forwarding, replying, and annotation.

In recent years, email user agents have become compatible with Multipurpose Internet Mail Extensions (MIME)[5] and are capable, to a limited degree, of creating and rendering email messages with multimedia content. Paralleling these MIME developments, there has been increased email user agent and Web browser integration. For example, mail readers now parse text messages for URLs and render them as hyperlinks, which when users click on them, launch a companion Web browser that retrieves and displays the referenced page. These developments allow a form of sender-stored CM email delivery in which the sender delivers a hyperlink to remotely stored CM message data. When the user clicks on the hyperlink, the recipient's user agent delegates processing to the browser. The browser in turn delegates rendering to a media player when it recognizes the multimedia content. In fact, Wimba.com and Onebox.com are already offering such schemes in the marketplace. As CM messaging becomes more widely used, we expect that user agents will be able to render sender-stored CM messages within their own window to present the message as an integrated part of the user's messaging system.

Implementing sender-stored CM email

In our example, Alice sends a video message to Bob. When Bob checks for new messages in his mailbox, the pull phase of transport for this message begins. His user agent will build a list of messages that are in his mailbox, comprised of senders' names, subject headings, message dates, and so on. When he selects the new message, he'll have the option to render the CM.

Presently, to deliver sender-stored CM email with adaptive streaming to an arbitrary recipient, a media player outside the recipient's user agent needs to process the base message. With current user agents, 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 user agent will prompt its companion Web browser to process the hyperlink request to the secondary object, which ultimately leads to the media player's launch and the start of the media stream.

Several methods implement streaming playback for the recipient—we'll describe two examples that illustrate two different approaches. In the first example, Bob's system uses 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 isn't already in the client's system, it's downloaded automatically—assuming the user grants permission to do so—and then initializes the control to stream the CM stored with the sender's CM server.

In the second example, the recipient uses 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 refer-

ence to a Java applet. The browser loads the applet and runs it 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 this:

```
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
```

Most likely, Bob's user agent will display the URL as a hyperlink, which he can activate to launch his browser and retrieve the secondary object (12345.html). If his mail reader doesn't support this function, then Bob must 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 supporting 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. Here's one possible response:

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

If Bob's browser doesn't support the Java environment the applet needs, the browser prompts the user to allow the automatic installation of the enabling software.

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

Sender-stored email delivery follows a more logical economic model, because the sender bears the cost of message storage.

Benefits of sender-stored delivery

Sender-side storage combined with streaming solve the four basic problems we summarized earlier. First, we solved the problem of universal message delivery. Because the sender stores the CM data and only sends a small referencing base message, it's unlikely that the recipient mail system will reject the base message because of storage limitations in its message store. Additionally, most mail readers delegate hyperlink processing to a companion Web browser, so when the base message is an HTML document or contains a hyperlink, the user agent 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 we plan to adaptively stream the CM, senders can deliver CM messages that render without significant startup delays regardless of the recipients' access rates. Recipients behind slow network connections aren't 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 lets a thin client with relatively small local memory render CM messages.

Sender-stored email delivery follows a more logical economic model, because the sender bears the

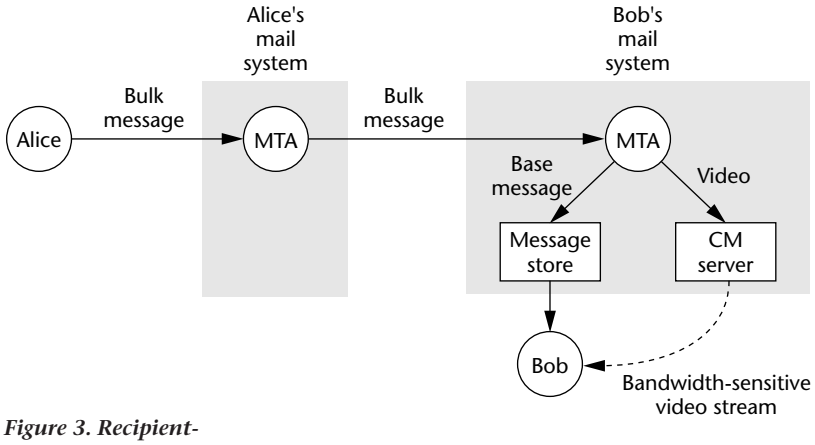

Figure 3. Recipient-stored CM email.

cost of message storage. The recipient only pays for the bandwidth used to transmit the portion of the message data that he chooses to render.

Finally, sender-stored email conserves bandwidth when recipients don't render the message or render only part of the message.

With sender-storage mechanisms in place, recipient systems can opt to accept only small-sized messages into their message stores, 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.

Because a sender-stored deliver architecture represents a major paradigm shift in email practice, it naturally engenders new problems, such as reduced QoS, deletion of stale message data, and the intricacies of forwarding and replying, which we address in the following sections.

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 might be congested. Under congestion, the server can only transmit a highly compressed version of the CM or might be forced to introduce rendering delays to build up a large playback buffer. Therefore, we propose moving the message data closer to the recipient to improve quality. To solve this problem, we introduce recipient-stored delivery of CM email, where the sender's mail system transfers the CM into the recipient system's message store using SMTP, but streamed to the recipient from his message store when 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 most email users.

Figure 3 depicts the recipient-stored delivery process, where the message is transferred in bulk from Alice's user agent 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 recipient's mail system can extract the CM data from the message and give it to a CM server under the control of the recipient's mail system. The message that the recipient's user agent retrieves via post offfice protocol (POP), Internet message access protocol (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's most likely closer and thus less likely to suffer from network congestion, resulting in higher quality playback. An additional benefit is that the power to delete the message is now under the recipient's control.

Our proposed design for pure recipient-stored delivery introduces a filter within the recipient MTA that modifies the push phase of message transport. The system first passes all incoming messages through the filter before entering them into the message store. When the filter finds a MIME body part that contains CM, it extracts the CM data and removes any base-64 encoding. It puts the resulting CM data in storage that's 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 user agent. When the recipient chooses to render the CM message, the CM server streams the message from the recipient's mail system to a media player on the recipient's machine.

We can completely implement recipient-stored delivery within the recipient's mail system without making changes at the sender or recipient user agent. 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.

Integrating recipient and sender-stored delivery

Although we can generally improve QoS with recipient-stored delivery, there are three situations in which using a pure sender-stored delivery approach is desirable. The first is when the recipi-

ent's mail system isn't 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 user agent retrieves the entire message before commencing playback. The second situation is when mailing to a large distribution list where you expect most recipients to render little of the CM content. In this case, streaming only the data that the user requests, rather than pushing it all into each recipient system's message store, conserves bandwidth. The third situation is when the sender addresses a message to multiple recipients within her mail system. If her system uses recipient-stored delivery, it will place a separate copy of the message data in each of the recipients' allocated storage (mailboxes). Since the message isn't subject to modification by the recipients—it's read only—there will be needless duplication. Furthermore, the recipients will retrieve the message data from the sender's mail system, and thus there isn't a QoS improvement with recipient-stored delivery.

We advocate the following mail delivery strategy for a message with a single recipient. If the recipient is local—that is, 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 will stream the CM data to the recipient from its message store. 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 by a client to see if it is CM-aware by using extended SMTP.[6] Alice's MTA can query Bob's MTA with the ESMTP protocol exchange in the following example:

```
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
```

Bob's MTA responds that it supports the extended functionality identified by the keyword CMAWARE.

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. In this case, the sender's MTA delivers a message addressed to all of them 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 message's recipients by the domain name appearing in their email addresses. For the local recipients, use sender-stored delivery. If the number of nonlocal recipient mail systems exceed some threshold, then use sender-stored delivery. Otherwise, query each mail system to see if it's 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 we described earlier, users should be able to specify those sources from which they're 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.

Message deletion

In sender-stored email, the sender (or the sender's system) decides when to delete message content from storage. The recipient would prefer that the CM data referenced by his received base message be available until he deletes it. However, Internet email doesn't currently support a form of storage negotiation between sender and recipient systems that would avoid prematurely deleting sender-stored message content. Thus, sender-stored email systems must rely on nondeterministic methods for deleting 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. For example, she might be unwilling to service all requests for the object if its recipients created many external references by repeatedly forwarding the base message.

When email systems use recipient-stored CM email delivery—and deliver the entire content of the message into the recipient's storage—these problems don't exist. 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- and Web-based user agents, because the remote store can delete CM when users delete its referencing base message. But it doesn't work for POP-based systems, because base messages are kept in local storage

The simplest storage management scheme resembles the ordinary manual management of users' mailboxes.

rather than in remote storage with the CM. When the user deletes a message in local storage, the user agent doesn't inform the remote store of the deletion. Therefore, nondeterministic methods of CM deletion are also relevant for recipient-stored messages 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 the user has deleted his referencing base messages. This protects CM from deletion until all users have deleted known referencing base messages. However, if one of the recipients forwards the base message outside the mail system, the system loses 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, we must use a nondeterministic policy to delete CM.

Manual deletion

The simplest storage management scheme resembles the ordinary manual management of users' mailboxes. Inside the user agent, the sender views her messages arranged into a tree of folders. This includes messages that she received from other users, messages she sent and retained a copy of, and in particular, base messages she sent that refer to CM she created. When she deletes a base message from her mailbox, her mail system also deletes the CM to which it refers, so she controls when the recipient will no longer be able to stream the message data from her CM server.

For manual deletion, the user agent should provide the user with information about the capacity of her mailbox storage and the amount of storage the CM stored in it is using. To make room for new outgoing CM (and incoming messages) the user deletes from her mailbox messages that she considers expendable. The user agent should respond by deleting both the base message and the CM to which the message refers. The system should indicate message sizes in her display, so that she 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 using a sender-stored system, the user only had to make decisions regarding preservation or deletion of the messages she receives. Now she must additionally manage messages that could still be rendered by other users. Users already manage their finite storage resources, and so they might not perceive the added responsibility of deciding which outgoing messages to save and which to delete as inconvenient.

In addition to the senders, the recipients of sender-stored messages must be aware that CM has a limited lifetime. If a recipient of sender-stored CM wishes to access a message's CM at an arbitrary point in the future and doesn't believe the sender will provide it to him at that time, he must copy the CM into storage that's under his control. If the recipient desires to move the CM into his own storage, then the sender's system should provide a lossless mechanism of CM transport so that the recipient can obtain a high-quality copy of the message.

FIFO deletion

Another 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. The email system retains messages for as long as possible, but when it needs room for new content, it deletes messages starting with the oldest until there's enough space for the new content.

The advantage of this approach is that it's automatic, relieving the user of deciding which messages to delete. One problem with the FIFO approach is that it could delete nonrendered messages 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 hasn't 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 nonrendered 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.

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'll have at 2:30 p.m. the same day. Clearly, such a message has a short lifetime. As a contrasting example, suppose Alice sends Bob a description of a product she thinks would be interesting to his company. Alice might 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 Bob requests it later.

To accommodate messages with different lifetime expectancies, the sender can use expiration dates to override the FIFO order of automatic message deletion. Email systems can implement this option as two queues (see Figure 4). Messages initially enter an expiration queue. When they reach their expiration date, the system moves them to an expendable (FIFO) queue. It keeps messages in the expendable queue for as long as possible, but starts deleting the CM content (beginning with the oldest) when it needs space for new content.

Expiration information can be in both human- and machine-readable form. The human-readable expiration date lets recipients manually copy CM message content into their own storage when they desire to retain it beyond its expiration date. The machine-readable form lets recipient mail systems automatically prefetch CM that hasn't yet been rendered—or CM that's referenced by undeleted messages—just prior to its approaching expiration date or to implement other messaging policies that use expiration-date information.

The sender's mail delivery system can add expiration information regarding referenced CM as a header in the base message. For example, if Alice's video data expires on 18 January 1999, the following header could be added to our example base message.

```
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 in the message from links to sender-stored CM. The second component is the time the CM expires.

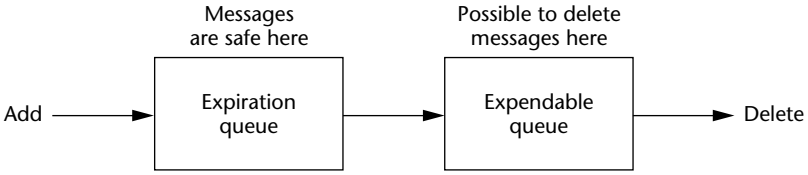

Figure 4. Implementing a message deletion policy with expiration dates and expendable queues.

CM access statistics

CM access statistics can enhance both manual and automatic deletion mechanisms. The sender's mail system can record this access when a recipient accesses a referencing message's CM. Because users can retrieve CM in parts, keeping track of access statistics could become complex. The most detailed record keeping would include a record of each streaming event—the time that a particular byte range within the CM data file was streamed. A less detailed mechanism could mark whether any part of the CM data was streamed.

In the manual deletion approach, the user agent can display the access statistics for CM attached to messages in a list of sent messages. The sender would use these access records to decide whether to delete 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 delivery time. When Alice runs her user agent, 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. Depending on the message's nature, she decides if it has already served its purpose and can be deleted. If she's using an IMAP or Web-based user agent, the remote mail system processes the command to delete the message and thus deletes both the base message and its referenced CM. If she's using a POP-based system, her user agent would only be able to delete the base message from its local storage; deleting the CM from the CM server's remote storage would require a new protocol between the user agent and mail system.

In the automatic deletion system, the recipient's mail system can use the access statistics to order the CM in the expendable queue, so that the oldest CM isn't necessarily the first deleted. CM that has not yet been retrieved could be given higher priority for retention over newer CM that the recipient has already 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 pol-

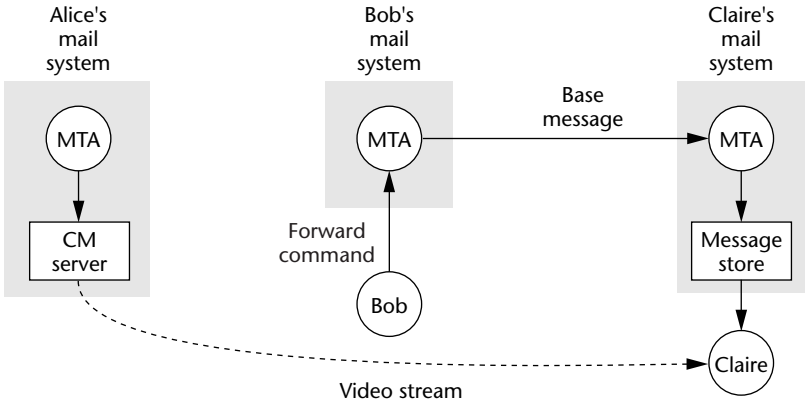

Figure 5. Unreliably forwarding a CM message.

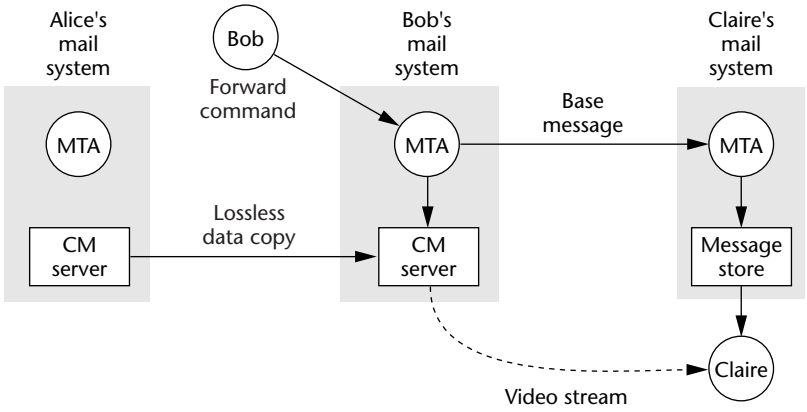

Figure 6. Reliably forwarding a CM message.

icy that tracks recipient access statistics is identifying which recipient is retrieving the message data when a message has multiple recipients or when a recipient has forwarded it. The IP address of the recipient's mail server most likely won't be the same as the system at which the recipient renders the message. When a recipient retrieves message data, the sender's message delivery system can't 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. 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 requests the CM content, 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'll see that only Bob has accessed the message content.

Unique URLs fail when recipients forward messages, because more than one recipient will have a base message with an identical URL. The CM server can't distinguish between an incoming request from the intended recipient or a forwarded party. This problem won't occur if the forwarder makes a copy of the CM in his storage and reconstructs the base message to point to this copy. We call this approach reliable forwarding, which we explain in the next section.

A large percentage of email in a corporate environment is directed toward recipients within the corporation and thus aren't transferred beyond the corporate mail server. In this case, message deletion can be 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. Companies can implement this strategy if they use IMAP or HTTP as the method of mailbox access, because message data remains in the central mail system's domain rather than in the user's local storage as is done under POP. Such a system reduces network traffic and conserves disk space by reducing the amount of redundant static data in the system.

Forwarding

A sender-side storage architecture represents a fundamental paradigm shift in email distribution. Consequently, we must completely rethink the operations of forwarding and replying. For example, suppose Bob has a base message from Alice and he views 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 sends a copy of the base message to the forward recipient, and the forwarder has no control over the referenced CM data. The original author can delete the CM before the forward recipient has a chance to render it. Figure 5 illustrates unreliable forwarding. In this example, Alice sends Bob a sender-stored CM message. When Bob instructs his user agent to forward the base message to Claire, Bob's MTA transfers the base message to

Claire's MTA. Claire retrieves the forwarded base message from her MTA through her user agent and uses the base message to stream the video message from Alice's mail system.

Alternatively, with reliable forwarding the forwarder's MTA first retrieves a copy of the CM, then sends it to the forward recipient using the integrated (sender- and recipient-stored) delivery strategy we described earlier. The approach is reliable because the forwarder now has control over the CM data. Figure 6 illustrates the scenario where Bob's MTA copies the video data into his system's message store 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 could then stream the message data from her own mail system.

It's possible for the user to decide the method of forwarding on a per message basis. Under this manual approach, Bob can estimate the likelihood that the audio data will become inaccessible before Claire tries to render it. His estimate is influenced by the nature of the message, by the expiration date Alice may have specified in the base message, and whether Bob wants to attempt bulk delivery of the message to improve Claire's playback quality.

If Bob's mail service uses an automatic approach, it decides the type of forwarding by comparing the expiration date in Alice's base message with an expiration date Bob specifies. 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 a new base message 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 with a lossless protocol, such as FTP or HTTP, to avoid compounding data loss resulting from streaming protocols. Additionally, senders should provide a reliable transport mechanism for their outgoing CM that lets recipients 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 indefinitely. He believes that Alice will eventually delete the video data, so he instructs his user agent to copy the CM referenced within the base message into his system's message store. When Bob's user agent issues this instruction, his mail service retrieves the message content from Alice's storage using a lossless protocol such as FTP. Bob's mail service then modifies the base message so that it links to the local copy of the video. Note that the message is still delivered to Bob's user agent as a base message, only now it links to CM data as it is stored in his mail system.

Adding new functionality that involves communication between a user agent and mail systems is much easier to accomplish with a Web-based user agent system.

Adding new functionality that involves communication between a user agent and mail systems (as we just described) is much easier to accomplish with a Web-based user agent system, because the user's mail service provides the user agent interface through HTML documents or mobile code. Thus, changes in the behavior of the remote mail service and the user agent interface can be simultaneous. Implementing this new functionality in a standalone user agent system is more difficult, because it requires concurrent changes in two separate applications: the remote MTA and the local user agent.

Replying

Frequently, recipients include original messages, or pieces of original messages, in their replies. It's possible to do the same with CM email. For example, suppose Bob plays Alice's video message by streaming it from her CM server, and he wishes to comment on something specific that Alice says. 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 ends with an additional message of his own. To do this, he uses the repositioning controls available to him, such as a slider bar and rewind and fast-forward buttons. (If his

user agent caches Alice's video message the first time he streams it, replaying parts of her message doesn't generate any additional network traffic.) Because there's no need to deliver data that's already in Alice's storage, Bob's mail system uses a reference to the video data in her mail system rather than storing a second copy of Alice's data.

When Bob issues the command to send his annotated message, his mail system constructs a base message and secondary object with this SMIL[7] file:

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

Notice that the link pointing to the embedded video data references the content stored in Alice's mail system.

Conclusion

We have identified the major weaknesses of Internet email that obstruct CM messaging development. 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 users send email to multiple recipients, and the wasteful delivery of nonrendered message data.

We solve these problems with a sender-stored message delivery architecture, which industry can implement and incrementally deploy with current Web technology. Because this represents a major paradigm shift in existing email practice, it naturally engenders new problems, including reduced QoS, stale message data deletion, and the intricacies of forwarding and replying. To improve QoS in sender-stored delivery, we propose the combined use of both sender and recipient-stored delivery. **MM**

References

1. D. Turner and K. Ross, "Continuous Media E-mail on the Internet: Infrastructure Inadequacies and a Sender-Side Solution," *IEEE Network*, vol. 14, no.4, July/Aug. 2000, pp. 30-37.
2. G. Schürmann, "Multimedia Mail," *Multimedia Systems*, vol. 4, no. 5, Oct. 1996, 281-295.
3. C. Hess, D. Lin, and K. Nahrstedt, "VistaMail: An Integrated Multimedia Mailing System," *IEEE MultiMedia*, vol. 5, no. 4, Oct.–Dec. 1998, pp. 13-23.
4. B. Kervella and V. Gay, "MHEGAM: A Multimedia Messaging System," *IEEE MultiMedia*, vol. 4, no. 4, Oct.–Dec. 1997, pp. 22-29.
5. N. Freed and N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, Internet Eng. Task Force, Network Working Group, Nov. 1996.
6. J. Klensin, N. Freed, and K. Moore, *SMTP Service Extensions for Message Size Declaration*, RFC 1870, Internet Eng. Task Force, Nov, 1995.
7. P. Hoschka, *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*, Synchronized Multimedia Working Group, W3C Recommendation, June 1998.

David Turner is a research assistant in the Multimedia Communications Department at the Eurecom Institute, Sophia Antipolis, France. His current research interests include object-oriented distributed programming, Web technology, and asynchronous multimedia messaging. He has a BS in mathematics from Wichita State University and an MS in mathematics from the University of Massachusetts, and he is a PhD candidate at the Eurecom Institute.

Keith Ross is a professor in the Multimedia Communications Department at the Eurecom Institute. He is coauthor with Jim Kurose of the book *Computer Networking: A Top-Down Approach Featuring the Internet* (Addison-Wesley, 2000). His research interests include multimedia networking, asynchronous learning, Web caching, streaming audio and video, and traffic modeling. He has a BS in electrical engineering from Tufts University, an MS in electrical engineering from Columbia University, and a PhD in operations management from the University of Michigan.

Contact Turner at Eurecom Institute, 2229 route des Cretes, 06904 Sophia Antipolis, France, email david.turner@eurecom.fr.