Asynchronous Multimedia Messaging

David A. Turner
Institut Eurécom
Sophia Antipolis, France

15-Jun-2001

Acknowledgments

My foremost expression of gratitude goes to my thesis advisor Keith Ross for providing me with the opportunity to do research at the Eurecom Institute, and for guiding me over the past three and a half years. Much of my future success will be due in large part to the help that Keith gave to me.

I also acknowledge the contribution of Kim Hyunjung for insisting that I pursue Ph.D. research. Without her influence, I would have probably wondered onto another path and been less fulfilled than I am now.

I would also like to thank the professors, research assistants and fellow Ph.D. students at Eurecom who helped me at various points in time with information and discussions that directly helped me with this research, and assisted me in other ways. For their help with translating parts of my thesis into French, I would like to acknowledge and thank both Philippe de Cuetos and Nathalie Saulquin.

When I first arrived at Eurecom, my discussions with Philipp Hoschka on synchronized multimedia presentations ultimately lead to my research on optimal streaming algorithms. I would like to thank him for this contribution, and also for serving as a member of my jury. I also thank the other members of my jury—Max Mühlhäuser, Andrzej Duda, and Samir Tohme—for spending the long hours necessary to read and evaluate my thesis.

I also thank the administration of Eurecom for their support, and to ENST and the other academic and industrial partners of Eurecom for providing me with the opportunity to accomplish what I did. Many thanks to Arianna Mazzoni for her assistance in the final stages of my thesis writing and defense. Also, thanks to Stephanie Villa for her assistance with many tasks that needed to get done.

Of course, I should also mention the debt to my parents, Albert and Patricia Turner, for giving me all the opportunities I've had in life, including this one.

Résumé

Dans cette thèse nous étudions le problème de délivrer des messages multimédia asynchrones par Internet. Dans un premier temps, nous considérons le problème d'adjoindre des médias continus aux e-mails (comme de l'audio, de la vidéo, des animations, des présentations, etc.), et nous présentons un forum vocal qui est construit comme un service Web. Le ¡¡streaming¿¿ de données enregistrées est essentiel pour transmettre des messages multimédia. Dans un deuxième temps, nous discutons des politiques optimales de transmission dans le cadre du ¡¡streaming¿¿ de données multimédia asynchrones codées en couches.

Dans la première partie, nous démontrons que l'infrastructure actuelle de transmission des messages électroniques ne convient pas au support de données multimédia continues. Pour résoudre les problèmes existants, nous utilisons le stockage des données du côté de l'émetteur et un mécanisme de livraison des données par ¡¡streaming¿¿, qui peut être implémenté progressivement en ajoutant de nouvelles fonctionnalités seulement aux émetteurs. Ce nouveau modèle de stockage et de livraison des données induit plusieurs nouveaux problèmes, dont la réduction de qualité des médias, les réponses aux messages, la suppression et les transferts de messages. Toujours dans cette première partie, nous présentons un prototype de forums vocaux appelé *aconf*, et discutons des contraintes de conception et des alternatives possibles.

Dans la deuxième partie, nous formulons le problème du ¡¡streaming¿¿ de données multimédia préalablement stockées sur un serveur et codées en couches, comme un problème d'optimisation, contraint par la bande passante et la mémoire disponible chez le client. Le terme de présentation multimédia est utilisé pour désigner toute donnée multimédia stockée sur un serveur, incluant les messages. Les présentations multimédias sont un ensemble d'unités de données multimédia ayant des échéances d'exécution relatives au commencement de la présentation. Afin de réduire le délai entre une demande d'exécution et le début de l'exécution de la présentation, l'application effectue le ¡¡streaming¿¿ des données du serveur au client, c'est-

à-dire qu'elle commence l'exécution de la présentation peu de temps après le début de la transmission. Pour réduire le délai initial autant que possible, dans le contexte d'une bande passante limitée, l'application diminue la qualité de la présentation afin que les unités qui la composent arrivent avant leur échéance d'exécution. La diminution de qualité devient également nécessaire lorsque la capacité mémoire du client est relativement faible, ce qui limite la quantité de données qui peuvent être envoyées en avance au client avant le début de leur exécution. Lorsque la capacité mémoire du client est une contrainte, l'application doit déterminer à la fois l'ampleur de la dégradation de qualité, ainsi que le taux de transmission relatif à l'échéance d'exécution de la présentation. La dégradation de qualité est aussi nécessaire lorsqu'il s'agit d'implémenter des mécanismes d'adaptation à la bande passante dans l'Internet. La bande passante des connexions à Internet est en général imprédictible, et par conséquent les applications doivent diminuer la taille de représentation des données lorsque la bande passante diminue, et l'augmenter lorsque la bande passante augmente.

Nous formulons le ¡¡critère max-min affiné¿¿ pour sélectionner une politique optimale de transmission, et la comparons avec un autre critère naturel, appelé le ¡¡critère de qualité totale¿¿. Nous présentons plusieurs algorithmes qui permettent de générer les politiques optimales de transmission à partir d'hypothèses diverses, comme un tampon mémoire limité au client, un média codé en couches, un média codé en couches de manière continue, des images affichées progressivement, et des environnements où la bande passante est imprédictible.

Abstract

This thesis is an engineering study of the problem of providing asynchronous multimedia messaging within the context of the Internet. In part one, we consider the problem of adding continuous media (audio, video, animation, slide show, etc.) to Internet email, and we present a voice message board prototyped as a Web service. Central to asynchronous multimedia messaging systems is the streaming delivery of stored data. In part two, we discuss optimal transmission policies for streaming scalable asynchronous multimedia data.

In part one, we demonstrate the inadequacy of the current email infrastructure for supporting continuous media data. We solve the problems with a sender-side storage and streaming delivery mechanism that can be incrementally implemented in the Internet by adding new functionality only to sender systems. This new storage and delivery model introduces several new problems, including reduced media quality, message deletion, replying, and forwarding. We propose solutions to these problems. Also in part one, we present the prototype voice message board *aconf*, and discuss the design constraints and alternatives.

In part two, we formulate the problem of streaming stored scalable multimedia data as a bandwidth and client memory constrained optimization. We use the term *multimedia presentation* to mean any stored multimedia data, including message data. Multimedia presentations are collections of multimedia data units with rendering intervals relative to the start of the presentation. In order to reduce the delay between user request for playout and the start of playout, the application *streams* the data to the client, that is, it begins playout at the client after commencing transmission of the data. In order to reduce the start up delay as much as possible in the context of limited bandwidth, the application scales down the data representation of the media so that the component units will arrive before their rendering intervals start. Scaling also becomes necessary when client memory capacity is relatively small, limiting the amount of data that can be pre-fetched before their playout intervals commence. When client memory is an issue, the ap-

plication must determine both the magnitude of the scaling, and the rate of transmission relative to the presentation timeline. Scaling is also necessary when implementing bandwidth adaptation schemes in the Internet. Internet bandwidth is not generally predictable, and so applications must scale down the size of the data representation during times when bandwidth decreases, and scale up when bandwidth increases.

We formulate the *refined max-min criterion* for selecting an optimal transmission policy, and compare it with another natural criterion, called the *total quality criterion*. We present several algorithms that generate optimal transmission policies under various assumptions, including a finite client buffer, layer-encoded media, continuously scalable media, progressively rendered images, and unpredictable bandwidth environments.

Contents

List of Figures

List of Tables

Chapter 1

Résumé Complémentaire

1.1 Messages Electroniques Multimedia Continus

1.1.1 Introduction

Bien que les médias continus ne vont peut-être pas remplacer complètement le texte dans les messages électroniques, il y a beaucoup de situations où l'utilisation d'audio ou de vidéo dans les messages est plus préférable que du texte. Cependant, le modèle de stockage et de livraison des messages électroniques dans l'Internet existant pose des problèmes pour supporter les médias continus (audio et vidéo). Nous passons ces problèmes en revue et expliquons comment les surmonter dans le contexte du Web actuel. Cependant, un nouveau modèle de stockage et de livraison basé sur le Web pour les messages électroniques pose un certain nombre de questions à propos de la Qualité du Service (QoS), de suppression de messages, de renvoi et réponses aux messages. Nous identifions ces problèmes et donnons des solutions.

Les problèmes fondamentaux que pose le support des médias continus dans les messages électroniques incluent : (1) un mauvais modèle de coûts, dans lequel le récepteur paye plus de 50 pourcent du coût de livraison du message, (2) la duplication des données du message, (3) la livraison superflue des messages qui ne sont pas complètement ou en partie visualisés par le destinataire, (4) un manque de connaissances de l'environnement de l'utilisateur, tel que la bande passante disponible au récepteur (ce qui induit des délais excessifs lors du transfert de volumineux messages), et une méconnaissance des ressources en stockage de l'utilisateur, ce qui se traduit par des messages indélivrables, et (5) des protocoles d'accès inadéquats, qui ne peuvent pas être adaptés aux médias continus.

1.1.2 Le modèle de livraison avec stockage à l'expéditeur

Pour résoudre les problèmes posés par le support des médias continus dans les messages électroniques, nous proposons un modèle de stockage et de livraison appelé *sender-stored email* [TURN00a] qui repose sur la technologie Web actuelle, et qui prend en compte les besoins et l'hétérogénéité des utilisateurs. Nos propositions sont réalistes, car elles ne requièrent que des modifications incrémentales dans l'infrastructure existante des messageries électroniques. Dans ce mécanisme, le système de messages est responsable du stockage du contenu multimedia de ses messages sortants, qui sont streamés aux agents récepteurs (ou aux players) au moment où l'utilisateur désire visualiser le message. On n'a pas besoin d'envoyer la totalité des données, mais seulement les parties qui sont demandées par le destinataire, et qu'il contrôle par l'intermédiaire de son interface.

La figure 1.1 illustre le processus de livraison avec stockage chez l'expéditeur. Pour mieux comprendre le processus, supposons qu'Alice (A) envoie un message vidéo à Bob (B). Le logiciel de messagerie d'Alice transfère tout le message à son agent de transfert des messages sortants (MTA). Ce MTA place les données multimedia continues dans son serveur multimedia, et le MTA construit un message de base qui donne une référence sur ces données. Le MTA envoie ce message de base au MTA entrant de Bob. Bob récupère le message de base de son système de messages, et l'utilise pour streamer le message vidéo à partir du serveur multimedia du système de messages d'Alice.

Le stockage au niveau de l'expéditeur combiné avec le streaming résout les problèmes fondamentaux des messages électroniques sur Internet. Avec ce système, l'expéditeur supporte maintenant le coût de stockage du message, et le destinataire ne paye que pour la bande passante utilisée lors de la transmission de la partie du message qu'il choisit de visualiser. L'espace disque n'est pas occupé par des données redondantes. La bande passante n'est pas gâchée par la transmission de données multimedia non visualisées, ni l'espace disque par le stockage de telles données. Les destinataires qui ont un espace de stockage de messages limité seront maintenant capables de recevoir des messages multimedia. Les récepteurs qui sont reliés à des connections au réseau lentes ne vont pas subir des délais excessifs avant de recevoir les messages. De plus, le streaming des contenus des messages à partir d'un disque distant permet aux clients avec de faibles capacités de stockage, comme les équipements sans fils, de visualiser des messages multimedia.

Depuis récemment, les 'user agents' des messages électroniques sont devenus compatibles avec la norme MIME[MIME], et sont donc maintenant

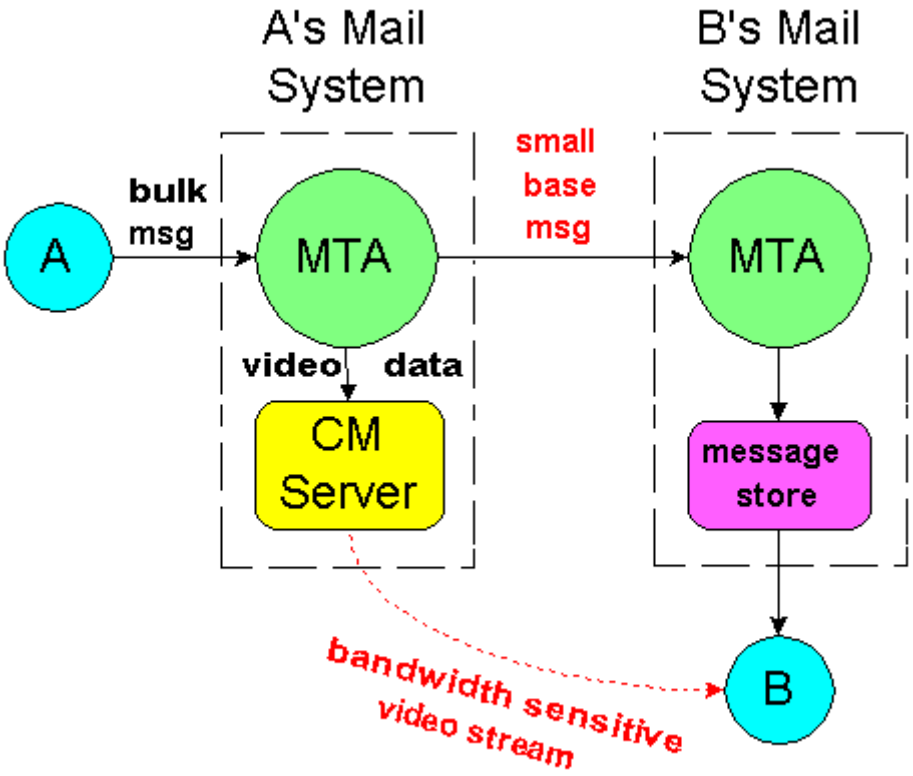

Figure 1.1: Livraison avec stockage chez l'expéditeur

capables de créer des messages incluant du contenu multimedia. En parallèle avec les développements de MIME, il y a aussi eu une intégration accrue entre le 'user agent' et le browser Web. Par exemple, les messageries électroniques cherchent à présent les URLs dans les messages textes, et les présentent en tant qu'hyperliens. Lorsque l'utilisateur clique sur le lien, la messagerie fait démarré un browser Web qui récupère et présente la page référencée. Ces développements permettent une forme de livraison avec stockage chez l'expéditeur de messages multimedia, dans laquelle l'expéditeur envoie un hyperlien qui pointe sur des données multimedia stockées à distance.

Afin de livrer des messages électroniques multimedia à un destinataire arbitraire avec stockage chez l'expéditeur, le message de base doit être traité par un "media player" en dehors de "l'user agent" du destinataire. Avec les "user agents" actuels, le seul moyen d'accomplir cela est d'envoyer un document qui pointe sur un objet secondaire dans le système de messagerie de l'expéditeur. Lorsque le récepteur clique sur le lien, son "user agent"

délègue le traitement du lien à un browser Web. Le browser récupère alors l'objet référencé en envoyant une requête HTTP. Quand le serveur Web de l'expéditeur reçoit la requête HTTP, il sera capable de connaître le type et la version du browser utilisés par le client, et le système d'exploitation sur lequel il fonctionne. Grâce à ces informations, il peut créer une réponse qui sera correcte pour l'environnement du récepteur.

Il y a plusieurs méthodes pour implémenter le play-back de données multimedia au destinataire. Nous décrivons une méthode qui est applicable lorsque le récepteur utilise Internet Explorer. Dans ce cas, l'objet secondaire est un ficher HTML qui contient une référence à un objet de contrôle ActiveX qui fonctionne comme "media player". Si l'objet de contrôle n'est pas déjà présent dans le système du client, il est downloadé automatiquement. Il est alors initialisé pour streamer le contenu multimedia stocké dans le serveur multimedia de l'expéditeur.

Le message de base est représenté sur la figure 1.2.

```
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 1.2: Message de base pointant sur un objet secondaire

"L'user agent" de Bob va afficher l'URL de la figure 1.2 comme un hyperlien, sur lequel il clique pour que son browser récupère l'objet secondaire (12345.html). Lorsque le serveur Web d'Alice reçoit la requête pour l'objet secondaire, il détermine le système d'exploitation et le browser que Bob utilise en lisant les en-têtes appropriés de la requête HTTP. Grâce à ces informations, le système d'Alice retourne un objet secondaire approprié.

Dans cet exemple, nous avons utilisé des URLs courtes, comme 12345.html et 12345.mpg. Mais dans une implémentation réelle, ces URLs devront être de longues chaînes de caractères choisies de manière aléatoire. Un tel mécanisme permettra un niveau de protection équivalent à l'envoi de mots de passe en texte, et permettra aux récepteurs d'accéder à leurs messages à partir d'ordinateurs arbitraires et d'adresses IP arbitraires. Pour avoir une

plus grande securité, il sera nécessaire d'utiliser le cryptage des données et l'identification du récepteur par certificats.

1.1.3 Messagerie avec stockage chez l'expéditeur/destinataire intégré

Livraison avec stockage exclusif au destinaire

Le streaming de données multimedia à partir du système de livraison de l'expéditeur peut résulter d'une Qualité de Service réduite, parce que le réseau entre le serveur de media de l'expéditeur et le récepteur peut être congestionné. Dans ce cas, le serveur ne pourra transmettre qu'une version hautement compressée du media continu, ou peut-être forcé d'introduire des délais de présentation afin de pouvoir remplir un buffer de play-back. Par conséquent, il est souhaitable de déplacer les données du message plus près du récepteur pour améliorer la qualité. Pour résoudre ce problème, nous présentons un mécanisme de livraison de messages multimedia avec stockage chez le destinataire. Dans ce cas, le media continu est transféré dans le disque de stockage du récepteur de manière normale en utilisant SMTP, mais le message est streamé au récepteur à partir de son disque au moment où il choisit de le consulter. Après avoir décrit ce concept en détail, nous proposerons un mécanisme plus flexible, appelé messagerie avec stockage chez l'expéditeur/destinataire intégré.

Le processus de livraison avec stockage au récepteur est presenté sur la figure 1.3, où le message entier est transféré de "l'user agent" d'Alice vers son MTA, qui à son tour transfère tout le message au MTA du récepteur. Une fois que le message arrive au MTA du récepteur, le media continu peut être extrait du message et donné au serveur multimedia sous le contrôle du sytème de messagerie du récepteur. Le message qui est récupéré par "l'user agent" du récepteur via POP, IMAP ou HTTP sera le message de base référençant les données multimedia stockées séparément.

En déplaçant le message jusqu'au disque du récepteur, le media peut être streamé au destinataire à partir d'un endroit qui est plus près de lui. Il y aura ainsi plus de bande passante disponible pour fournir une meilleure qualité. De plus, la suppression du message est à présent sous le contrôle du récepteur, ce qui est un autre avantage.

La livraison avec stockage au destinataire peut être complètement implémentée dans le système de messagerie du récepteur, sans toucher à "l'user agent" de l'émetteur ou du récepteur. Ainsi, un fournisseur de service de messagerie électronique peut implémenter le système sans devoir faire de changement au logiciel du client.

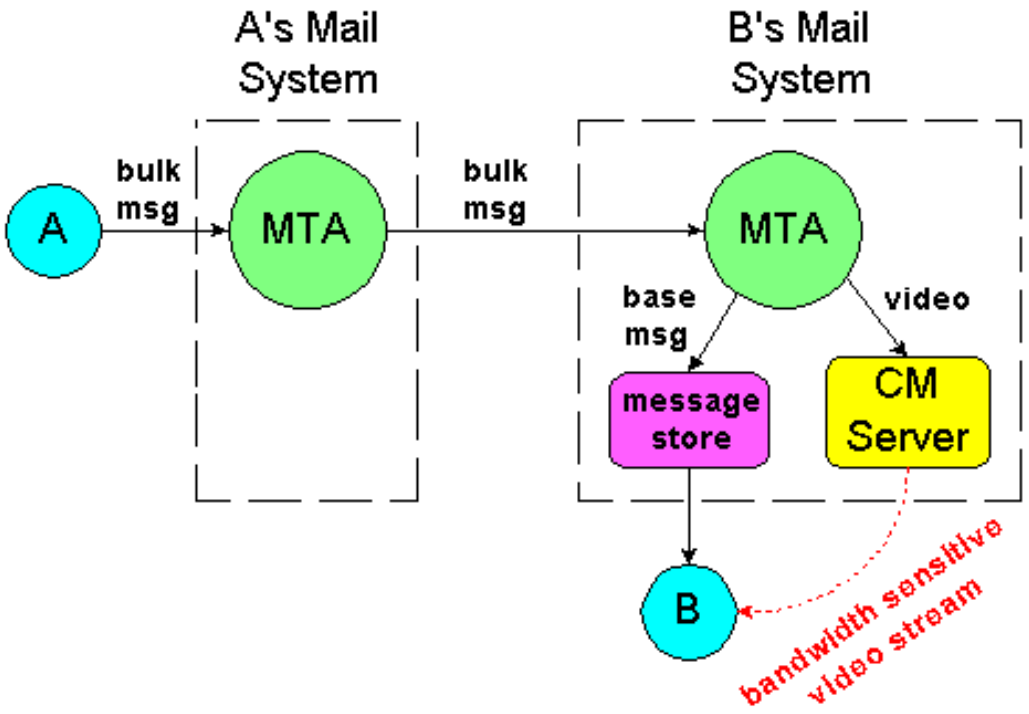

Figure 1.3: Message multimedia stocke au récepteur

Livraison avec stockage chez l'expéditeur et le destinataire

La qualité du service peut généralement être améliorée avec une livraison avec stockage chez le destinataire, il existe trois situations dans lesquelles il est souhaitable d'utiliser une approche basée sur un stockage unique chez l'expéditeur. La 1^{ere} de ces situations est quand le système de mail du destinataire n'est pas capable de générer un flux continu multimédia au destinataire. Dans ce cas, si le destinataire d'un message volumineux est doté d'une connexion à faible débit, il aura à subir d'un long délai pendant que son "user agent" tentera de retrouver l'intégralité du message avant d'entamer tout play-back. La 2^{e} situation est lorsque l'envoi est fait en grand nombre, où la plupart des destinataires s'attendent à rendre très peu du contenu du media continu. Dans ce cas, une large bande passante est conservée en "streamant" seulement les données nécessaires, plutôt que d'obliger toutes ces données à être réceptionnées en bloc par chacun des bénéficiaires. La 3^{e} situation est quand un message est adressé à de multiple destinataires depuis le système mail de l'expéditeur. Si la livraison avec stockage chez le receveur

est utilisée, une copie distincte des données du message sera mise dans la boite mail de chaque receveur. Tant que le message n'est pas sujet à modification par le receveur—il est lecture seulement—il pourra y avoir au besoin duplication des données du message. De plus, les receveurs retrouveront les données du message dans le système mail de l'expéditeur, et ainsi il n'y a aucune amélioration de la qualité en présence d'une livraison avec stockage chez le receveur.

Nous proposons et sommes partisans de la stratégie suivante en matière de livraison de message dans le cadre d'un message personnalisé à destination d'un unique destinataire. Si le destinataire partage avec l'expéditeur le même système mail, alors utiliser la livraison avec stockage chez l'expéditeur. Dans le cas contraire, demander au système mail du receveur s'il "streamera" en continu les données media à ce destinataire. Dans l'affirmative, délivrer alors le message complet à ce système. Dans la négative, alors utiliser une livraison avec stockage chez l'expéditeur afin de garantir une exécution sans délai.

Un système mail peut être interrogé afin de vérifier si continous media-aware en utilisant ESMTP[ESMTP]. Le MTA d'Alice peut ainsi interroger le MTA de Bob grâce au protocole d'échange ESMTP illustré figure 1.4. Le MTA de Bob répond qu'il supporte une fonction étendue identifiée sous le nom de "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 1.4: le serveur ESMTP déclare que c'est "CMAWARE"

Dans le cadre d'un message avec plusieurs destinataires, nous proposons une stratégie de livraison un peu plus compliquée. Quand un groupe de destinataires partage dans leurs adresses e-mails un nom de domaine commun, cela signifie qu'ils utilisent le même MTA et qu'un message adressé à tous peut alors être remis sans un message de transfert unique SMTP. Ainsi, le coût de la bande passante pour l'envoi d'un message volumineux à de nombreux destinataires qui partagent le même système mail équivaut au coût d'un envoi de ce message à l'un seul d'entre eux. Pour cette raison, nous regroupons les destinataires du message par nom de domaine apparaissant dans leur adresse e-mail. Pour les destinataires en local, utiliser la livraison avec stockage chez l'expéditeur. Si le nombre de systèmes mails des desti-

nataires non locaux excède un certain nombre, alors utiliser la livraison avec stockage chez l'envoyeur. Dans le cas contraire, demander à chaque système mail si c'est un continous media-aware. Dans l'affirmative, alors délivrer le message complet au système. Dans la négative, alors utiliser une livraison avec stockage chez l'envoyeur.

1.1.4 Suppression de Message

Dans le cadre d'un e-mail stocké chez l'expéditeur, celui-ci décide du moment où il veut supprimer un message dans sa boite de stockage. Toutefois, le destinataire souhaite que le référencement des données media en continu par la base message soit possible avant qu'il ne les supprime de sa boite message. Malheureusement, l'e-mail Internet ne s'appuie pas souvent sous la forme d'une négociation de stockage entre les systèmes de l'expéditeur et du receveur qui pourrait être utilisée afin d'éviter une suppression prématurée du contenu du message dans le stockage de l'envoyeur. Ainsi, le système de stockage e-mail de l'expéditeur doit compter sur les méthodes non déterministes pour supprimer les données.

Nous identifions un nombre de stratégies différentes de gestion du stock de messages pour les systèmes e-mails de stockage chez l'expéditeur, et nous examinons comment ils devraient comporter sous de divers scénarios.

Suppression Manuelle

Le schéma le plus simple de gestion du stockage est identique à la gestion habituelle de la boite mail de l'utilisateur. A l'intérieur du "user agent", l'expéditeur voit les messages ranger dans des dossiers. Dans ces messages sont compris les messages réceptionnés par d'autres utilisateurs, et la base message qu'elle a envoyée se réfère au media continu qu'elle a crée.

La taille du message devrait être indiquée dans l'affichage pour qu'ainsi l'utilisateur connaisse l'impact de chaque message sur sa répartition du stock. Quand elle supprime la base message de sa boite mail, son système mail supprime aussi le référencement du media continu.

Pour créer un espace pour un nouveau media sortant en continu (et messages entrants), l'utilisateur supprime ces messages dont il n'a plus besoin.

Suppression FIFO

Une approche de la suppression des messages est une simple file "1^{er} entré/ 1^{er} sorti" des données media continu. A travers cette approche, l'expéditeur a un niveau déterminé de réserve de stockage pour le contenu du media

continu. Les messages sont conservés aussi longtemps que cela est possible, mais quand un espace est requis pour un nouveau message, ils suppriment tout d'abord le plus vieux.

L'avantage de cette approche est son automatisme. Le problème est que l'approche FIFO en matière de suppression de message rend cela possible pour les messages non rendus devant être supprimés avant les messages rendus.

Suppression de la Date d'Expiration

Afin de gérer les messages possédant des durées de vies différentes, les dates d'expiration peuvent être utilisées avant la suppression automatique des messages dans l'ordre FIFO. Cela peut être implémenté en deux "queues" voir figure 1.5. Les messages entrent dans le commencement d'une file d'attente d'expiration. Quand leur date d'expiration est atteinte, ils sont alors transférés vers une file (FIFO) non cruciale. Le système conserve les messages dans la file "non cruciale" aussi longtemps que possible. Mais quand de l'espace est nécessaire pour de nouvelles données, le media continu est supprimé de la file non cruciale.

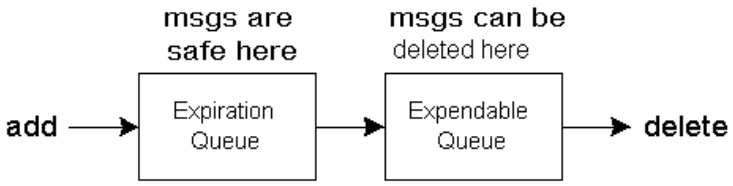

Figure 1.5: Queues de durée de vie du message

L'expiration de l'information peut être donnée sous une forme lisible par l'homme, et sous une forme lisible par la machine. Cela permet aux receveurs de copier manuellement le contenu du message media continu dans leur propre stock quand ils désirent les conserver après leur date d'expiration. Cela permet également aux systèmes mails des destinataires de "pre-fetch" automatiquement le media continu qui n'aurait pas encore été restitué avant sa date d'expiration.

Des dates d'expiration peuvent être ajoutées dans la base message. A titre d'exemple, si les données vidéo d'Alice expirent le 18 Février 2000, l'entête suivant pourra être ajouté à la base message :

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

L'entête d'expiration comprend deux composantes. La première composante identifie le lien qui pointe sur le second objet, lequel est nécessaire pour distinguer les liens ordinaires qui peuvent être une partie du message des liens du media continu de stockage de l'expéditeur. La seconde composante de l'entête est la date à laquelle expirera le media continu.

Statistiques Accessibles au Media Continu

Des mécanismes de suppression à la fois manuels et automatiques peuvent être accrus par l'utilisation du "continous media access statistics." Lorsqu'un receveur accède au media continu d'une base message, le système mail de l'expéditeur peut effectuer un enregistrement de cet accès.

Dans l'approche manuelle de la suppression, le "user agent" montre l'accès aux statistiques au media continu de la base message qu'elle a envoyée. La personne qui le souhaiterait, pourrait utiliser cet accès aux statistiques afin de décider de supprimer ou sauvegarder un message particulier du media continu.

Dans l'approche de la suppression automatique, l'accès statistiques peut être utilisé afin de ranger le media continu dans une file non cruciale. Ainsi le plus ancien media continu n'est pas obligatoirement le premier à être supprimé. Le media continu qui n'aurait pas encore été retrouvé donne une priorité plus importante pour sauvegarder le tout nouveau media continu qui aurait déjà été accessible au receveur.

1.1.5 Transmission et Réponse

Transmission

Les utilisateurs doivent choisir entre deux formes de transmission sécurisée et non sécurisée. Dans une transmission non sécurisée, le système de l'utilisateur envoie une copie de la base message au receveur du message transmis. Cette approche n'est pas sécurisée dans la mesure où la personne qui transmet n'a aucun contrôle sur l'existence de données référencées dans le media continu. Il est possible que l'auteur original du message supprime le media continu avant que le receveur du message transmis ait une chance de le rendre. Le schéma 1.6 illustre la transmission non sécurisée. Dans cet exemple, Alice a la première transmis à Bob un message media continu à stockage chez l'expéditeur. Quand Bob demande à son "user agent" de transmettre la base message à Claire (C), le MTA de Bob transfère la base message au MTA de Claire. Claire retrouve la base message transmise depuis son MTA grâce à

son "user agent", et utilise la base message pour streamer le message vidéo depuis le système mail d'Alice.

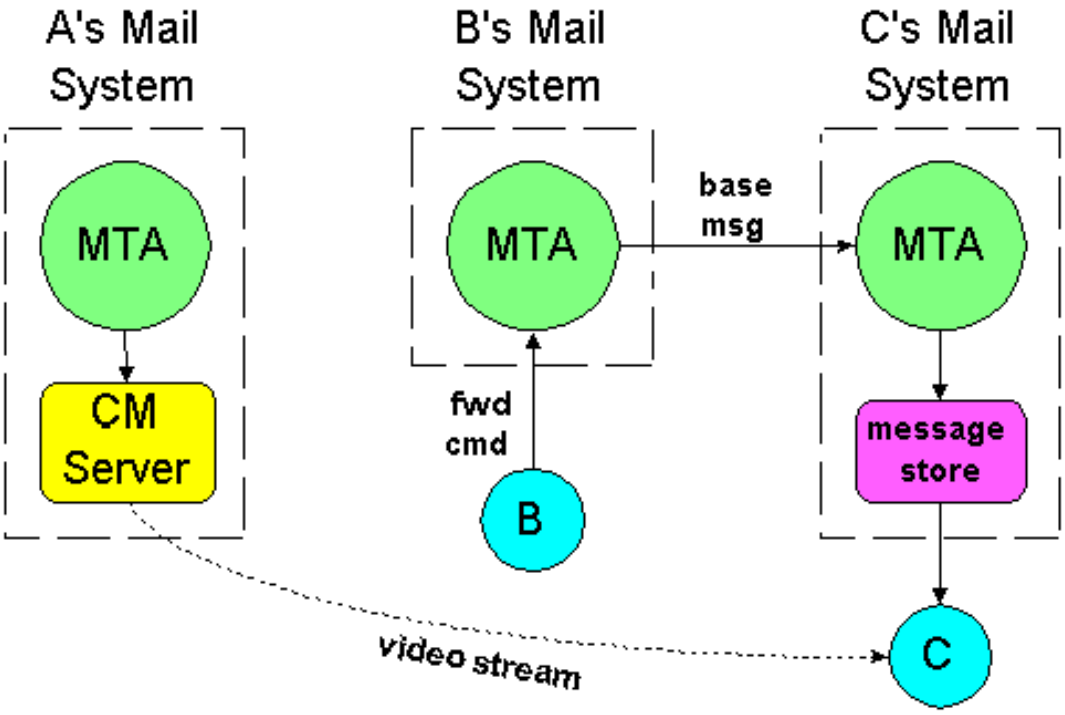

Figure 1.6: Transmission non sécurisée

De façon alternative, avec une transmission sécurisée, le MTA de la personne transmise retrouve tous les media continus, et ensuite les envoie au receveur suivant utilisant la stratégie de la livraison décrite plus haut. L'approche est sécurisée dans la mesure où la personne qui transmet a le contrôle pendant toute la durée de vie des données du media continu. Le schéma 1.7 illustre le scénario dans lequel le système de Bob (MTA B) copie les données vidéos dans le stock de son système, et délivre une base message à Claire. Claire retrouve, quant à elle, la base message dans sa boite mail et streame la vidéo depuis le serveur de media continu de Bob.

Il est possible pour un utilisateur de choisir le mode de transmission par un message de base. Avec cette approche manuelle, Bob évalue la probabilité que les données vidéos deviennent inaccessibles avant que Claire essaie de les rendre. Son estimation est influencée par la nature du message, par la date d'expiration spécifiée par Alice dans la base message.

Si une approche automatique est utilisée, le service mail de Bob décide du type de transmission en comparant la date d'expiration placée dans la

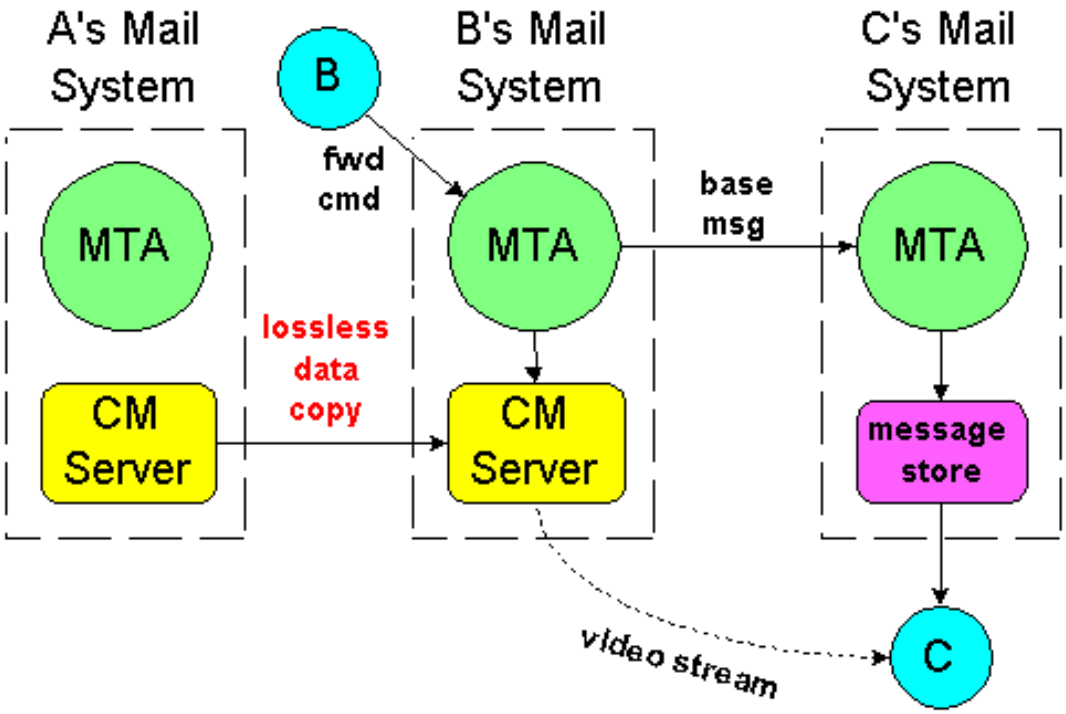

Figure 1.7: Transmission sécurisée

base message par Alice à celle spécifiée par Bob. Si la date d'expiration de Bob est plus tôt que celle d'Alice, alors le service mail de Bob transmet au système de Claire la base message originale avec un réfèrencement du media continu stocké dans le système mail d'Alice. Mais, si la date d'expiration de Bob est postérieure à celle d'Alice, alors le système mail de Bob copiera les données vidéos dans son stock et le délivrera depuis ce dernier à Claire.

Renvoi

Fréquemment, les receveurs incluent le message original ou des parties de celui-ci dans leurs réponses. Il est possible de faire la même chose avec l'e-mail d'un media continu. Regardons un exemple. Supposons que Bob regarde le message vidéo d'Alice en le streamant depuis le serveur media continu d'Alice. Il souhaite apporter quelque chose de spécifique à ce que dit Alice. Il élabore une réponse vidéo dans laquelle il commence par parler, ensuite y insère la section du message vidéo d'Alice qu'il souhaite citer, et enfin termine le message avec quelques propos supplémentaires de lui-même. Pour réaliser cela, il utilise des contrôles de repositionnement qui lui sont disponibles, comme la barre du curseur, rembobine et aussitôt transmet les

boutons. En une fois, il localise l'endroit exact dans la vidéo d'Alice où il désire intervenir, il clique sur le bouton "départ de copie" pour commencer la capture vidéo dans le système de clipboard. Lorsque la vidéo tend à la fin de ce qu'il désire citer, il clique sur "arrêter la copie", et alors le clipboard contient la partie vidéo d'Alice qu'il souhaite adjoindre à son message de réponse. A présent, Bob a un message vidéo avec une video quote encastré. C'est parce qu'il n'y a pas besoin de livrer les données qu'il est déjà dans le stockage d'Alice, une référence des données vidéos de son système mail est utilisée, plutôt que d'accumuler une deuxième copie des données d'Alice dans le système mail de Bob.

Lorsque Bob lance la commande "envoyer le message", son système mail construit une base message et un objet secondaire avec le fichier SMIL décrit en figure 1.8. Notez que le lien pointant sur les données vidéos insérées est un élément de référence dans le stockage du système mail d'Alice.

```
<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 1.8: Un objet secondaire avec le fichier SMIL

1.2 Administration de Message Audio

1.2.1 Introduction

Il existe généralement un certain nombre de formes différentes de conférences asynchrones sur le Net. Les "mailing" listes en sont une forme, où les participants soumettent l'e-mail au serveur central, qui compile ces commentaires en un unique message et le dispatche aux participants. Les "News groups" sont une autre forme de conférences asynchrones, où les participants mettent leurs messages dans une arborescence de messages. Les "News groups" se retrouvent actuellement sous deux formes différentes. D'une part, il y a les "News groups" traditionnels, basés sur le Network News Transfer Protocol (NNTP), et d'autre part, les "News groups" de type Web, qui eux utilisent un accès à travers leur navigateur.

Ces systèmes de messages asynchrones sont désormais basés sur du texte. En effet, l'audio n'est pas encore largement utilisée en matière de conférences

asynchrones sur le Net. De toute manière, parce que les systèmes informatiques dotés de playback audio et que des possibilités de capture se répandent, il est désormais faisable d'ajouter de l'audio à une conférence asynchrone. Afin de mieux comprendre les réalisations techniques impliquant ce type de développement, et commencer une expérimentation, nous avons développé un prototype audio basé sur le système de conférence asynchrone dénommé "aconf".

1.2.2 Les Objectifs des Design

Le principe que nous avons suivi pendant le "design" de notre système prototype était de faire cela gratuitement pour toute installation d'un logiciel spécial, incluant d'aller sur un site Web pour télécharger et installer un "plug-in". Si l'utilisateur avait des enceintes et un micro reliés à son système, alors il devenait possible pour un utilisateur d'écouter ses messages, et de créer des messages sans avoir à réaliser une procédure d'installation compliquée.

Nous étions capables d'atteindre ce but dans des limites acceptables. En premier lieu, l'utilisateur va dans une conférence, il est présenté dans une fenêtre qui lui demande s'il accepte le contrôle ActiveX avec validation de la signature électronique. Si l'utilisateur clique sur "OK", le contrôle ActiveX est alors installé. L'utilisateur n'aura plus ensuite qu'à répondre à cette question. Dans toute conférence où il se rendra, sur tout site Web, il y aura une transparence interactive quant au contrôle installé.

Dans un deuxième temps, nous voulions que le système soit accessible au plus grand nombre, en d'autres termes être utilisable par tout "browser" et par tout système opérationnel. Depuis que Java est une plate-forme indépendante, nous avons bien entendu réfléchi sur le développement côté client d'un système avec applet Java. Quoiqu'il en soit, au moment où nous développions le prototype, Java ne fournissait pas de développeurs avec une fonctionnalité de capture audio. Une telle fonctionnalité est désormais disponible avec la version 1.3 de Java, mais elle n'est pas encore avec l'API de Java montré par les "browsers" d'applets Java. De toute manière, si les utilisateurs installent le "plug-in" de Java 1.3, un applet peut être utilisé afin d'effectuer une capture audio et un playback.

Parce que la solution d'une plate-forme indépendante n'était pas réalisable (et le demeure), nous avons réalisé une solution avec une plate-forme spécifique chez le client. Nous avons choisi d'utiliser un contrôle ActiveX pour fournir une capture audio et une soumission des données au serveur, ainsi une conférence réduite est accessible aux utilisateurs d'Internet Explorer fonctionnant sous le système d'exploitation Windows.

1.2.3 La Page Web du Client

Le format standard d'une conférence est une hiérarchie de messages. Ces messages sont soit placés à un niveau élevé soit sous le message parent. Les messages sont présentés par des lignes contenant le titre du message, son auteur, et d'autres informations telles que la date par exemple. Ces lignes sont ensuite présentées dans une liste verticale où les messages enfants sont "indented" sous leurs parents. Une boite de tri apparaît à gauche du message, sur laquelle l'utilisateur clique pour étendre ou "colapse" l'enfant d'un message. C'est l'interface standard pour voir le contenu d'une collection d'objets classés de manière hiérarchique.

Quand on clique sur le titre d'un message, le texte du message apparaît dans une boite texte en bas de l'écran et le "player" média du système retrouve et joue les composantes audio du messages (fig 1.9). L'utilisateur contrôle le playback audio grâce aux contrôles playback du "player" média.

Sous le dernier message enfant, il y a un lien "ajouter des commentaires". L'utilisateur clique sur ce lien pour insérer un nouveau message à cet instant de l'arborescence. Après ce click sur ce lien, l'arborescence du message est remplacée par la création d'un message d'interface (fig. 1.10). Cette nouvelle fenêtre comprend des boites textes pour le non utilisateur, le titre et le texte du message. Il y aura également des contrôles de capture audio sous forme de boutons. L'utilisateur clique sur ENREGISTRER pour commencer la capture audio, STOP pour arrêter cette capture, JOUER pour entendre les contenus de la capture buffer et EFFACER pour effacer les contenus de la capture buffer. Il peut aussi cliquer sur STOP pour arrêter le playback et ENREGISTRER des données à ajouter dans l'espace disponible. Si l'utilisateur est satisfait de son message, il clique sur ENVOYER pour soumettre le message au serveur de la conférence. S'il décide de ne pas soumettre son message, il clique sur le bouton ANNULER.

1.2.4 Le Serveur de la Conférence

Les données de la conférence sont organisées sous une seule directory ; dans notre implémentation, nous appelions la directory "aconf". Dans le haut de la directory, nous plaçons tous les fichiers communs à toutes les conférences et les sous-directories qui contiennent des fichiers pour des conférences spécifiques. Les fichiers du haut du système de la directory incluent :

- aconf.cab — un fichier désigné cab contenant aconf.ocx, qui est un contrôle ActiveX.

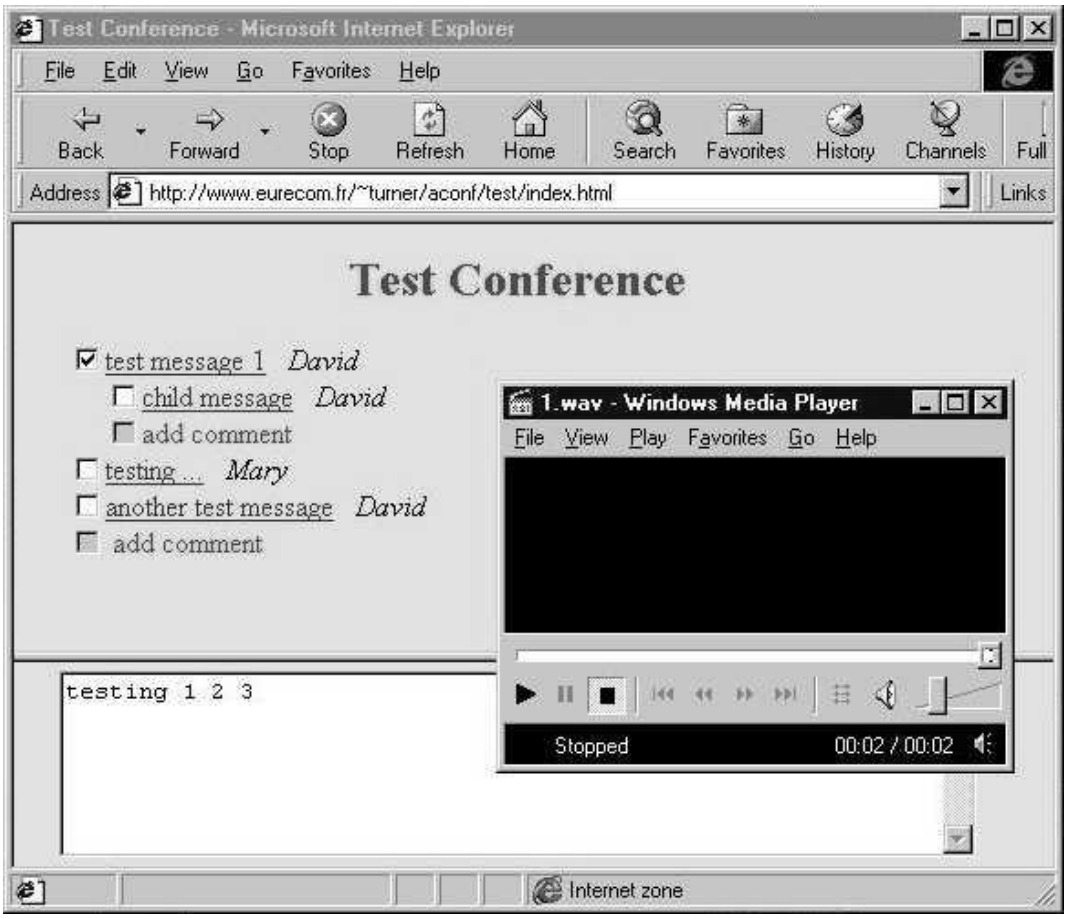

Figure 1.9: L'arborescence du message

- AconfApplet.class — un applet Java dont le rôle est de retrouver les données textuelles lors de l'initialisation de la page Web.
- aconf.css — une feuille de style en cascade [HTTP] utilisée pour contrôle le format de toute la conférence.
- aconf.js — un fichier de fonctions Javascript qui coordonne la séparation des éléments du système. La fonction Javascript "onload" dans ce fichier interagit avec l'applet Java afin de retrouver le message textuel stocké dans le serveur, utilisé pour construire l'arborescence de message.
- TextBox.html — un document HTML qui définit la zone de texte pour apparaître dans le frame le plus bas de la page Web.
- Les sous-directories de conférence, qui contiennent ce qui suit :

Figure 1.10: Interface pour créer de message

- index.html — un document HTML qui contrôle le "layout" de la page Web.
- msgTree.html — un document HTML qui définit les éléments qui apparaissent dans le frame supérieur de la page Web.
- params.scr — un fichier contenant une variable Javascript de conférence spécifique.

- data.txt — un fichier texte qui contient des données textuelles pour l'ensemble des messages de la conférence.
- 1.wav, 2.wav, 3.wav, etc. — les fichiers audio.

Le système requiert un serveur Web qui pourra à tout moment être interrogé pour délivrer n'importe quel fichier identifié dans la liste précédente. En plus du serveur Web, un serveur de conférence (écrit en Java) écoute pour une connexion TCP les requêtes sur un nombre prédéterminé de quelques ports. Ce nombre de ports est configuré dans le fichier Javascript aconf.js, et est passé en un contrôle ActiveX durant l'initialisation de la page web. Le travail du serveur conférence est d'accepter de nouveaux messages de clients et de les ajouter à l'arborescence de message public. Il réalise cela en deux étapes. En premier lieu, il assigne un numéro de message au nouveau message et sauvegarde les données audio comme fichier avec un nom identique au numéro de message. En second lieu, il ajoute les données textuelles du nouveau message au fichier data.txt.

1.2.5 Initialisation d'une Page Web

L'utilisateur entre dans une conférence en chargeant le fichier index.html dans la directory de la conférence, il en résulte une séquence d'initialisation qui implique le chargement de fichiers ayant des supports divers, l'installation d'un contrôle ActiveX, le fonctionnement d'un applet Java, et l'exécution de Javascript qui construit l'arborescence du message. La fig 1.11 illustre les composantes impliquées dans cette initialisation. Quand le "browser" charge le top-niveau du message HTML, il retrouve les documents de référence qui sont intégrés dans le cadre supérieur. Le cadre le plus bas, la feuille de style, le fichier Javascript, le contrôle ActiveX et l'applet java. Le "browser" construit l'interface utilisateur et active la fonction Javascript référencée dans l'attribut "onload" dans le corps du tag du frame supérieur.

Le contrôle ActiveX est référencé avec le tag HTML <OBJECT> comme suit :

```
<OBJECT id="Aconf" classid="CLSID:996F6602-7895-11d2-8F18-
    006008644547" codebase="http://www.eurecom.fr/~turner/aconf/
    aconf.cab#Version=1,0,0,4"></OBJECT>
```

L'attribut "classid" est un identificateur globalement seul (GUID) que Windows utilise pour identifier des composants. Quand le "browser" rencontre cet élément, il cherche le registre système afin d'évaluer si la version courante de ce modèle est installée. S'il ne découvre une telle composante avec ce GUID existant, alors il retrouvera le fichier cab et installera

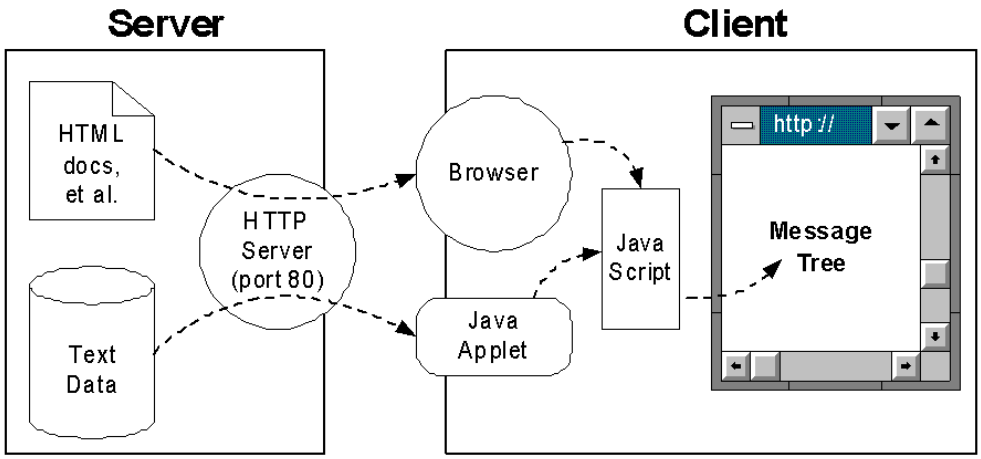

Figure 1.11: Initialization du page Web

le contrôle. Quand il réalise cela, le "browser" ouvre une fenêtre montrant que le site web demande l'installation d'un contrôle ActiveX et que ce contrôle aura été authentifié par un certificat de validation émis par David A. Turner. Si l'utilisateur clique sur le bouton OK pour poursuivre, le système de l'utilisateur extrait le fichier aconf.ocx du fichier cab, et le copie dans une sous- directory sous le contrôle du "browser".

Le contenu du fichier CAB est authentifié par une clé privée fournie par Verisign, et un certificat Verisign est inclus avec une clé publique que le "browser" de l'utilisateur utilise pour vérifier authenticité du code du signataire. Lorsque le "browser" de l'utilisateur charge le contrôle ActiveX pour la première fois, il fait apparaître une fenêtre attirant l'attention sur la sécurité, fenêtre dans laquelle le certificat est en attente de validation et de signature par David A. Turner. On demande à l'utilisateur s'il accepte qu'un code soit installé sur sa machine. S'il clique sur OK, le "browser" met un contrôle ActiveX dans la sous- directory gérée par lui-même, puis appelle la fonction d'installation du contrôle.

L'initialisation du code Javascript commence en mettre l'attribute "on-

load" de l'élément <BODY>, comme dans <BODY onload="buildMessageTree()" ...>. Cette fonction ajoute un élément HTML qui comprend l'arborescence du message. Avant qu'il ne commence à ajouter des éléments, et quoiqu'il en soit, il fait appel tout d'abord à la méthode "loaddata" de l'applet Java pour retrouver les données textuelles pour la conférence, ce qui est réalisé en faisant une requête GET au serveur web. Quand cet appel revient, le script entre une "loop" dans laquelle il est ajouté deux sections <DIV> pour chaque message dans l'arborescence du message.

1.2.6 Interface Utilisateur

Quand le script d'initialisation est entier, le client aura toutes les données textuelles, notamment le titre du message, le nom de son auteur et le texte du message. Ensuite, les données audio resteront dans le serveur. Quand l'utilisateur souhaitera écouter le message, il cliquera sur le titre du message, lequel possède un lien. Le titre du message pour le message 1 de la conférence test a la forme suivante :

<A href="1.wav" onclick="showtext('1')"> test message 1 </A>

En cliquant sur ce lien alors le "browser" doit faire deux choses. La première, exécuter la fonction Javascript "showtext()" qui met le texte du message dans la boite texte du cadre le plus bas. La seconde, demander au player du système media de retrouver puis restituer le fichier "wave" référencé par l'attribut "href". La figure 1.9 montre un résultat possible en cliquant sur ce lien.

Quand l'utilisateur désire étendre une branche de l'arborescence de messages, il clique sur la boite de tri située à côté du message. Considérons le premier message de la conférence test de l'exemple fig 1.9. La boite de tri provient du tag HTML suivant :

<INPUT type="checkbox" onclick="expand('1')>

Le click effectué, le "browser" appelle la fonction Javascript "expand()" avec le numéro de message en argument. La fonction "expand()" regarde si les enfants du message en cours ont l'attribut display dans leur tag enclu <DIV> sur "aucun", ce qui les rend invisibles, ou sur "bloquer" ce qui les rend alors visibles. S'il trouve que les enfants sont invisibles, cela change leurs attributs display en "bloquer". S'il se trouve que les enfants sont visibles, cela change leurs attributs en "aucun" et il en est de même pour les descendants des enfants. Après la réalisation de cette opération, le "browser" donne la nouvelle version du document HTML.

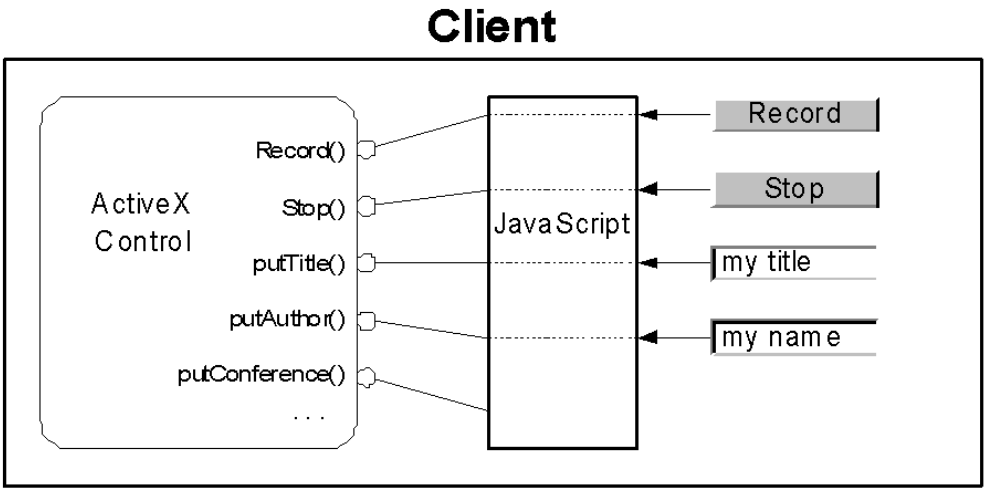

Figure 1.12: Message data and audio capture functions mapped to ActiveX control by JavaScript

Si l'utilisateur souhaite ajouter un message, il clique sur "ajouter un commentaire", lien qui est positionné dans l'arborescence là où il veut insérer un nouveau message. L'ajout d'une phrase de commentaire qui ajoute le message 1 au message enfant est rendu avec le HTML suivant :

<SPAN onclick="addComment('1')" style="cursor:hand; color:red">
add comment </SPAN>

Quand l'utilisateur clique sur le texte, la fonction "ajout commentaire ()" est appelée, elle fait partie des autres fonctions Javascript contenues dans le fichier script msgtree.scr. Cette fonction met à "none" l'attribut affiché du tag <DIV> contenant l'arborescence de messages et elle met à "block" l'attribut affiché du tag <DIV> contenant les fonctions de création de message. Cette transition de l'arborescence de messages à création de message (figure 1.9 to figure 1.10) est instantanée, parce que le browser n'a pas besoin de rechercher d'autre donneés sur le réseaux.

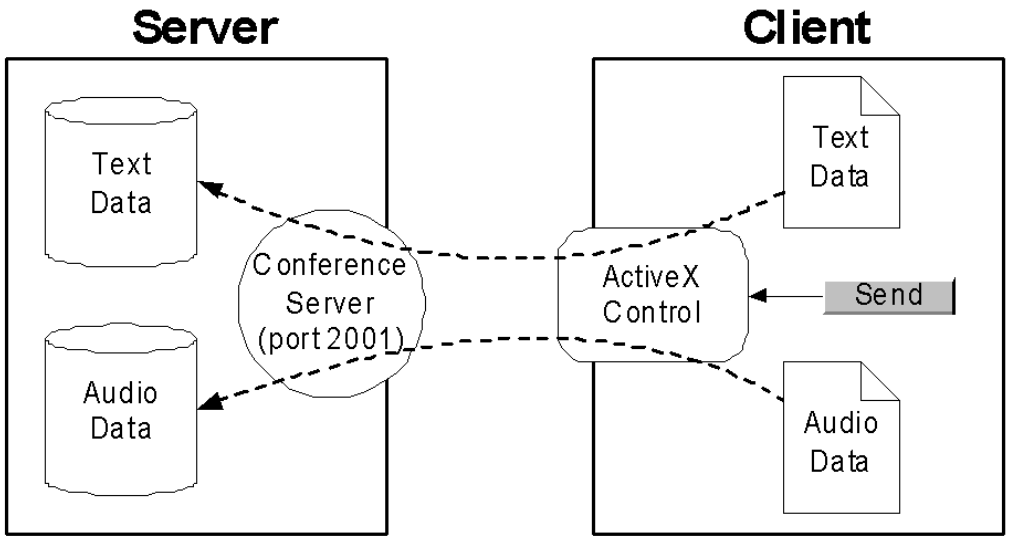

Figure 1.13: Le contrôle ActiveX envoie un nouveau message au serveur de conférence

A ce moment, l'utilisateur voit des cadres des textes pour un auteur, titre de message, et le texte de message. En outre, il y a des boutons pour contrôler la capture audio et la soumission de message. Cliquer sur n'importe lequel de ces boutons a comme conséquence l'exécution d'une fonction de JavaScript pour accomplir la tâche, ce qu'il accomplit en appelant les méthodes associées au contrôle ActiveX. Le processus de tracer des événements de page Web produits par l'utilisateur au contrôle ActiveX est illustré dans la figure 1.12.

Ensuite, l'utilisateur complète les zones de titre, de nom et de message, et sur option des textes supplémentaires, il enregistre un message. Quand il est satisfait de l'enregistrement, il clique sur la fonction ENVOI, qui passe l'exécution vers la fonction de JavaScript "send" . Cette fonction transfère les données des textes à partir des cadres des textes HTML au contrôle ActiveX , et appelle alors la méthode "send" du contrôle. La commande établit une connexion TCP avec le serveur de conférence, et soumet le nouveau message. Ce processus est illustré dans le figure 1.13.

1.3 Streaming Optimal de Présentations Encodées en Couches

1.3.1 Introduction

Généralement, les systèmes distribués multimédia délivrent un contenu (des présentations multimédia) au travers de chemins du réseau ayant une capacité limitée. Ils sont aussi typiquement contraints par un délai minimum de playback initial et une capacité mémoire limité chez le client. Afin de tenir compte de ces contraintes, le système de livraison dégrade la qualité des éléments qui constituent la présentation, afin de réduire la taille de représentation des données.

Un grand nombre des systèmes qui ont été proposés et implémentés pour le streaming de présentations multimédia, essayent d'adapter leur flux multimédia dans le contexte d'une bande passante fluctuante. Cependant, il y a une très forte certitude que la bande passante IP soit fortement ératique [PAX97a]. Dans ces conditions, même les mécanismes adaptatifs échouent à des moments imprévisibles et pour des durées imprévisibles. Parce que les utilisateurs désirent que les flux multimédia leur soient livrés de manière régulière et en des délais courts, il y a un intérêt croissant dans l'évolution de l'Internet actuel en un Internet qui serait capable de garantir une Qualité de Service des flux multimédia.

En particulier, il y a eu beaucoup de travaux à propos d'IntServ [RFC2212] et de DiffServ [RFC2475]. Il est probable que l'une ou l'autre de ces technologies permettra de pouvoir négocier des canaux de transmission à bande passante fixe pour la durée d'une présentation multimédia. Nous commençons par résoudre le problème d'échelonnage optimal d'une présentation multimédia dans le contexte d'un tel environnement. Nous étendons alors nos solutions à bande passante fixe aux réseaux qui n'ont pas de qualité de service fiable.

Lorsque le réseau ne peut pas fournir assez de bande passante pour transmettre la meilleure résolution de la présentation, ou si la mémoire du client est inadéquate pour stocker des données en avance, l'application doit réduire la taille de représentation afin que les données soient délivrées à temps et affichées de manière régulière. Lorsque les données multimédia sont codées en couches, l'application peut s'adapter à la bande passante disponible en envoyant plus ou moins de couches, ce qui résulte en une qualité plus ou moins élevée.

Un média encodé en couches est qualifié de *progressif*, dans le sens où la représentation des données est ordonnée. Une application peut utiliser l'unité

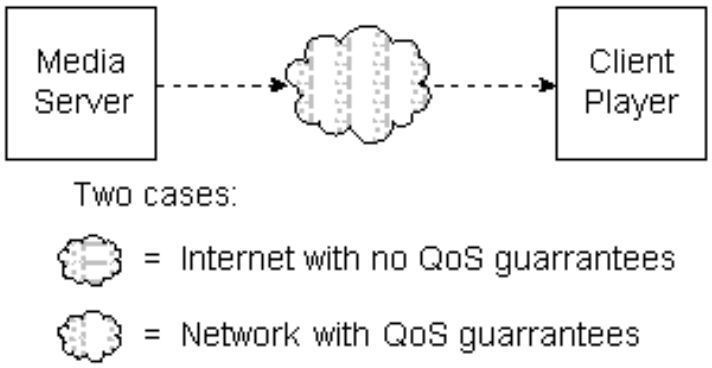

Figure 1.14: Environnements de réseau

de données ℓ seulement si elle est combinée avec les unités $1, \ldots, \ell - 1$. Par conséquent on a besoin d'un protocole de transport fiable, afin que la perte d'un paquet au début du flux n'entraîne pas le transport de données inutiles. Par conséquent nous supposons un canal de transport sans perte, tel que TCP.

La présentation est composée d'objets codés en couches, associés à des intervalles de temps d'affichage. Le début de l'intervalle de temps d'affichage coïncide avec l'écheance de livraison de l'objet. Par exemple, une image statique de 2400 Kbits peut avoir un intervalle de temps d'affichage commençant à $t = 10$ secondes, et terminant à $t = 15$ secondes, ce qui résulte en une durée d'affichage de 5 secondes, et une échéance de livraison de $t = 10$ secondes. (Toutes les données temporelles sont relatives au début d'affichage de la présentation).

Nous résolvons le problème d'échelonnage d'une présentation multimédia comme un problème d'optimisation sous contraintes. Dans le cas d'une capacité mémoire client infinie, nous cherchons une politique de transmission sous la forme d'un vecteur à N dimensions, représentant le nombre d'unités de données à transmettre pour chaque objet, quand le nombre total d'objets est N. Entre toutes les politiques de transmission possibles, nous considérons celles qui rendent possible la livraison des données à temps. Selon la mesure de qualité d'une présentation utilisée, nous cherchons à trouver une politique de transmission qui maximise la qualité de la présentation affichée. Nous considérons deux méthodes différentes pour comparer la qualité globale de la présentation affichée : une mesure de qualité totale, et une mesure max-min affinée. Dans ce contexte, nous supposons l'existence d'une mesure de qualité pour un objet générique, qui détermine la qualité de la présentation globale. Nous démontrons la supériorité de la mesure max-min affinée pour résoudre les problèmes de ce domaine.

Table 1.1: Eléments de la Présentation

élément	intervalle d'affichage
image 1	[0, 15]
image 2	[15, 30]
image 3	[30, 50]
audio	[0, 50]

1.3.2 Politiques de Transmission

Pour illustrer les concepts de bases et les notations, nous ferons référence à la présentation de la Table 1.3.2, qui se compose de quatre éléments : trois images et un flux audio. Cette présentation a une durée d'affichage totale de 50 secondes. La première image a un intervalle d'affichage de $[0, 15]$, ce qui signifie qu'elle est affichée au client au temps $t = 0$ seconde, et continue à être affichée jusqu'au temps $t = 15$ secondes. La seconde image a un intervalle d'affichage de $[15, 30]$, et la troisième image de $[30, 50]$. La présentation est également accompagnée d'un flux audio, qui s'étend du temps $t = 0$ seconde jusqu'au temps $t = 50$ secondes.

Tous les éléments de la présentation ne peuvent pas être traités de la même manière. Les données des éléments discrets — notamment les images — doivent arriver au client avant le début de leurs intervalles d'affichage. Cependant, les éléments continus n'ont pas la même restriction. Par exemple, on peut commencer à jouer des données audio avant que toutes les données arrivent. Si l'on nécessitait que toutes les données audio arrivent au client avant le temps $t = 0$ seconde, cela ajouterait un délai superflu à l'affichage de la présentation. Pour résoudre ce problème, nous divisons le flux audio en de courts intervalles continus d'affichage, comme représenté dans la Table 1.3.2, et nous faisons référence aux unités qui comprennent les données de la présentation sous le terme d'"objet" plutôt qu'"élément", après l'application de cette transformation.

A présent, toutes les unités de données qui composent la présentation sont traitées comme des éléments discrets ayant des échéances de livraison égales au début de leurs intervalles d'affichage. La Table 1.3.2 illustre notre vision actuelle des données de la présentation.

Notre transformation finale est de classer les objets suivant leurs échéances de livraison et de leur assigner un index. Le résultat de l'application de cette transformation est illustré dans la Table 1.3.2. Nous désignons l'échéance de livraison de l'objet i par t_i.

Les objets qui composent la présentation peuvent être un ensemble de

Table 1.2: Objets de la présentation

objet	intervalle d'affichage
image 1	[0, 15]
image 2	[15, 30]
image 3	[30, 50]
audio 1	[0, 1]
audio 2	[1, 2]
⋮	⋮
audio 50	[49, 50]

Table 1.3: Echéances de livraison

objet	échéance de livraison
image 1	0
image 2	15
image 3	30
audio 1	0
audio 2	1
⋮	⋮
audio 50	49

Table 1.4: Numérotation des objets

objet	échéance de livraison	index des objets
image 1	0	1
audio 1	0	2
audio 2	1	3
⋮	⋮	⋮
audio 14	13	15
image 2	15	16
⋮	⋮	⋮
audio 50	49	53

plusieurs sortes de média, tels que des trames vidéo, des images fixes, des segments audio, etc. L'objet i est composé de L_i couches, et la j^{eme} couche de l'objet i contient x_{ij} bits. La Figure 1.15 donne un exemple d'illustration de la notation développée.

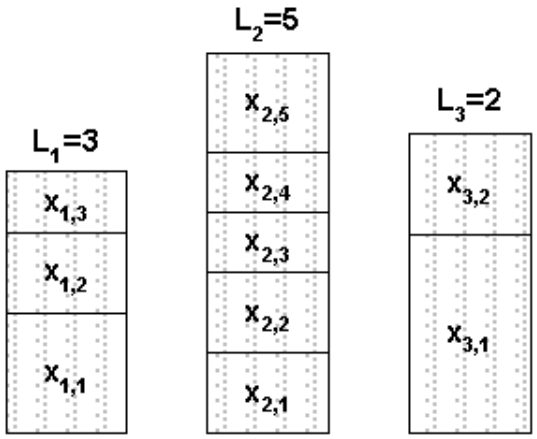

Figure 1.15: Données multimedia codées en couches

La base de temps de la présentation (Figure 1.16) commence au temps $t = -d$, où d est le nombre de secondes pendant lesquelles l'application récupère des données avant le début de l'affichage. Pour simplifier les notations, nous posons $t_0 = -d$.

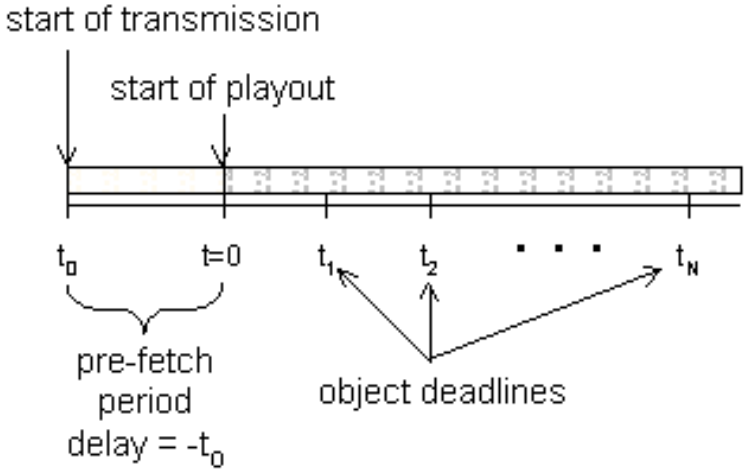

Figure 1.16: Base de temps de la présentation

Tout d'abord, nous supposons que la bande passante est connue, et que nous pouvons donc déterminer le nombre de bits qui peuvent être transmis dans chaque intervalle de la base de temps de la présentation. Soit b_i le nombre de bits qui peuvent être transmis dans l'intervalle i, c'est-à-dire l'intervalle $[t_{i-1}, t_i]$. Parce que la bande passante peut ne pas être suffisante pour transporter la meilleure résolution de tous les objets avant leurs échéances, l'application doit déterminer le nombre de couches qu'elle peut transmettre pour chaque objet, afin que tous les bits délivrés arrivent avant leurs échéances. Nous appelons politique de transmission, le nombre de couches à envoyer pour chaque objet, représentée par un vecteur à N dimensions $(j_1, \ldots, j_N)$. y_i représente le nombre de bits sélectionné par la politique de transmission pour le i^{eme} objet. La Figure 5.5 donne une illustration de ces variables.

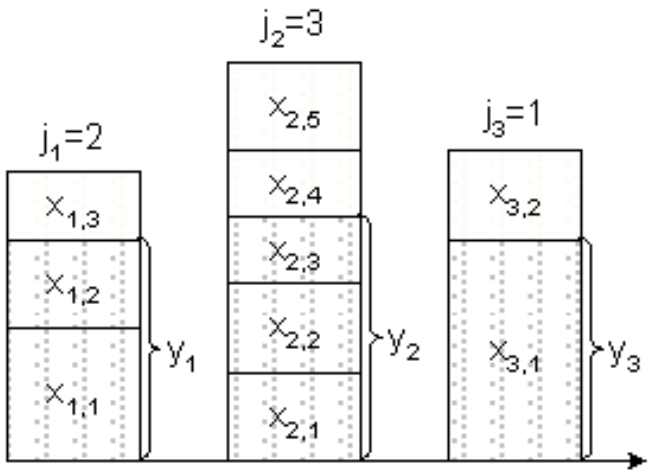

Figure 1.17: Politique de transmission

Nous disons qu'une politique de transmission $P = (j_1, \ldots, j_N)$ est *réalisable* si tous les bits arrivent au client avant leurs échéances.

Etant donné que chaque objet possède une échéance pour la livraison de tous ses bits, on peut supposer que l'application va toujours transmettre les bits des objets ayant des échéances plus courtes avant les objets ayant des échéances plus longues. Cette supposition fournit un moyen de déterminer si une politique de transmission particulière est réalisable. La Figure 5.6 montre comment cela fonctionne. Le nombre total de bits transmis pour l'objet 1 doit être inférieur ou égal au nombre total de bits qui peuvent être transmis durant le premier intervalle de temps, c'est-à-dire l'intervalle entre le début de la transmission t_0 et la première échéance t_1. Comme les bits de l'objet 2 sont transmis immédiatement après les bits de l'objet 1, la somme des bits

pour les objets 1 et 2 doit être inférieure ou égale au nombre total de bits qui peuvent être transmis avant l'échéance du deuxième objet t_2. La suite de cette logique devrait être claire à partir de la Figure 5.6, et le système d'inégalités représentant ces contraintes de transmission sont alors données en (1.1).

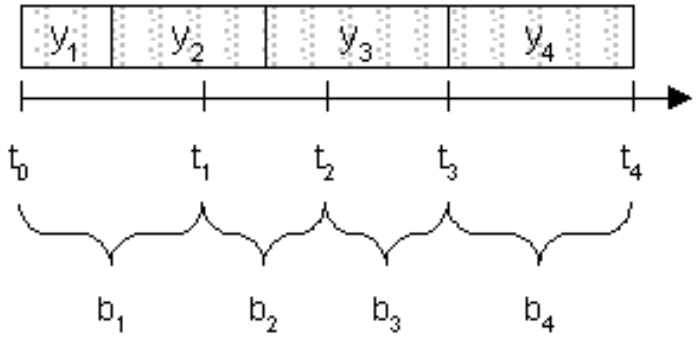

Figure 1.18: Contraintes de transmission

$$\begin{aligned} & y_1(j_1) \leq b_1 \\ & y_1(j_1) + y_2(j_2) \leq b_1 + b_2 \\ & \vdots \\ & y_1(j_1) + \ldots y_N(j_N) \leq b_1 + \ldots + b_N \end{aligned} \tag{1.1}$$

Afin de déterminer une politique de transmission optimale à partir de l'ensemble des politiques réalisables, on doit maximiser une mesure de la qualité globale de la présentation. Pour ce faire, on suppose que $Q(P)$ est une mesure de la qualité globale de la présentation résultant de la politique P. On dit qu'une politique de transmission est *optimale* si elle est réalisable et qu'il n'y a pas d'autre politique réalisable ayant une qualité globale plus élevée.

Pour spécifier $Q(P)$, on a besoin d'une mesure de la qualité d'un objet individuel. On suppose l'existence d'une mesure de qualité d'un objet générique $q_i(j)$ pour chaque objet, qui est une fonction du nombre de couches affichées. On suppose que q est non-décroissante, et prend ses valeurs dans $[0, 1]$, où $q_i(0) = 0$ et $q_i(L_i) = 1$. Notre mesure de qualité d'un objet prendra $L_i + 1$ valeurs différentes pour un objet i, étant donné qu'il y a autant de qualités d'affichage différentes.

1.3.3 Le Critère Max-min Affiné

On définit la *qualité totale* d'une présentation comme la somme des qualités des objets qui la composent. C'est-à-dire,

$$Q(P) = \sum_{i=0}^{N} q_i(j_i)$$

Une autre mesure possible de la qualité globale de la présentation est simplement de prendre la pire qualité d'un objet sur tous les objets qui composent la présentation. Une politique réalisable serait alors optimale si elle maximise la qualité minimale parmi tous les objets de la présentation. Nous appelons cela le *critère max-min*.

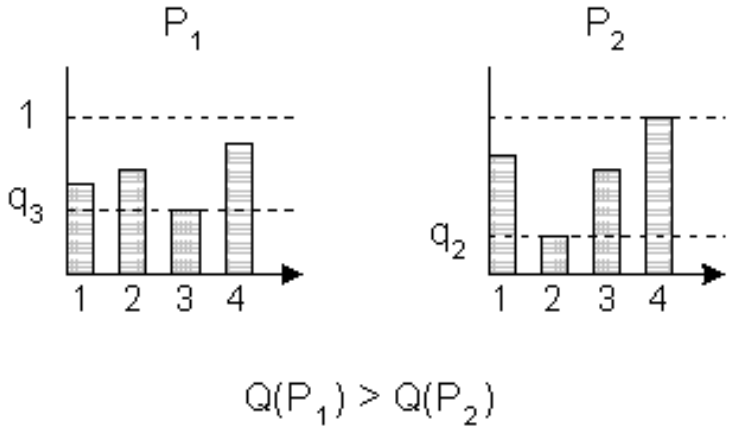

Figure 1.19: Critère Max-min

Bien que le critère max-min soit un choix naturel pour un critère d'optimalité, cela fournit typiquement des politiques qui n'allouent pas toute la bande passante disponible. Par conséquent, le fait d'envoyer des couches additionnelles pour quelques objets améliore la qualité globale de la présentation, bien que la qualité minimale reste fixe. En général, on dispose de moins de bande passante pour transmettre des objets ayant des échéances plus courtes, ainsi la qualité minimum que l'on peut atteindre sera dominée par la bande passante disponible pour les objets ayant des échéances courtes. Nous considérons un *critère max-min affiné* qui résout ces problèmes.

Avec le critère max-min, nous représentons la qualité globale de la présentation affichée par un vecteur de qualité des objets, dont les composants sont triés par ordre croissant de qualité.

Definition 1 (Vecteur de Qualité Max-min Affiné) *Soit $P = (j_1, \ldots, j_N)$ une politique de transmission. Si a_i est l'objet avec la i^{eme} moins bonne qualité sous P, alors nous appelons $Q(P) = (q_{a_1}(j_{a_1}), \ldots, q_{a_N}(j_{a_N}))$ le vecteur de qualité max-min affiné de P.*

Par exemple, supposons que nous avons une présentation composée de trois objets, une politique de transmission $P = (2, 4, 3)$, et des valeurs de qualité $q_1(2) = 16.3$, $q_2(4) = 30.4$, et $q_3(3) = 20.5$. Dans ce cas, nous avons $a_1 = 1, a_2 = 3, a_3 = 2$, et $Q(P) = (16.3, 20.5, 30.4)$. Il faut noter que s'il y a des qualités égales, alors il y a plus d'une affectation des objets à $a_1, \ldots, a_N$. Cependant, ces affectations diverses résulteront en un même vecteur de qualité, et ainsi le vecteur de qualité max-min affiné restera bien défini dans le cas de qualités égales.

Cette transformation des politiques de transmission en des vecteurs de qualité max-min affinés est un passage d'un domaine de vecteurs à N dimensions à valeurs entières, à un domaine de vecteurs à N dimensions à valeurs réelles dont les composants sont non-décroissants dans leur index : $n < m \Rightarrow q_n(j_n) \leq q_m(j_m)$. On définit un ordre total sur cet ensemble, afin de distinguer entre des politiques avec des qualités plus ou moins grandes.

Definition 2 (Ordonnancement des Vecteurs de Qualité) *Soit $Q(P) = (q_1, \ldots, q_N)$ et $Q(P') = (q'_1, \ldots, q'_N)$.*

Nous posons $Q(P) > Q(P')$ si et seulement si il existe un entier $k \in \{1, \ldots, N\}$ tel que $q_i = q'_i$ pour $i = 1, \ldots, k-1$, et $q_k > q'_k$.

La Figure 1.20 donne une illustration graphique pour montrer comment la Définition 2 est utilisée pour comparer deux politiques à 4 dimensions P_1 et P_2.

Definition 3 (Politique Max-min Affinée Optimale) *Nous disons qu'une politique de transmission réalisable est optimale s'il n'existe pas d'autre politique réalisable ayant une plus grande qualité.*

Nous présentons à présent un algorithme qui détermine la politique optimale en supposant que les valeurs de qualité $q_i(j)$ sont distinctes pour toutes les valeurs de i et j. L'algorithme (Fig. 1.21) initialise un vecteur de politique $(j_1, \ldots, j_N)$ à zéro. Ensuite, il entre dans une boucle dans laquelle il essaie d'ajouter une couche à l'objet ayant la qualité la plus faible. Si en ajoutant une couche à cet objet on obtient une politique réalisable, alors le compteur de couches pour cet objet est incrémenté. Au contraire, si l'on obtient une politique non réalisable, alors les couches de cet objet sont fixées, et ne sont plus considérées (en les enlevant de S) pour une éventuelle amélioration de qualité. Ainsi, l'algorithme a la forme d'un algorithme glouton classique.

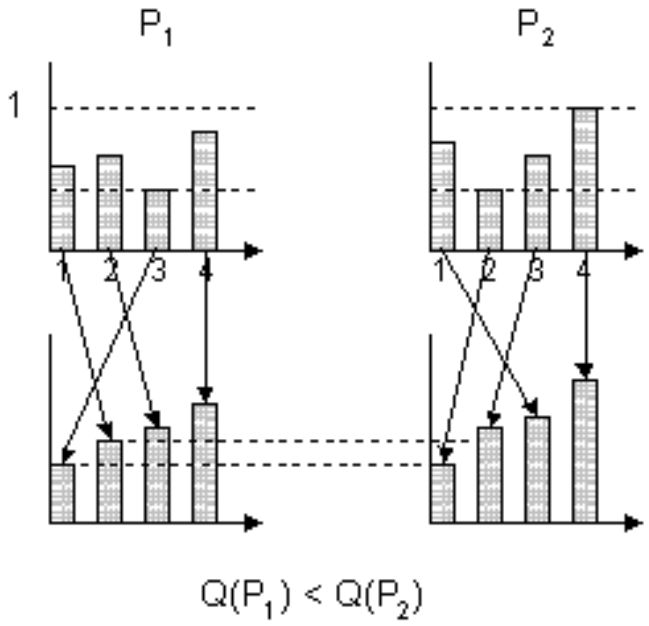

Figure 1.20: Critère max-min affiné

pour $i \leftarrow 1, \ldots, N$
    $j_i = 0$
$S = \{1, \ldots, N\}$
tant que S n'est pas vide
    trouver $k \in S$ t.q. $q_k(j_k) \leq q_i(j_i)$ pour tout i dans S
    si $(j_1, \ldots, j_k + 1, \ldots, j_N)$ est réalisable alors
        $j_k \leftarrow j_k + 1$
    sinon
        enlever k de S

Figure 1.21: Algorithme max-min affiné

Nous avons montré que l'algorithme max-min affiné converge vers une politique optimale pour des mesures de qualité croissantes prenant des valeurs distinctes dans l'ensemble des objets de la présentation. Pour les mesures de qualité qui prennent leurs valeurs dans un intervalle relativement petit, tel que $q_i(j) = j$, l'algorithme max-min affiné peut ne pas converger vers une politique optimale, parce qu'il ne résout pas vraiment les cas de qualités égales. Cependant, nous avons trouvé que l'heuristique suivante donne de bonnes performances dans ce cas : en présence de valeurs de qualité égales, choisir l'objet dont la prochaine couche a le plus petit nombre de bits. Intuitivement, cette heuristique est appropriée, car nous augmentons la qualité

d'une couche avec la moindre consommation de bande passante.

1.3.4 Résultats Expérimentaux

Nous avons réalisé une présentation composée de 2 images JPEG en noir et blanc et de 8 images en couleurs, afin de comparer le critère max-min affiné avec deux mesures de qualité d'objet différentes. Nous avons codé les images en utilisant l'" Independent JPEG Group's library" [JPEG], qui, par défaut, code les images couleur en 10 couches et les images en noir et blanc en 6 couches. Les deux mesures de qualité d'objet étaient une mesure de couches, où la qualité est égale au nombre de couches affichées sur le nombre de couches totales de l'objet, et une mesure de bits, où la qualité est égale au nombre de bits affichés sur le nombre de bits total de toutes les couches.

A partir de ces expériences, nous avons trouvé que le critère max-min affiné est supérieur aux critères max-min et de qualité totale, dans le sens d'une variance de qualité d'objet moins élevée, et tout en utilisant presque toute la bande passante disponible pour améliorer la qualité globale de la présentation.

1.3.5 Adaptation aux Changements de Bande Passante

Dans cette section, nous considérons le problème du streaming d'une présentation multimédia codée en couches d'un serveur à un client arbitraire sur un réseau de communication avec aucune garantie de QoS, tel que le réseau IP global. Des études à propos de la bande passante TCP sur Internet [PAX97a] ont démontré la non stationarité du débit TCP. Pour cette raison, l'application doit observer continuellement le taux de perte des paquets et les caractéristiques de délai, et utiliser cela pour ajuster la bande passante attendue.

Nous supposons qu'une présentation qui est comprise seulement de la première couche de chaque objet donne une qualité minimum acceptable, et nous utilisons le terme de *couche de base* pour faire référence à la première couche, et *couche d'amélioration* pour dénommer les couches plus hautes, qui contribuent à la qualité mais ne sont pas requises pour obtenir le niveau minimum de qualité.

Etant donné que nous transmettons les données sur un réseau sans garantie de QoS, l'application ne connaît pas le débit auquel elle peut transmettre les données de la source jusqu'à la destination. Pour cette raison, l'application commence par transmettre les données de la couche de base seulement, qu'elle stocke dans un tampon de "prefetching" au client, jusqu'à ce qu'elle détermine qu'elle peut débuter l'affichage de la présentation sans craindre que la

présentation s'arrête.

Pendant que l'application transmet les couches de bases, elle exécute deux autres tâches en parallèle. Premièrement, elle met à jour un historique de la transmission des paquets, qu'elle utilise pour prédire la bande passante future. Deuxièmement, elle détermine si elle peut commencer la présentation.

On suppose que l'application possède une méthode pour estimer une limite inférieure de la bande passante attendue, à partir des paquets précédemment transmis. Soit $B(a,b)$ une limite inférieure du nombre de bits que l'application espère transmettre du serveur au client dans l'intervalle de temps $[a,b]$.

i	index des objets de la présentation
N	nombre d'objets dans la présentation
L_i	nombre total de couches pour l'objet i
j_i	couches de l'objet i à transmettre (et à afficher)
$B(a,b)$	limite inférieure estimée de la bande passante sur $[a,b]$
t_i	échéance d'arrivée des couches de l'objet i

Figure 1.22: Notations

Imaginons que l'application a transmis les couches de base des objets 1 à $q-1$ au client, et considère à présent s'il faut commencer ou non l'affichage. Soit $x_{i,j}$ le nombre de bits de la j^{eme} couche de l'objet i. Si l'application commence l'affichage maintenant, toutes les couches de base qui n'ont pas été envoyées arriveront à temps si le nombre de bits cumulés requis à chaque échéance est inférieur ou égal au nombre de bits cumulés qui peuvent être transmis. Ainsi, l'application commence l'affichage si les inégalités suivantes sont satisfaites :

$$\begin{aligned} & x_{q,1} \leq B(0,t_q) \\ & x_{q,1} + x_{q+1,1} \leq B(0,t_{q+1}) \\ & \vdots \\ & x_{q,1} + \ldots + x_{N,1} \leq B(0,t_N) \end{aligned}$$

Si ce système d'inégalités n'est pas satisfait, l'application ne commence pas l'affichage, mais continue à récupérer des couches de base supplémentaires.

Après que l'application donne la commande de commencer le play-back, elle passe à la deuxième phase, dans laquelle elle doit décider de la couche à envoyer. Pour prendre cette décision, l'application continue à enregistrer des statistiques sur la transmission et à affiner son estimation d'une limite

inférieure de la bande passante future. Grâce à cette estimation, elle détermine une séquence de couches P (la politique de transmission) qui peut être délivrée à temps (réalisable) et qui maximise une mesure objective de qualité de la présentation $Q(P)$. La première couche de la séquence de cette politique est choisie pour être transmise.

Si l'on suppose que l'affichage des premiers r objets a déjà commencé, alors l'application n'a besoin de s'occuper que d'ordonnancer la livraison des couches des objets $r+1$ à N. Pour simplifier les notations, nous ré-indexerons ces objets de 1 à M.

Une politique de transmission spécifie le nombre de couches à envoyer pour chaque objet, que nous représentons par un vecteur à M dimensions P, dont le i^{eme} composant j_i représente le nombre de couches de l'objet i à envoyer au client. Nous dirons qu'une politique est *réalisable* si tous les bits qu'elle envoie arrivent au client avant leurs échéances.

Soit $b_i = B(t_{i-1}, t_i)$, l'estimation, par l'application, du nombre de bits qui peuvent être transmis entre les échéances t_{i-1} et t_i. A chaque échéance, le nombre de bits cumulés requis doit être inférieur ou égal au nombre de bits cumulé qui seront transmis. Ainsi, si $y_i(j)$ est égal au nombre de bits de l'objet i qui n'ont pas encore été transmis dans les couches 1 à j, alors la politique P est réalisable si le système des M inégalités suivantes est valide :

$$\begin{aligned} & y_1(j_1) \leq b_1 \\ & y_1(j_1) + y_2(j_2) \leq b_1 + b_2 \\ & \vdots \\ & y_1(j_1) + \ldots y_M(j_M) \leq b_1 + \ldots + b_M \end{aligned}$$

Maintenant que nous avons défini l'ensemble des politiques réalisables, nous devons déterminer celles qui sont optimales, c'est-à-dire celles qui donnent une présentation ayant la meilleure qualité. L'application applique alors le critère max-min affiné pour trouver une politique optimale. Il envoie alors la couche restant à envoyer suivant la politique optimale.

1.3.6 Espace Mémoire Limité au Client

Avec un tampon mémoire infini au client, l'application peut transmettre les données au débit maximum disponible, et les bits qui sont sélectionnés pour être envoyés au client sont seulement contraints par la bande passante disponible. Cependant, avec un tampon limité au client, l'application doit contrôler le débit auquel elle transmet les données pour éviter de faire déborder le tampon du client. Dans ce but, nous introduisons une nouvelle variable z_i qui représente le nombre de bits que le serveur transmet pendant le i_{eme}

intervalle (voir la figure 1.23). Une politique de transmission P est à présent représentée par une paire de vecteurs à N dimensions $(\vec{j}, \vec{z})$, où j_i est le nombre de couches à envoyer pour le i_{eme} objet, et z_i le nombre de bits à transmettre sur le i_{eme} intervalle. Les valeurs j_i sont entières, tandis que les valeurs z_i sont réelles.

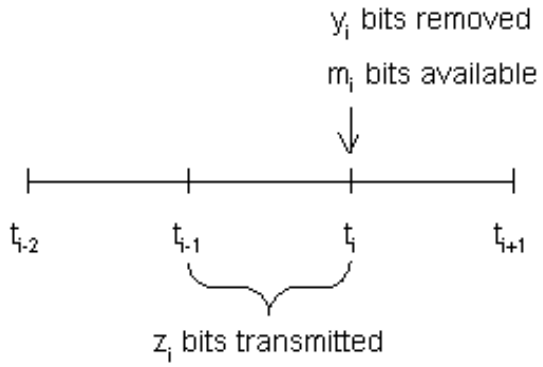

Figure 1.23: Le i^{eme} intervalle

Comme précédemment, nous supposons que l'application envoie les données de la présentation de manière séquentielle, c'est-à-dire qu'elle envoie les bits de l'objet 1, puis de l'objet 2, et ainsi de suite. Cette supposition est valide, parce que les objets sont ordonnés par leurs échéances. Ainsi, la variable z_i remplace b_i comme ressource de contrainte sur les sommes de bits, donnant le système de contraintes suivant : (à noter que y_i est une fonction de j_i)

$$\begin{aligned} & y_1 \leq z_1 \\ & y_1 + y_2 \leq z_1 + z_2 \\ & \vdots \\ & y_1 + \cdots + y_N \leq z_1 + \cdots + z_N \end{aligned}$$

Le nombre de bits que l'application peut transmettre sur le i_{eme} intervalle doit être inférieur ou égal au nombre maximum de bits que le canal de transmission permet sur cet intervalle, ce que nous avons déjà défini comme b_i.

Ainsi, nous avons les contraintes de taux de transmission suivantes :

$$z_i \leq b_i \text{ for } i = 1, \ldots, N.$$

Nous devons aussi limiter le taux de transmission pour éviter le débordement du tampon au client. Nous supposons que les bits du i_{eme} objet sont enlevés du tampon client au début de l'intervalle d'affichage de l'objet, t_i. Pour ces raisons, la quantité de stockage libre dans le tampon du client augmente par y_i au temps t_i. (Voir la figure 1.23.)

Soit m_i la quantité de mémoire libre dans le tampon au temps t_i, après que le i^{eme} objet ait été enlevé. Soit M la capacité du tampon, avec $m_0 = M$ pour faciliter les notations.

L'application ne fera pas déborder le tampon au client si le nombre de bits qu'elle transmet pendant le i^{eme} intervalle est inférieur ou égal à la quantité de mémoire libre dans le tampon après que l'objet $i-1$ ait été enlevé, égale à m_{i-1}. Ainsi, nous avons les contraintes de mémoire suivantes :

$$z_i \leq m_{i-1} \text{ for } i = 1, \ldots, N,$$

L'application a besoin de connaître à la fois le nombre de couches j_i à transmettre et les taux de transmission z_i. Ainsi, le nouvel espace de recherche est $(\vec{j}, \vec{z})$, et l'ensemble des solutions réalisables dans cet espace est défini par les trois systèmes d'inégalités précédents, et par le fait que les j_i et z_i soient non-négatifs.

La mémoire disponible à la fin du i^{eme} intervalle, m_i (au temps t_i), est égale à la mémoire qui était disponible au début du i^{eme} intervalle, m_{i-1}, moins le nombre de bits transmis sur cet intervalle, z_i, plus le nombre de bits enlevés à la mémoire du client au temps t_i. Ainsi, pour $i = 1, \ldots, N$, on peut écrire :

$$m_i = m_{i-1} - z_i + y_i.$$

Par déduction, on a :

$$m_i = M + (y_1 - z_1) + \cdots + (y_i - z_i),$$

pour $i = 1, \ldots, N$, et alors les contraintes de mémoire peuvent être réécrites comme :

$$z_i \leq M + (y_1 - z_1) + \cdots + (y_{i-1} - z_{i-1}).$$

Ainsi, l'état de réalisabilité peut être exprimé complètement en fonction des variables dont dépend notre politique de transmission, $(\vec{y}, \vec{z})$, sans reposer sur les variables intermédiaires m_i.

Nous présentons deux algorithmes qui calculent des politiques de transmission optimales. On peut classifier le premier algorithme comme une méthode exhaustive, et le second algorithme comme une méthode gloutonne. Les

politiques optimales ne sont généralement pas uniques, parce que la transmission des bits peut quelques fois être retardée. Cependant, nos algorithmes exhaustif et glouton convergent vers la même solution, qui est d'envoyer les bits le plus tôt possible.

L'algorithme exhaustif est représenté sur la figure 1.24. Il donne une politique de transmission réalisable en un temps $O(L_1 L_2 \cdots L_N)$. Cet algorithme n'est pas absolument exhaustif dans le sens où il essaie toutes les valeurs entières de $(j_1, \ldots, j_N)$ mais n'essaie pas toutes les valeurs réelles de $(z_1, \ldots, z_N)$. Avec chaque affectation de $\vec{j}$, on affecte les plus grandes valeurs possibles de $z_1, \ldots, z_N$ permises par les contraintes de mémoire et de bande passante sur $\vec{z}$. Cette affectation est accomplie en exécutant l'algorithme suivant pour $k = 1, \ldots, N$:

$$z_k \leftarrow min(b_k, m_{k-1})$$
$$m_k \leftarrow m_{k-1} - z_k + y_k$$

Après l'affectation de z_k et de m_k lors de la k^{eme} itération de la boucle vue plus haut, on vérifie les inégalités des sommes de bits jusqu'à la k^{eme} échéance. Si cette vérification échoue, on sait que toutes les politiques $P = (\vec{j}, \vec{z})$ avec $\vec{j} = (j_1, \ldots, j_N)$ ne sont pas réalisables. Dans ce cas, l'algorithme passe à la prochaine affectation de $\vec{j}$.

si $\sum_{i=0}^{k} y_i > \sum_{i=0}^{k} z_i$
    essayer la prochaine affectation de $(j_1, \ldots, j_N)$

Si les inégalités des sommes de bits sont satisfaites, alors on sait que $(j_1, \ldots, j_N, z_1, \ldots, z_N)$ est réalisable. On vérifie alors l'amélioration de qualité par rapport à l'affectation précédemment enregistrée :

si $Q(j_1, \ldots, j_N) > Q(\hat{j_1}, \ldots, \hat{j_N})$ alors
    $(\hat{j_1}, \ldots, \hat{j_N}) \leftarrow (j_1, \ldots, j_N)$

L'algorithme complet est donné dans la figure 1.24. On peut noter que $(\hat{j_1}, \ldots, \hat{j_N})$ représente la plus grande qualité réalisable affectée à $\vec{j}$ sur l'ensemble des affectations testées. Lorsque l'algorithme se termine, $(\hat{j_1}, \ldots, \hat{j_N})$ contiendra une affectation optimale à $\vec{j}$. Bien que cela n'est pas visible, lorsque l'algorithme se termine, l'affectation à $\vec{z}$ est déterminée par les variables intermédiaires $m_1, \ldots, m_N$.

// boucle principale
$(\hat{j_1}, \ldots, \hat{j_N}) \leftarrow (0, \ldots, 0)$
pour toutes les $L_1 L_2 \cdots L_N$ affectations possibles à $(j_1, \ldots, j_N)$
    $m_0 \leftarrow M$
    pour $k = 1, \ldots, N$
        $z_k \leftarrow min(b_k, m_{k-1})$
        $m_k \leftarrow m_{k-1} - z_k + y_k$
        si $\sum_{i=0}^{k} y_i > \sum_{i=0}^{k} z_i$
            essayer la prochaine affectation à $(j_1, \ldots, j_N)$
    si $Q(j_1, \ldots, j_N) > Q(\hat{j_1}, \ldots, \hat{j_N})$ alors
        $(\hat{j_1}, \ldots, \hat{j_N}) \leftarrow (j_1, \ldots, j_N)$

Figure 1.24: Algorithme exhaustif pour trouver des politiques de transmission optimales contraintes par l'espace mémoire pour des présentations multimédia codées en couches.

L'algorithme exhaustif fonctionne pour n'importe quelle mesure de qualité de présentation $Q(P)$. Cependant, si nous nous limitons à la mesure max-min affinée, alors nous pouvons remplacer la recherche exhaustive parmi $\{0, \ldots, L_1\} \times \cdots \{0, \ldots, L_N\}$ valeurs par une recherche gloutonne. Dans ce cas, on commence par affecter une valeur nulle à $\vec{j}$, et on incrémente la composante de $\vec{j}$ ayant la qualité minimale dans l'ensemble S. Nous supprimons les éléments de S qui ne peuvent plus être augmentés dans l'état réalisable, ou qui ont atteint leur couche limite L_i. On continue aussi l'affectation simple à $\vec{z}$ grâce aux variables mémoire intermédiaires $\vec{m}$ qui satisfont les contraintes de bande passante et de mémoire placées sur $\vec{z}$. La figure 1.25 montre l'algorithme complet.

L'algorithme glouton est bien plus efficace que la version exhaustive, avec un temps de terminaison borné par $O(LN^2)$. La table 1.3.6 montre la différence de temps d'exécution de ces algorithmes en Java, à partir de plusieurs exemples de présentations, et avec des bandes passantes et des capacités différentes. Dans ces présentations, la version "b" a une plus grande bande passante et une plus grande mémoire que la version "a" correspondante.

$S = \{1, \ldots, N\}$
$m_0 \leftarrow M$
$z_1 \leftarrow min(b_1, m_0)$
tant que S n'est pas vide
    trouver $k \in S$ ayant la qualité la plus faible
    $j_k \leftarrow j_k + 1$
    $y_k \leftarrow y_k + x_{k,j_k}$
    // affecter les valeurs à $\vec{z}$ qui respectent les contraintes de bande passante et de mémoire
    pour $i = 1$ à $N - 1$
        $m_i \leftarrow m_{i-1} - z_i + y_i$
        $z_{i+1} \leftarrow min(b_{i+1}, m_i)$
    // vérifier l'état réalisable
    si $(j_1, \ldots, j_N, z_1, \ldots, z_N)$ n'est pas réalisable alors
        $y_k \leftarrow y_k - x_{k,j_k}$
        $j_k \leftarrow j_k - 1$
        enlever k de S
    sinon si $j_k = L_k$ alors enlever k de S

Figure 1.25: Algorithme glouton pour trouver des politiques de transmission optimales contraintes par l'espace mémoire pour des présentations multimédia codées en couches.

1.4 Streaming Optimal de Présentations Continuement Scalables

1.4.1 Introduction

Les formats JPEG progressif, GIF et MPEG-2 sont des types d'encodage qui possèdent la propriété de scalabilité à granularité épaisse (CGS comme "Coarse Grained Scalability") : leurs données sont codées en couches, et chaque couche successive contribue à augmenter la résolution du média affiché. JPEG-2000, MPEG-4 et Ogg Vorbis sont des formats de codage récents, qui possèdent la propriété de scalabilité à granularité fine (FGS comme "Fine Grained Scalability"), ce qui signifie que la taille des couches de données peut être rendue soit très petite, soit arbitrairement petite. Par souci de simplicité, on suppose la propriété de "scalabilité continue". Avec un objet continuement scalable, lorsque r pourcent des bits de cet objet sont délivrés au client, alors r pourcent des bits sont utilisés pour son affichage.

Nous supposons que la séquence de transformations appliquée aux données de la présentation est la même que dans le cas de présentations dis-

Table 1.5: Temps d'exécution en millisecondes pour les algorithmes exhaustif et glouton

présentation	N	exhaustif	glouton
P4a	4	60	20
P4b	4	110	20
P8a	8	2123	20
P8b	8	5768	20
P10a	10	67277	20
P10b	10	188431	20
P16a	16	∞	30
P16b	16	∞	40

crètement scalables. Ainsi, nous avons une séquence de N objets discrets continuement scalables ayant des échéances de livraison de $t_1 \leq \ldots t_N$.

Nous résolvons le problème d'échelonnage optimal des données de la présentation comme un problème d'optimisation sous contraintes selon le critère max-min affiné. Dans cette section, nous considérons le cas d'une bande passante fixée. Dans les environnements avec des qualités de service irrégulières, le mécanisme adaptatif décrit précédemment peut être utilisé d'une manière identique dans le cas d'un média continuement scalable.

1.4.2 Faisabilité

Nous voudrions délivrer (et afficher) la pleine résolution du i^{eme} objet, ce qui nécessite x_i bits. Cependant, pour satisfaire aux limitations en bande passante et en mémoire client, et suivant la contrainte d'un délai de play-back initial, l'application peut n'être capable de transmettre que r_i pourcent du i^{eme} objet. Nous appelons le vecteur $R = (r_1, \ldots, r_N)$, le vecteur de résolution. La Figure 1.26 illustre le modèle.

On dit qu'un vecteur de résolution R est *réalisable*, si la transmission suivant $(r_1x_1, \ldots, r_Nx_N)$ permet à chaque objet d'arriver à sa destination avant le début de son intervalle d'affichage. Ainsi, nous considérons $t_1, \ldots, t_N$ comme les échéances de livraison des objets 1 à N.

Soit B la bande passante disponible à l'application, et d le délai initial entre le début de la transmission et le début d'affichage de la présentation. Nous désignons par t_0 le début de la transmission, donc $t_0 = -d$. Si l'échéance du premier objet est au temps $t_1 = 0$, on affectera à d une valeur positive non nulle afin que telle ou telle version du premier objet soit affichée. (Voir la Figure 1.27.)

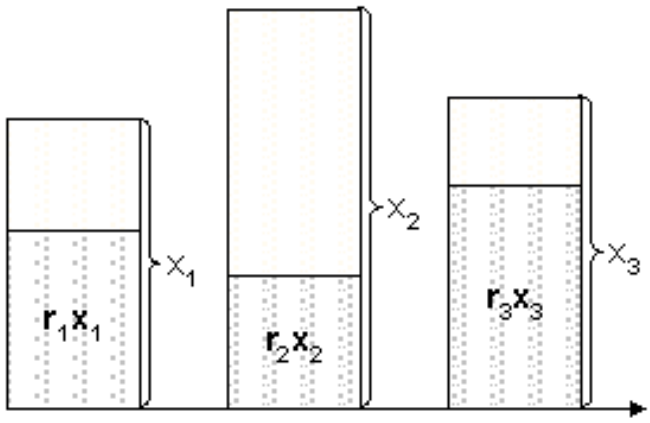

Figure 1.26: Objets multimédia continuement scalables

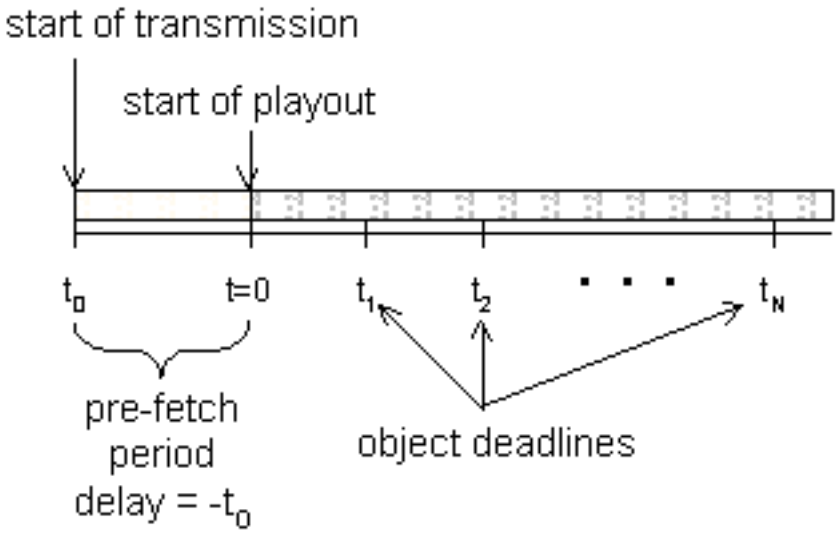

Figure 1.27: Base de temps de la présentation

On définit b_k comme le nombre de bits que l'application peut transférer de la source à la destination sur l'intervalle de temps $[t_{k-1}, t_k]$. Ainsi, $b_1 + \cdots + b_k$ représente le nombre de bits cumulés que l'application peut transférer à partir du début de la transmission jusqu'au début de l'intervalle d'affichage du k^{eme} objet. Selon la politique de transmission R, toutes les échéances seront satisfaites si le nombre de bits cumulés nécessaire à chaque échéance est inférieur ou égal au nombre de bits cumulés qui peuvent être délivrés avant cette échéance. Ainsi, le vecteur de résolution $(r_1, \ldots, r_N)$ est réalisable si

les inégalités de la Figure 1.28 sont satisfaites. La faisabilité signifie que la présentation est affichée de manière continue au client.

$$
\begin{aligned}
& r_1 x_1 \leq b_1 \\
& r_1 x_1 + r_2 x_2 \leq b_1 + b_2 \\
& \vdots \\
& r_1 x_1 + \ldots + r_N x_N \leq b_1 + \cdots + b_N
\end{aligned}
$$

Figure 1.28: Contraintes en bande passante

Ces contraintes spécifient l'ensemble des vecteurs de résolution réalisables, ou de manière équivalente les politiques de transmission. Notre but est de déterminer une méthode pour sélectionner un élément de cet ensemble, qui maximise la qualité globale de la présentation.

1.4.3 Algorithme de redistribution

On dit qu'une politique de transmission P est optimale dans le sens max-min affiné, si elle est réalisable et si sa qualité max-min affinée $Q(P)$ est supérieure ou égale à la qualité max-min affinée de toutes les autres politiques réalisables. Nous présentons un algorithme, appelé l'algorithme de redistribution, qui converge vers une politique optimale. La preuve de cette optimalité est basée sur un lemme qui dit que les composantes d'un vecteur de résolution optimal sont non-décroissantes.

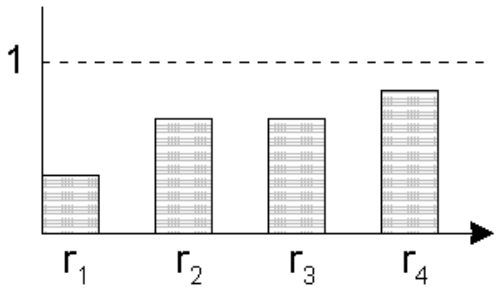

Figure 1.29: Lemme de la non-décroissance

Ce lemme nous fournit un aperçu de la manière dont on pourrait déterminer efficacement une politique optimale. Nous passons en revue tous les objets de 1 à N. Dans l'étape numéro k, nous initialisons la résolution de

l'objet k à celle qui peut être obtenue en utilisant la bande passante qui suit l'échéance de l'objet $k-1$, c'est-à-dire la bande passante sur l'intervalle k. Si cette résolution est inférieure à la résolution de n'importe quel objet précédant, alors nous ré-allouons la bande passante d'un ou plus des intervalles précédents afin que la résolution de l'objet k soit égale à la résolution de l'objet précédent $k-1$. Cet algorithme, illustré dans la Figure 1.30, converge vers un vecteur de résolution max-min affiné optimal, qui respecte les contraintes en bande passante.

pour $k = 1$ á N
    $b \leftarrow b_k$
    $x \leftarrow x_k$
    $r \leftarrow b \div x$
    $j \leftarrow k$
    tant que $(j \geq 2)$ et $(r_{j-1} > r)$
        $j \leftarrow j - 1$
        $b \leftarrow b + b_j$
        $x \leftarrow x + x_j$
        $r \leftarrow b \div x$
    pour $i = j$ à k
        $r_i \leftarrow r$

Figure 1.30: L'algorithme de redistribution

La figure 1.31 illustre, de manière graphique, le fonctionnement de cet algorithme à travers plusieurs étapes d'exécution, commençant à la première itération de la boucle extérieure. Dans la première itération, l'algorithme alloue toute la bande passante du premier intervalle pour la livraison du premier objet. Il en résulte une valeur de r_1, comme indiqué sur la figure. L'itération suivante sélectionne le second objet, et alloue toute la bande passante du second intervalle à la livraison des bits de l'objet 2. Il en résulte une résolution r_2 plus grande que le premier objet, et l'algorithme continue dans la boucle extérieure. Il alloue toute la bande passante du troisième intervalle au troisième objet, donnant une résolution meilleure que les objets précédents, et l'algorithme passe au quatrième objet. Mais quand l'algorithme utilise la bande passante du quatrième intervalle pour délivrer l'objet 4, cela donne une résolution inférieure à l'objet précédent. L'algorithme rentre alors dans la boucle intérieure dans laquelle il revient dans le temps pour ré-allouer la bande passante afin d'améliorer la résolution du quatrième objet, en diminuant la résolution des objets précédents. Dans la première étape de la boucle

intérieure, l'algorithme ajuste l'allocation de la bande passante entre les objets 3 et 4, donnant des résolutions identiques entre ces deux objets. Cependant, la résolution ainsi atteinte est toujours inférieure à un objet précédent, l'objet 2. Alors, l'algorithme reste dans la boucle intérieure, et diminue le nombre de bits délivrés du second objet afin d'augmenter les bits envoyés pour les objets 3 et 4. Cela résulte en des résolutions identiques pour les objets 2, 3 et 4. A cet instant, les résolutions forment à présent une séquence non-décroissante dans le temps, et l'algorithme sort de la boucle intérieure pour continuer dans la boucle extérieure, dans laquelle il considère l'objet 5, et ainsi de suite.

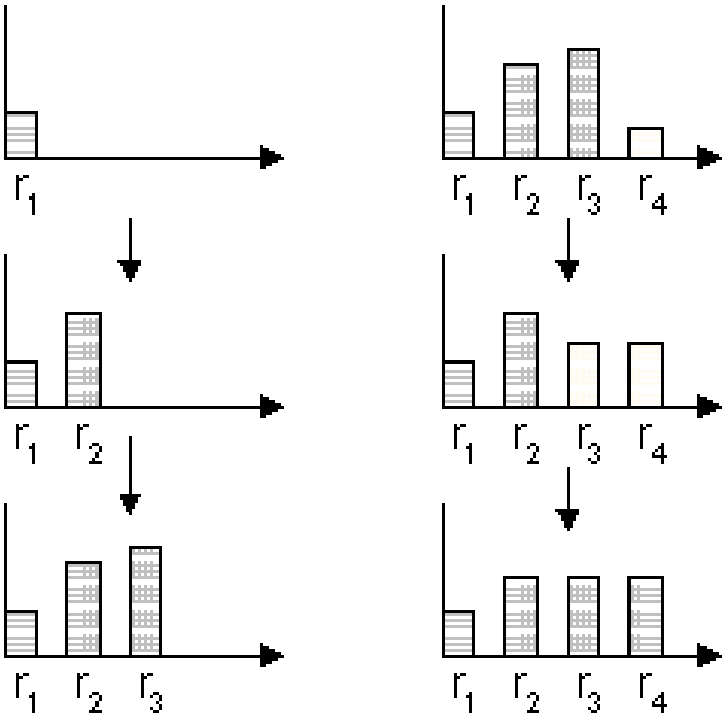

Figure 1.31: Algorithme de redistribution

A l'étape k de la boucle extérieure, l'algorithme fixe la résolution du k^{eme} objet afin que l'égalité soit vérifiée dans la k^{eme} contrainte, en utilisant les valeurs de $r_1, \ldots, r_{k-1}$ fixées dans les itérations précédentes. L'algorithme vérifie alors la résolution de l'objet précédent $k-1$; si sa résolution r_{k-1} est plus grande que r_k, l'algorithme rentre dans la boucle intérieure dans laquelle il redistribue la bande passante afin que r_k soit augmentée et r_{k-1} diminuée d'autant. Il continue ce processus de redistribution jusqu'à ce que la résolution précédente soit inférieure ou égale à r_k. Ainsi, par sa con-

ception, l'algorithme converge vers un vecteur de résolution dans lequel les composantes sont non-décroissantes dans leur index.

A des étapes intermédiaires de son exécution, l'algorithme peut fixer des composantes du vecteur de résolution à des valeurs supérieures à 1. Dans ce cas, cela représente l'excès de capacité du canal de transmission pour le transport des bits des objets successifs. Lorsque l'algorithme de la Figure 1.30 se termine, l'application doit mettre à 1 toutes les résolutions supérieures à 1.

1.4.4 Contraintes Mémoire au Client

Concernant le problème de mémoire limitée au client, nous développons un système de contraintes qui exprime les limitations à la fois de mémoire au client et de capacité de transmission. Nous montrons comment une politique de transmission optimale peut être atteinte en utilisant la programmation linéaire.

L'application ne peut plus transmettre les données au taux maximum disponible, parce que le tampon mémoire au client peutêtre totalement rempli si les données ne sont pas enlevées du tampon assez rapidement. Soit z_i le taux auquel le serveur transmet les données dans le i^{eme} intervalle (Figure 1.32).

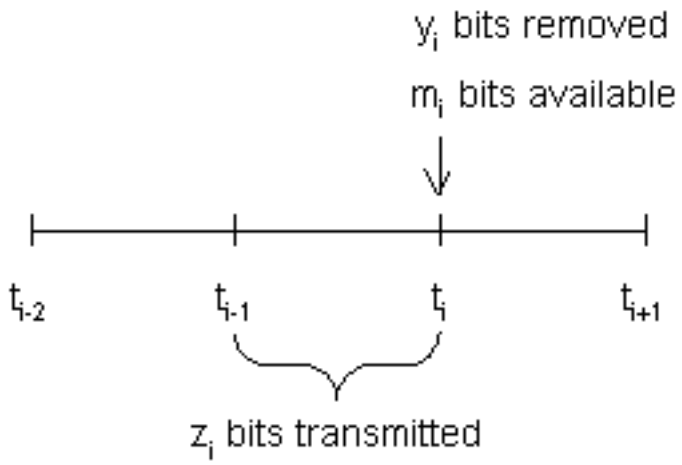

Figure 1.32: Le i^{eme} intervalle

Comme précédemment, nous supposons que l'application envoie les données de la présentation de manière séquentielle, c'est-à-dire qu'elle envoie les bits de l'objet 1, puis de l'objet 2, et ainsi de suite. Cette supposition n'affecte pas la généralité de nos propos, parce que les objets sont ordonnés par leurs échéances, et tous les bits d'un objet doivent arriver avant l'échéance de l'objet. Ainsi, les z_i remplacent les b_i en tant que contraintes sur les facteurs de résolution :

$$\begin{aligned} & r_1 x_1 \leq z_1 \\ & r_1 x_1 + r_2 x_2 \leq z_1 + z_2 \\ & \vdots \\ & r_1 x_1 + \cdots + r_N x_N \leq z_1 + \cdots + z_N \end{aligned}$$

Figure 1.33: Contraintes sur les facteurs de résolution

Le taux de transmission z_i est contraint par le taux maximum de transmission sur le i^{eme} intervalle, et par l'espace libre dans le tampon mémoire du client au début de l'intervalle. Ainsi, les contraintes de taux de transmission sont

$$z_i \leq b_i \text{ pour } i = 1, \ldots, N,$$

et les contraintes de mémoire sont

$$z_i \leq m_{i-1} \text{ pour } i = 1, \ldots, N,$$

où m_i est le nombre de bits disponibles (libres) dans le tampon du client juste après le départ de l'objet i au temps $t = t_i$. (Voir la Figure. 1.32.) Par souci de simplicité, on pose m_0 la capacité du tampon client M.

L'application doit connaître à la fois les facteurs de résolution r_i, et les taux de transmission z_i. Ainsi notre nouvel espace de recherche est $(\vec{r}, \vec{z})$, et l'ensemble des solutions réalisables est défini par les trois systèmes d'inégalité donnés plus haut, et par le fait que les r_i et z_i sont toutes non négatives. Notre prochaine étape est d'éliminer les variables m_i des contraintes.

A chaque échéance t_i, l'application retire $r_i x_i$ bits du tampon pour être affichés. Cela signifie que l'espace libre dans le tampon juste après le retrait a augmenté de $r_i x_i$. Mais l'espace libre dans le tampon juste avant le retrait est l'espace libéré par le retrait de l'objet $i-1$ au temps t_{i-1}, moins le nombre de bits transmis pendant le i^{eme} intervalle z_i. Ainsi, pour $i = 1, \ldots, N$, on peut écrire :

$$m_i = m_{i-1} - z_i + r_i x_i.$$

Par déduction, on a :

$$m_i = M + (r_1 x_1 - z_1) + \cdots + (r_i x_i - z_i).$$

Ainsi, les contraintes de mémoire peuvent être réécrites comme suit :

$$z_i \leq M + (r_1 x_1 - z_1) + \cdots + (r_i x_i - z_i).$$

Il est facile de voir que les trois ensembles de contraintes ont la forme :

$$\sum_{i=1}^{N} c_{ij} z_i \leq M_j,$$

pour $j = 1, \ldots, J$, et $z_j \geq 0$.

On peut déterminer le facteur de résolution max-min relatif à ces contraintes en introduisant une variable temporaire y, et en résolvant le problème de programmation linéaire suivant :

$$\begin{array}{l} \text{maximiser } y \text{ tel que } z_i \geq y, \text{ et} \\ \sum_{i=1}^{N} c_{ij} z_i \leq M_j,\ j = 1, \ldots, J \end{array}$$

Après avoir résolu ce problème, nous connaîtrons la première composante du vecteur de qualité max-min affiné. La procédure consiste alors à fixer la valeur du facteur de résolution minimum, réduisant l'espace de un, et à appliquer encore la même procédure pour obtenir le prochain minimum. Cette procédure est poursuivie jusqu'à ce que tous les facteurs de résolution soient déterminés.

Chapter 2

Introduction

2.1 Asynchronous Messaging

Multimedia encompasses the full range of visual and auditory artifacts of communications devices, such as audio, video, animation, text, still images, etc. It also includes synchronized play-out of two or more separate media elements, which we refer to as a synchronized multimedia presentation, or simply a multimedia presentation. A multimedia presentation is a generalization of an asynchronous multimedia message: it is stored multimedia data that can be retrieved and rendered by users at a remote location.

When humans communicate directly with each other using telephones, text chat, or video conferencing systems, communication is called synchronous. Typically, application developers attempt to keep the delay between source and destination to less than 100 ms, which is sometimes difficult in congested communication networks. A delay greater than this is noticeable by users, who must reduce the spontaneity of their communications; one needs to think more before speaking to determine if there will be a collision of speech if both parties start to speak at the same instant.

The public switched telephone network was designed for synchronous communication. Normally, the delay is within tolerable limits, and people can communicate easily at great distances. The packet-switched data networks that make up the Internet do not always provide the same service level as the circuit switched telephone network in the sense that data packets are lost or delayed due to network congestion, which results in quality degradation and increased delays between parties when lost data is retransmitted. There are attempts to overcome these problems with packet-switched systems. Intserv [RFC2212] and diffserv[RFC2475] are two examples where researchers are proposing various methods of controlling QoS for time-sensitive

data streams. Because of the efficiency and great savings in cost when using packet-switched networks, there is a strong push in the direction of making the Internet usable for synchronous communication.

The situation for asynchronous messaging is different from synchronous messaging in several aspects. In asynchronous messaging, users communicate with a variable period of time between message creation and message consumption (or rendering). For example, I write and mail a letter to a friend. The first delay, which is out of my control, is the delivery delay: the postal system will take a certain amount of time, usually measured in days, to deliver the letter to my friend's mailing address. The second delay is the time taken by the message recipient to access and *read* the message. For example, my friend may not actually read my letter when it arrives. Perhaps he is in a hurry to go to an appointment, and leaves the newly arrived letter on a table to be read when he returns home at a later time. Asynchronous messaging is convenient for the recipient, because the recipient can process the message at a time that is convenient for him. It is also convenient for message senders, because senders do not compete with other senders to establish a communication channel with the recipient: the sender never gets a "busy signal."

There is a third mode of communication, which is a hybrid of the two, which we refer to as *semi-synchronous*. In this case, the two (or more) persons communicating must tolerate a short, but noticeable, communications delay. This is this case with many chat systems, and with rapid exchange of email and message board postings. Similar to asynchronous communication, semi-synchronous communication also provides convenience to senders and receivers. Typically, communicating parties using semi-synchronous communication tools will be multiplexing with other tasks. In other words, the users will be doing several things at the same time, such as servicing intermittent clients, performing manual labor, while engaging in one or more semi-synchronous sessions.

It should be noted that current email systems are not typically optimally configured for semi-synchronous communications. The problem is that when the sending mail system fails to make a TCP connection with the recipient system because of congestion, the message is queued for a later delivery attempt. The typical default queuing period between retries is one hour, which is inadequate for semi-synchronous applications. Because of this problem, today's text chat or message boards are more effective semi-synchronous communication tools.

This thesis is concerned with asynchronous messaging over data communication networks. Message content is first created and stored. Then it is delivered and rendered by one or more message recipients after an undeter-

mined period of time. We pay particular attention to two issues: the delay the user experiences after issuing the command to render the media, and the resolution, or quality, of the rendered media. We would like to deliver as high a quality media stream as possible to the client, but at the same time, we would like to start playout of the stream as soon as possible. But because of bandwidth limitations, the application may need to reduce the resolution of the rendered media so that it arrives in a timely manner at the client, and so that the resulting playout of the media is smooth. Thus, there is a tradeoff between playout delay and resolution. The application will only be able to enter into this tradeoff decision if the media is scalable, that is, if it's data representation can be reduced so that the rendered version of the media is of lower resolution. Thus, the user gains more immediate playout of the media at the expense of lower resolution. If the user desires high resolution rendering of the message, then he indicates his willingness to wait for a longer period before commencing playout, and the application transmits a larger ammount of data over a longer period of time.

The thesis is divided into two parts. In the first part, we discuss design constraints and implementation issues with asynchronous multimedia messaging applications. We consider the problem of adding continuous media (audio, video, etc.) to Internet email. We also review a prototype that demonstrates the feasibility of a voice message board Web service, and illustrates the design constraints and alternatives. Central to distributed asynchronous multimedia messaging systems is the streaming delivery of stored data. In the second part, we examine streaming of stored multimedia message data. We formulate the problem of streaming multimedia presentations as a bandwidth and client memory constrained optimization. We use the term *presentation* rather than *message*, because the results apply to the larger context of delivering stored multimedia data.

2.2 Continuous Media Email

Chapter 3 examines how the Internet's email infrastructure should evolve to better support continuous media email, such as audio and video. We assert that the problem is not simply a matter of adding the obvious new functionality to user agents (mail readers), such as audio and video playback and capture, but requires the adoption of a new delivery model. We do this by examining the problems that arise using current methods to deliver video messages, and we show how sender-side storage and media streaming solve these problems. Finally, we describe an implementation strategy that requires changes only to individual sender systems, but enables continuous

media email to be delivered universally to any recipient, thus providing a realistic evolutionary path that can be adopted incrementally by individual mail systems.

Because sender-stored delivery of continuous media email represents a new paradigm for email, it naturally engenders new problems. We identify these problems, 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 continuous media 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.

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 continuous media 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 continuous media. 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 continuous media 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 continuous media 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 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, 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. This faulty cost model is partially responsible for spam.

Currently, significant 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 continuous media, this would amount to a great waste of network resources for the transport of non-rendered content. In fact, the current model encourages ISP's and individuals to continue constraining the size of incoming email messages to a low byte count. For this reason, sender-stored email makes more economic sense, because it eliminates the cost of multimedia spam. Other problems are examined.

While there are similarities and differences between multimedia email and telephone voice mail, we argue that multimedia email is superior to voice mail in several important ways. We talk about the benefits of adding audio and video to email, such as easier message creation, meeting small user interface requirements for mobile phones and ubiquitous computing (ambient user interfaces). Message rendering can be done as a concurrent task within the context of other activities.

2.3 Voice Message Boards

Text message boards take two forms in the Internet today. There are the traditional message boards that operate within the context of the Network News Transport Protocol (NNTP), and there are the more recent Web service message boards that operate in the context of the Hypertext Transport Protocol (HTTP) and the Hypertext Markup language (HTML). In Chapter 4 we describe a voice-enabled message board that we implemented as a Web service. We made this decision, because the trend is to implement distributed services as Web services. Additionally, the Web browser environment provides an interface to script and the ability to use mobile code that can support the addition of voice to the text-only message boards.

The main problem from an engineering perspective is providing the Web service to as large an audience of end-systems as possible. Unlike text data, which can be submitted with the HTML FORM mechanism, the Web browser environment does not provide an HTML mechanism for capturing and submitting audio data. To achieve this, the service provider must install platform-specific executables on user end-systems. We demonstrate the use of mobile code as a transparent means of installing the required platform-specific code to enable audio capture and transport functionality within the Web browser.

Adding audio to message boards enables one important application to expand, which is asynchronous language learning. Unlike other disciplines, which would benefit in one degree or another from an archived tree of voice/text messages, language classrooms taught in the distance education mode would benefit greatly. The reason is simple: language learning requires speaking and listening between student and teacher, and so text-based messaging systems are clearly inadequate in this context.

Other teaching environments can also be enhanced through the use of voice-enabled asynchronous messaging. For example, a class on creative writing or poetry would benefit from being able to record messages in voice as well as text. In contexts outside the classroom, people may want to express themselves more completely. Although emotion can be conveyed in text, its expression may be more complete or easier to achieve when using one's voice. In addition to improving the expressiveness of messages, audio messaging systems can be more efficient than text-only systems, because message creation can be done more rapidly.

2.4 Streaming Stored Multimedia

Underlying the application domains that include multimedia email and voice message boards, there is the fundamental need to stream multimedia data over networks. Streaming enables users to render multimedia data at their terminals without excessive delay. Streaming means the rendering of a media stream commences before the arrival of all the media data. Streaming becomes especially important when message lengths become long. It may also be necessary when client devices do not have enough memory to store the entire data representation of the multimedia message.

Many distributed multimedia systems deliver content across network paths with different levels of bandwidth. Typically, they are constrained by a minimum start-up delay and finite client buffer. In order to meet these constraints within the context of available bandwidth, the content delivery system degrades the quality of the component parts of the presentation[1], which we call *objects*, in order to reduce the size of their data representations. The motivation is to achieve on-time delivery of these objects, so that play-out does not stall for lack of data.

Multimedia presentation streaming systems can try to adapt their media flows in the context of changing bandwidth levels; however, IP bandwidth is highly erratic, and attempts to model it have only led to the conclusion that without special QoS mechanisms, adaptive schemes will breakdown at

[1] Asynchronous multimedia messages are instances of multimedia presentations.

unpredictable times for unpredictable durations. Although best-effort transmission works relatively well for delay-tolerant applications, such as email delivery and the propagation of news messages between news servers, users are inconvenienced when media streams can not be delivered smoothly with short start-up delays. To better support playout of stored and live media, researchers have been working on the problem of providing QoS guarantees to data streams.

When the network can not provide enough bandwidth to deliver the full resolution of the presentation, or if client memory is inadequate for storing pre-fetched data, the application must reduce the size of the data representation in order for the data to be delivered on time and rendered smoothly, and the end user must accept a presentation with a correspondingly degraded quality. One way size reduction is accomplished is by sending fewer layers of a layer-encoded version of the data.

A synchronized multimedia presentation consists of a collection of objects, with each object having one or more rendering intervals within the presentation timeline. These intervals specify the objects' start times and end times relative to the presentation timeline. In Chapter 5 we consider the problem of streaming a multimedia presentation from a server to a client over a bandwidth-limited communication network. We suppose that each of the static objects is layered-encoded. For a given maximum delay, we consider the problem of finding the optimal number of layers in each object in order to maximize a measure of the overall quality of the presentation. We devise efficient algorithms for determining an optimal policy for several natural criteria.

In the context of a communication network with no QoS guarantees, we describe the problem of adaptive streaming of layer encoded multimedia presentation data as a two-phase decision problem. In phase one the application transfers only base layer data that comprise a presentation of minimum quality, which is stored at the client. When the application determines that commencing play out will result in an uninterrupted presentation of at least minimum quality, it does so, and then transitions into phase two. The application then loops on the decision on which data to send next: another base layer, or an enhancement layer. We present two different algorithms for making this decision, based on two different presentation quality metrics: the *total quality* metric, which yields an optimization problem that can be solved with dynamic programming, and the *refined max-min* metric, which yields a computationally inexpensive algorithm for computing an optimal decision. We also consider the problem of progressively rendering static objects after their start times as a means of improving presentation quality. Using a slide show presentation with a randomly generated sequence of layer-encoded

JPEG images, we compare the various approaches.

Considering the increasing prevalence of wireless connections to small memory devices, such as mobile phones and PDAs, it is important to consider the case of limited client memory. We work this into our constrained optimization problem appropriately. With an infinite client pre-fetch buffer, we only need to solve for the number of layers to send for each object, since transmission of the layers is ordered by their delivery deadlines. But in the case of a finite client pre-fetch buffer, we need to solve for an additional set of variables, namely, the rate at which the server transmits data between the time intervals demarcated by the removal of data from the client buffer. We assume that the data of object i is removed from the client buffer at the beginning of its rendering interval. At these points, the application can potentially transmit more data, since the client can accept more data into its buffer. However, it can transmit at a rate that exceeds the channel capacity. Thus, the extended problem that includes a limited client pre-fetch buffer involves both a vector specifying the number of layers to send of each presentation component object, and a vector specifying the application's transmission rates over the intervals defined by the application deadlines. Assignment to these variables must meet three sets of constraints, as described in Section 5.8.

Chapter 6 assumes the multimedia presentation data is encoded using fine grained scalability. We use the term *continuously scalable* when referring to finely grained layered media, because we model it using continuous variables. We present algorithms that determine optimal transmission policies for streaming these presentations across a lossless transport channel on a network with QoS guarantees. In this context, we extend the problem by considering additional constraints that reflect limitations in client memory. We assume the multimedia data is structured as loss-intolerant, continuously scalable objects, such as JPEG-2000 images or CELP-encoded audio. The algorithms determine a scaling vector, which represents the resolution (or percentage of bits) of each component object of the presentation that is delivered to the client for rendering. We define optimality with the *refined max-min criterion* developed in Chapter 5.

Part I

Asynchronous Multimedia Messaging Applications

Chapter 3

Continuous Media Email

3.1 Introduction

Although continuous media 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 continuous media. We review these problems and explain how they are surmountable with current Web technology. Given user interest in continuous media email, and the existing Web technology to provide it, we predict the rapid emergence of continuous media 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.

3.1.1 Background

Internet email, introduced in the 1970s, was the Internet's first killer application. It is the original application that provoked user interest in the Internet, and it has now become a common fixture of modern society, both in the workplace and in the home. Up through the early 1990s, almost all Internet email was in simple ASCII. But in recent years, email user agents (mail readers) have become MIME 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 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. Thus, email systems have access to the multimedia streaming

and rendering technology that has been developed for the Web.

Although the MIME standard has been widely adopted by email user agents, the vast majority of the email messages sent today are void of continuous-media (continuous media) content, such as audio and video. The lack of continuous media content in email messages is clearly not due to lack of user interest. Indeed, there are many contexts in which continuous media email would be highly desirable. For example, a young child who cannot type (or even read) would prefer sending an audio message to sending a text message. Family members and friends may prefer to give their email a personal touch by including video. An office worker may be able to record a continuous media message more quickly than typing a text message. Possibly the greatest impetus to the widespread adoption of continuous media email will come from the proliferation of small hand-held wireless devices with Internet connectivity—devices for which voice input is more convenient than text input due to space limitations.

Although current telephony voice mail systems provide much of the benefits of continuous media email, they do not provide all of the features available to continuous media email. First, telephony voice mail is voice-only, whereas Internet email can be voice-only, or any combination of voice, text, HTML, video, animation, static images, etc. Second, email is hyperlinkable, providing the message recipient with easy access to referenced Web objects. Third, email can be used to convey information when transporting objects in the form of email attachments. Fourth, email is exclusively asynchronous—the message author and message recipient are guaranteed a temporal separation. Conversely, with telephony voice mail, the "caller" does not know whether he or she will be asked to record a message or will enter into a synchronous conversation with the "called."

Storage capacity should also not be a major hindrance to widespread adoption of continuous media email. Indeed, the cost of disk storage has been declining at exponential rates, and message storage systems with terabytes of storage are now conceivable. Also, speakers are now becoming a common component of computer systems, thus there is little hindrance from the playback side for the widespread use of continuous media messaging.

3.1.2 Benefits

Adding continuous media 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.

Continuous media 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, continuous media email would be easier to compose and render than text email. For all users, continuous media 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 continuous media 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 continuous media messages. Additionally, people speak at a much faster rate than they type. Thus, it is easier and more efficient to create messages with continuous media content rather than text.

3.1.3 Problems

If the benefits of continuous media 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 continuous media 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 continuous media email is the manner in which Internet email is currently stored and delivered, which we describe in the next section.

There are obvious inadequacies in the existing user interfaces to email

systems for the creation of continuous media email. Current email user agents lack audio and video capture functionality. To send an audio message, the sender must record a message in a separate application and save the recording as a file. Then, inside the user agent, the sender incorporates the continuous media file into the message by including it as a MIME attachment. This whole process needs to be integrated into the interface of the user agent, so that users can compose and respond to messages as easily as they do with text email.

Aside from hardware and software inadequacies within the end user's computer system, there also exists a psychological barrier to asynchronous voice and video messaging. Because asynchronous voice messaging is unfamiliar to most users, many are timid when interacting with voice-based systems, much like the initial timidity displayed by people when first exposed to telephone answering machines. This observation, along with the fact that many users may not have the necessary privacy to be comfortable when composing an asynchronous voice message, was made in [VistaMail] and [TURN99a]. However, repeated exposure to voice messaging and ergonomic enhancements for increased privacy should gradually reduce these psychological barriers.

The problems mentioned so far are only minor compared to the more fundamental problems related to the underlying infrastructure of Internet email, which are impeding the widespread adoption of continuous media email. The first contribution of this paper is to identify these inadequacies, which include: (1) a faulty cost model for Internet email, in which the recipient bears much of the cost of email delivery, (2) the duplication of message data within mail systems, which occurs when email is sent to multiple recipients within the same domain, (3) the unnecessary delivery of message data that is not rendered (or not fully rendered) by the recipient, (4) a lack of sensitivity to the end user's access rate, which results in excessive delays when retrieving large messages, and an insensitivity to the end user's storage resources, which results in undeliverable messages, and (5) inadequate access protocols, which can not be adapted to continuous media.

To resolve these inadequacies, we propose a basic scheme of *sender-stored email*, which addresses these problems by taking into account the needs and heterogeneity of continuous media 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 continuous media content of its outgoing messages, which are streamed to the recipient user agents (or media players) in the moment the recipients desire rendering. 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 continuous media server

should have the ability to transmit a compressed and layer-encoded version of the continuous media data that matches the available bandwidth.

We do not restrict our discussion to any particular operating system or application layer software. Our proposals are equally relevant for the growing community LINUX/UNIX users as well as other platforms. Of course, users that rely on older mail readers (such as mh, pine, etc.) will not be provided with the automated procedures available to the more advanced user agents, such as emacs, Netscape, etc. A more serious barrier to users of older systems is the absence of sound cards, whose users will obviously not be able to render audio data. However, these systems represent a quickly diminishing segment of the user community, and so do not present a significant barrier to universal access.

Several authors including ourselves have recognized the importance of sender-stored delivery of email. Gay and Kervella[MHEGAM] describe a multimedia messaging architecture that supports the creation and playback of synchronized multimedia objects. They describe the MEGHAM system, which is implemented on the X.400 mail standard. Hess et al.[VistaMail] present VistaMail, a multimedia messaging system that emphasizes the unified nature of a video message and relies on extensions to MIME. Along with a detailed explanation of their architecture, they include valuable observations regarding user acceptance. Schürman[SCHM] gives an overview of MIME-based and X.400-based multimedia messaging, and presents the architecture of the BERKOM Multimedia-Mail Teleservice, which is built on the X.400 standard. All three of these approaches rely on some form of distributed storage of message content, where messages include references to separately stored content. MEGHAM and BERKOM both use X.400 approaches, and VistaMail relies on NFS access to continuous media data files. Our approach is different from these researchers in that we consider continuous media email schemes that are appropriate for the global Internet. Our schemes require only incremental changes in the existing Internet email infrastructure, and they explicitly address streaming, appropriate cost models for continuous media email, user heterogeneity, forwarding, annotation, and privacy.

The remainder of this chapter is organized as follows. In Section 3.2 we explain how continuous media email differs from traditional text email. In Section 3.3 we review current delivery procedures for multimedia email. In Section 3.4 we identify the inadequacies in the existing Internet email architecture to support continuous media email. In Section 3.5 we describe our basic scheme of sender-stored email that overcomes these inadequacies, and describes a Web-based solution that can be implemented incrementally in individual sender systems. Section 3.6 investigates security issues, and describes our proposals to ensure message privacy and integrity. The final sections of

the chapter present our solutions to the problems introduced by the sender-side storage paradigm. Sec. 3.7 describes our integrated recipient/sender-stored delivery solution to the QoS problem. Sec. 3.8 details our proposals for the problem of message deletion. In Sec. 3.9 we provide methods for message forwarding and replying.

3.2 How Does Continuous Media Email Differ from Text Email?

Given that there is significant user interest in continuous media email, what changes need to be made in the Internet email infrastructure so that continuous media email becomes more widespread? To answer this question, we need to look at how continuous media email differs from traditional text email. The most salient difference is that a continuous media message, and in particular a video message, is much larger in byte count than a text message. To roughly quantify this difference, consider a message that consists of 180 words. A text version of this message is about 1 KB. For a person who speaks at a rate of 180 words per minute, a 24 Kbps encoding of an audio version of this message is 180 KB. An MPEG-1 1.5 Mbps encoding of a video version of this message is 11,250 KB. Thus, the byte count of an audio message is roughly 100 to 200 times larger than a text message, and the byte count of a good-quality video message is roughly 10,000 times larger than a text message.

The second difference is that continuous media email has more stringent QoS requirements than traditional text email. When a recipient selects a continuous media message for rendering, playback should begin within a few seconds. Once playback begins, the audio and video quality should be good, and the playback should not be plagued with interruptions due to packet loss or client re-buffering. Moreover, the recipient should be able pause playback as well as make temporal jumps forwards and backwards within the message.

The third difference concerns the heterogeneity of user environments. For example, although some users today have high-bandwidth connections to the Internet, 98% of residential users access the Internet at dial-up modem speeds of 56 Kbps or less[MCPH98]. The fraction of low-bandwidth modem connections will remain significant for the foreseeable future, as many communities (particularly rural communities) may never get wired for emerging cable and ADSL residential access. In addition to heterogeneous access rates, users are also heterogeneous in terms of local storage. At one extreme are the thin clients with minimal local storage, such as hand-held wireless devices. At

the other extreme are systems with tens of gigabytes of storage. The issue of heterogeneity is critical for Internet email because arbitrary collections of user environments need to inter-operate with each other. Although this heterogeneity issue is also present for traditional text email, the issue is of much greater importance for continuous media email due to its size and QoS requirements.

3.3 Barriers to the Development of Continuous Media Email

Internet email requires that an arbitrary sender be able to send a message to an arbitrary recipient. This is possible, because of universal agreement regarding the format of messages and their method of transport. In general, an Internet email message is 7-bit ASCII text data that conforms to the specification given in RFC 822, and its transport is accomplished using Simple Mail Transport Protocol (SMTP). These two standards form the basis of Internet email. However, the RFC-822 message format is inadequate for messages that contain anything other than a single body of ASCII text. To provide for multiple-body messages composed of any data type, the Multipart Internet Mail Extension (MIME) was developed. (Reference [HUGH] gives a thorough introduction to Internet email.)

In order to allow incremental adoption by email systems, the MIME format was designed so that MIME-compliant messages also comply with the syntax rules of RFC 822 messages. This was accomplished by dividing the single body of an RFC-822 message into multiple parts, and encoding non-ASCII data into ASCII. Special headers are used to identify subdivisions of the body, the type and format of the data contained within each subdivision, and the ASCII encoding scheme that was applied to the data. For binary data, such as audio and video, the Base 64 encoding is used. The result of this encoding is to increase the size of the data representation by 33%; thus, continuous-media messages, which are already large, become even larger when transported in ASCII encoded form.

Because a MIME message is also an RFC-822 message, it can pass into the recipient's mailbox through its SMTP server. From a transport and storage perspective, SMTP and the MIME standard together enable multimedia messaging in the Internet. However, other than sending HTML documents and images, multimedia messaging through MIME and SMTP has not become commonplace. In the next section we examine the major barriers to its development.

Let's consider an example in which the MIME message format is used in conjunction with SMTP to deliver a video message. Suppose that Alice recorded herself speaking for one minute using MPEG-1 at 1.5 Mbps. She saves the file with the name "12345.mpg." In her user agent she types a short text message to supplement her video message and attaches the video file to her message. She addresses the message to Bob and presses the send button. First, we will examine the structure of the message that her user agent creates, and then we will examine how it is transported to Bob's mailbox. (Throughout this paper, we assume that the sender is Alice and the recipient is Bob.)

Fig. 3.1 contains the message that Alice's user agent created. It follows the organization scheme of RFC-822; because there is a header section, followed by a blank line, followed by a body. The header section contains the various pieces of meta-information, such as an identification of the sender, the recipient, message date, etc. In particular, note the presence of the *Content-Type* header, which declares that the body is of type multipart/mixed. This header also defines a *boundary* attribute, which specifies a string of characters used to demarcate the beginning and ending of the various body parts.

In our example, there are two separate parts within the main body. Each part follows the basic structure of a body part; it has a header section, followed by a blank line, followed by a body section. The boundary string is used to begin each part. The final part is also followed by the boundary string, but it is appended with two dashes, "--".

The first part contains a single header indicating that the content-type of the first body part is text/plain. The body contains the text created by Alice. The Content-Type of the second body part is video/mpeg, indicating that the body contains an MPEG data file. The Content-Transfer-Encoding header indicates the data has been transformed into simple ASCII text using the Base 64 algorithm. The Content-Disposition header indicates the recipient user agent should treat the data carried by this body part as an attachment, which means the user agent should not attempt to interpret the data for immediate display, but should present the user an option to open the attached data. The filename attribute is defined within the Content-Disposition header as a suggested name for the user agent to display to the user.

The route Alice's message takes is shown in Fig. 3.2. First, Alice's user agent transfers the message to a mail transfer agent (MTA) provided by her mail service provider. Alice's MTA then relays her message by SMTP to Bob's MTA, that is, the *mail server* that accepts messages into Bob's mailbox. These two steps comprise the push phase of message transport. The final step of transport occurs when Bob runs his user agent, which then

```
From: "Alice Adams" <alice@aaa.com>
To: "Bob Brown" <bob@bbb.com>
Subject: new product announcement
Date: Tue, 9 Feb 1999 13:18:45 +0100
MIME-Version: 1.0
Content-Type: multipart/mixed;
     boundary="--myBoundary"

--myBoundary
Content-Type: text/plain

Bob,
Tell me what you think.
     Alice
--myBoundary
Content-Type: video/mpeg
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
     filename="12345.mpg"

GdAfbBgA4A6B8kAIA+A2AlwB8ybABmYWNfmA
8kX67vi/6I/n7+ZP62mu3+Kf+p/yUA8gBqAb
  . . .
--myBoundary--
```

Figure 3.1: Message with video attachment

retrieves the message from his mailbox using a mailbox access protocol.

Because the message in our example is a one-minute 1.5-Mbps MPEG video clip, its size (after Base 64 encoding) is 15 MB. In today's environment, such a message is frequently too large to pass into the recipient's storage, either because the mailbox lacks available space, or because messages of this size are rejected by policy. If both servers support Extended SMTP, then Alice's MTA announces the size of the message before it attempts transmission. This allows Bob's MTA to reject the message prior to transmission, rather than mid-stream. However, for the sake of the example, we will assume that Bob's MTA accepts the message and places it in Bob's mailbox storage.

Different user agent systems accomplish message transport to and from the user's mail service using different protocols. The first distinction we

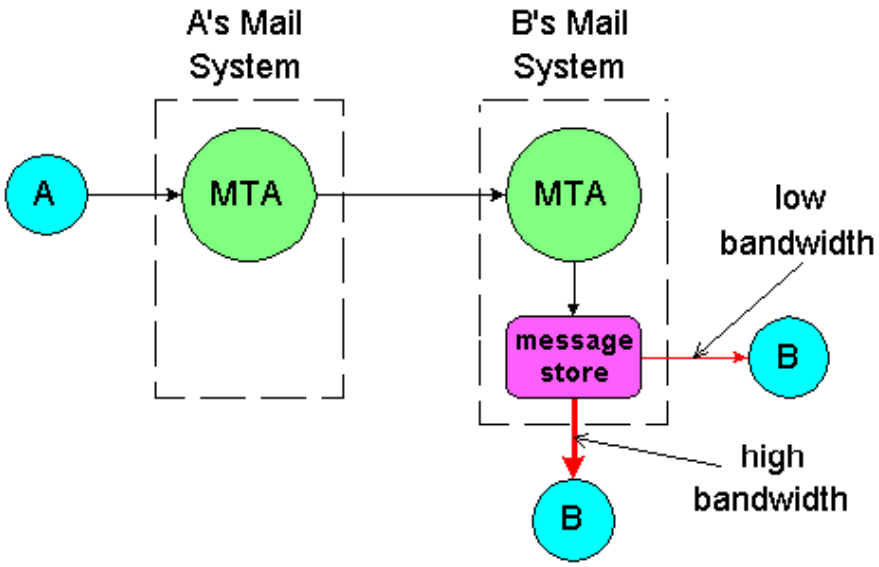

Figure 3.2: The current email delivery model

make is between *standalone* and *Web-based* user agents. In a standalone user agent system, a special purpose application runs on the user's computer and communicates intermittently with the remote mail system. It uses SMTP to upload messages to the user's outgoing mail server, and an access protocol, such as POP3 or IMAP4, to retrieve messages from the user's mailbox. Examples of standalone user agents include Netscape's Messenger, Microsoft's Outlook, and Qualcom's Eudora. In Web-based user agent systems, the remote mail system provides an interface to the user by sending HTML documents to the Web browser. Messages are thus sent and retrieved over HTTP. Examples of Web-based user agent systems include Hotmail and Yahoo! Mail. Fig. 3.3 graphically depicts the difference between standalone and Web-based user agents.

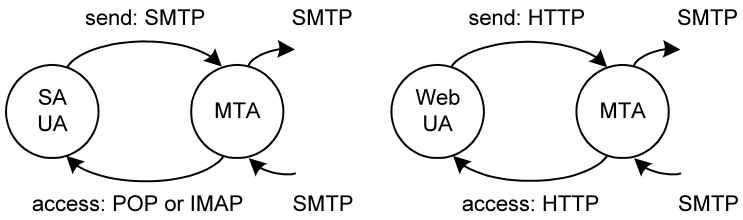

Figure 3.3: Standalone and Web-based user agents

Generally, standalone user agents transfer outgoing messages using SMTP.

However, the retrieval of messages is usually accomplished by Post Office Protocol (POP) or by Internet Message Access Protocol (IMAP). POP is designed to be a simple retrieve-and-delete method of mailbox access, where the location of stored message data is in the user's local computer. IMAP provides the ability to leave one's messages in remote storage, organized into a hierarchy of folders. The advantage of IMAP is that one's messages can be accessed from more than a single computer.

Web-based user agents don't run as standalone applications in the user's computer, but operate within a Web browser. The user agent can be thought of as running at the remote server site, which generates and sends Web pages that provide the user interface to one's mailbox. These interactive Web pages provide all the user agent functionality through HTTP exchanges with the server, including the reading of one's mail and the sending of newly created messages to other users.

A Web interface for text email is possible, because HTML defines input mechanisms to capture keyboard and mouse input and deliver it to an HTTP server. Within the browser window, the user enters text into textboxes, and clicks on various selectable objects, such as checkboxes, radio buttons, etc. The user submits his input to the server by clicking on a "submit" button, and the browser sends an HTTP POST request to the server that includes the values representing the user's actions. In this way, the remote mail system performs whatever user agent function is desired by the user, such as creating a new mail folder, deleting an old message, or sending the user's entered text as an email message to another user.

Web-based systems provide the greatest mobility to the user, because the end user need only have access to a Web browser in order to interact with the mailbox and create and send new messages. Similar to the IMAP approach, messages are kept in remote storage and are organized into a hierarchy of folders.

As an example, suppose that Bob's computer is on a 100 Mbps LAN, and his mail server is also located on this LAN. His user agent is configured to check the server every five minutes for the presence of new messages. When a new message is detected, it is immediately retrieved over POP and deleted from the remote store. When Alice's video message (Fig. 3.1) is detected, Bob' s user agent will consume approximately 1 second of LAN bandwidth to copy the data from the mail server's message store to his local storage.

As a second example, suppose Bob connects to his mail server over the Internet with a 28.8 Kbps modem instead of over a LAN. In this environment, his user agent will take one hour and ten minutes to retrieve Alice's video message, regardless of the access protocol it uses. Even if his mail service provider is willing to store such large messages in his mailbox, it is not

practical for Bob to spend the time and money retrieving them. Another possible barrier is that he doesn't have 15 MB of available disk space to store the video data.

3.4 Inadequacies of the Existing Internet Email Infrastructure

There are several problems in the underlying infrastructure of Internet email for supporting continuous media:

- Email suffers from a faulty cost model.
- The storage architecture results in excessive data redundancy.
- Resources are wasted in the transmission and storage of unread, or not fully read, message data. (In the case of continuous media email, we might say "un-rendered," rather than "unread.")
- There are long delays when retrieving continuous media messages in low bandwidth environments.
- Message delivery to the end user is insensitive to the end user's access rate and storage resources. Message delivery is not universally possible, because of storage limitations in the message store of the recipient system.
- The access protocols are inadequate.

Faulty Cost Model

The costs associated with email include (1) consumption of bandwidth when transporting messages into recipient mailboxes, (2) storage of the message until it is deleted from the message store, and (3) consumption of bandwidth when transporting the message from the recipient's message store to the recipient's local machine. These costs are indirectly paid by individuals through subscription payments to an ISP, and possibly telephone usage payments to the telephone company in the case the user's access to the Internet is by modem. Since the sender only pays for a portion of the bandwidth expense, and not for temporary storage of message data, the recipient actually pays more to receive email than the sender pays for sending it. This is contrary to the cost model of ordinary mail, where the sender bears the entire cost of a letter.

Although it is not the only cause of spam[RFC2635, DERN], this faulty cost model tends to encourage its proliferation. Although it is becoming more feasible to provide users with the required disk storage for continuous media email, there is a reduced incentive to do so, since this resource would likely contribute to the development of audio and video spam.

Data Redundancy

One impediment to continuous media email is the current storage architecture, which results in excessive data redundancy within mail systems. In corporations and large-scale ISPs, a single message is frequently distributed to many other recipients within the same mail system. In this case, an exact copy of the message data is duplicated in storage for each recipient, and statically stored until deleted. Forwarded messages and annotated replies also result in further duplication of message data. Thus storage resources, which include active disk space and their protective back up systems, are needlessly consumed with redundant static data.

Wasted Resources

Recipients of continuous media content will often only want to render small fractions of messages [PADH]. Recipients will question paying for the delivery and storage of entire continuous media messages when they only render small fractions of messages they receive. Furthermore, in the case of distribution lists, a recipient may not render any of the continuous media content for certain messages, particularly for messages containing continuous media spam. Thus, recipients (or recipients' ISPs) may continue limiting the size of messages they are willing to accept into their message stores, even as bandwidth and storage resources continue to expand.

A closely related problem is the waste of Internet backbone bandwidth in the current mail storage paradigm. Because continuous media email is delivered in its entirety to the recipient's mailbox, backbone bandwidth is wasted when the recipient only partially renders the message. This problem is exacerbated with distribution lists, which would send a copy of the continuous media data to each recipient. This waste will have an impact on the cost of the Internet—a cost shared by all Internet users.

Heterogeneous Users

Users are highly heterogeneous in terms of their access rates. The recipient of a short video clip who has T1 access to the Internet might have no

problem retrieving the message data, but for a person with modem access, the message retrieval delay would be excessive. It is not always possible to know in advance the transmission rate over which the recipient will be retrieving his mail, because users may access their mailboxes from different computers. Unlike the Web's continuous media servers[SURE], there is currently no mechanism within email that allows the recipient to tradeoff quality degradation in exchange for better response time.

Many existing message stores do not have the capacity for storing continuous media messages that include video or good quality audio. Thus, recipient mail transfer agents typically reject such messages, which makes sending continuous media messages to such recipients impossible. Although this restriction should eventually ease as disk storage continues to increase, ISPs may choose to continue rejecting large messages to conserve both storage and bandwidth costs in order to keep their subscription prices as low as possible, and thus remain competitive. For example, users of Yahoo's free email service have a storage limitation of 3 MB. All incoming messages that exceed 3 MB are rejected, and Yahoo users can not send email with attachments that exceed 1.5 MB.

Inadequate Access Protocols

The message store access protocols, such as POP and IMAP, treat message data as discrete objects, which are transferred in their entirety before any rendering of them is started at the user's local system. Thus recipients must suffer start up delays when rendering messages with continuous media, which is particularly problematic for users behind low-bandwidth connections, such as modems and many of the emerging wireless devices.

Recipients of continuous media content will want to pause/resume playback and make temporal jumps within the message. Although the IMAP standard allows a user agent to fetch a byte range of a message, greater functionality, such as that provided by Real Time Streaming Protocol (RTSP), may be necessary to support satisfactory user interactivity.

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 continuous media email.

3.5 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* [TURN00a] that relies on current Web technology, and that takes into account the needs and heterogeneity of continuous media 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 continuous media content of its outgoing messages, which are streamed to recipient user agents (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 continuous media server should have the ability to transmit a compressed version of the continuous media data that matches the available bandwidth.

We use the term *streaming* to describe a client-server system of continuous media delivery in which the client initiates rendering of a stream of continuous media data while it is in the process of retrieving the stream from the server. Streaming delivery of continuous media 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 continuous media data to match the bandwidth available between server and client. Adaptive streaming is used to deliver continuous media with the highest possible quality under existing bandwidth constraints. In this chapter, we always mean adaptive streaming whenever we use the word streaming.

Streaming continuous media delivery provides: (1) playback delays not exceeding a few seconds, (2) good audio or video quality without interruptions during playback, and (3) user interaction that includes pausing, temporal jumps and playback rate adjustment. In order to accomplish these objectives, the continuous media should be layer-encoded, so that the delivery system can send only those data layers that result in an encoding rate that is less than the available bandwidth. For example, suppose the stored media is comprised of 9 layers, with each layer containing data that is rendered at 25 Kbps. If the available bandwidth is 55 Kbps, then the continuous media delivery system should send the first 2 layers, resulting in a data rate of 50 kbps. Sending any more data would result in buffer starvation at the client if the continuous media were to be rendered without a significant startup delay.

Under the sender-stored delivery model for continuous media email, the continuous media portion of the message is placed in a continuous media

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 user agent uses the base message to instantiate an appropriate media player to stream the continuous media from the sender's continuous media media server.

Fig. 3.4 illustrates the sender-stored delivery process. To better understand this process, suppose Alice (A) sends Bob (B) a video message. Alice's user agent transfers the message in bulk to her outgoing mail transfer agent (MTA). This MTA then places the continuous media data with its continuous media 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 continuous media server of Alice's mail system.

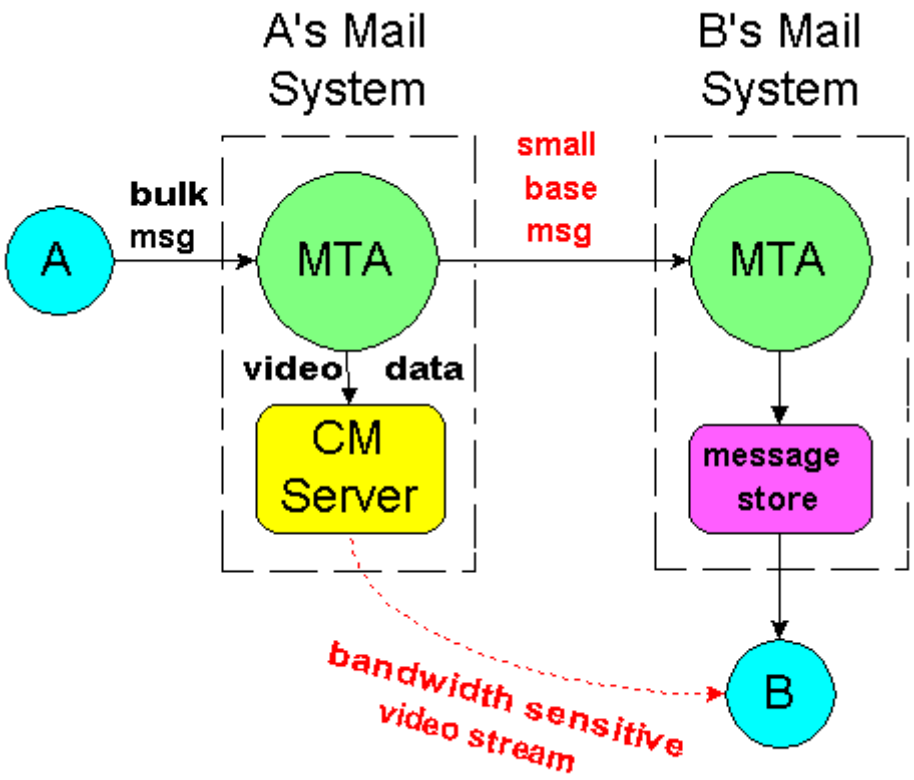

Figure 3.4: Sender-stored continuous media email

Several other research teams have also considered forms of distributed storage for continuous media email similar to our sender-stored scheme. The multimedia messaging system prototypes of the BERKOM Multimedia Mail Teleservice [SCHM], VistaMail [VistaMail], and MHEGAM [MHEGAM] 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 continuous media message forwarding, replying and annotation.

In recent years, email user agents have become MIME [MIME] 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 continuous media email in which the sender delivers a hyperlink to remotely stored continuous media message data. When the hyperlink is clicked upon, 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, such schemes are already appearing in the marketplace ([WIMB] and [ONEB]). As continuous media messaging becomes more widely used, we expect that user agents will be able to render sender-stored continuous media messages within their own window in order to present the message as an integrated part of the user's messaging system.

3.5.1 Implementation of Sender-Stored Continuous Media Email

We now describe by way of example how sender-stored continuous media 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 user agent transfers the

message data to her outgoing MTA. The MTA then stores the continuous media portion of the message with its continuous media 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 user agent 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 continuous media.

In order to deliver sender-stored continuous media email with adaptive streaming to an arbitrary recipient, the base message needs to be processed by a media player outside the recipient's user agent. 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. (Alternatively, a plain text message can be sent that contains a URL pointing to the secondary object. In this case, the user agent is responsible for rendering the URL as a hyperlink, which is the common approach within user agents.) When the recipient clicks on the link, his user agent will instantiate its companion Web browser and have it send the HTTP request to process the hyperlink. When the sender's Web server receives the HTTP request from the recipient's browser, it will be able to ascertain the browser type, its version number, and the operating system on which it is running. With this information, it can tailor a response that is suitable for the recipient's environment. Unfortunately, knowledge of the available bandwidth between continuous media server and the recipient can not be known in advance, and so the continuous media server must begin transmitting data with an initial *conservative* bandwidth assumption or via pre-configuration of the recipient's continuous media agent, as is done with the RealAudio agent. While transmission proceeds, the application can adjust the start up delay and rate of data compression as it refines its bandwidth estimate.

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 continuous media stored with the sender's continuous media 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 continuous media server to stream and render the continuous media.

In both examples, the base message would look something like that in Fig. 3.5 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 3.5: Base message referencing secondary object

Most likely, Bob's user agent will display the URL in Fig. 3.5 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. 3.6 is one such possibility for a response.

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.

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

Figure 3.6: Secondary object with applet

Additional security would require the use of encryption and certificate-based recipient identification.

Whether a plug-in or applet player is used in all of these examples, an RTSP session is established with the continuous media server that provides access to the continuous media in the sender's outbox. The player then issues RTSP commands to set up and control a UDP-based transport channel to deliver the video data with real-time properties[RTSP]. Within seconds of clicking on the play button, the video is rendered to Bob. Because RTSP is being used to control the data stream, Bob can pause, rewind, fast-forward, and jump to arbitrary points within the stream using the media controls available to him.

3.5.2 User Agent Design for Sender-Stored Email

To make sender-side delivery possible to all possible recipients, we propose a design that makes changes only to sender systems and leaves recipient systems unchanged (except for possibly a one-time automatic installation of a media player in the form of a plug-in, ActiveX control, etc.).

In the case of a standalone user agent system using POP or IMAP, we propose two basic design alternatives. Our first proposal (illustrated in Fig. 3.7) is to leave the outgoing MTA unchanged, and use it simply to forward the base message as it would a usual text message. In this design, a continuous media server would need to be running, and the sender's user agent would require permission to write into its storage. The user agent would be responsible for selecting the name of the media file, encoding the continuous media data into the format required by the media server, and constructing the base message that references the continuous media it has placed within the storage of the continuous media server.

Our second proposal (illustrated in Fig. 3.8) for the standalone user agent system is to have the user agent format the message as if it were going to be delivered in bulk to the recipient, using the MIME format as illustrated in Fig. 3.1. When the outgoing MTA receives the message, it extracts the

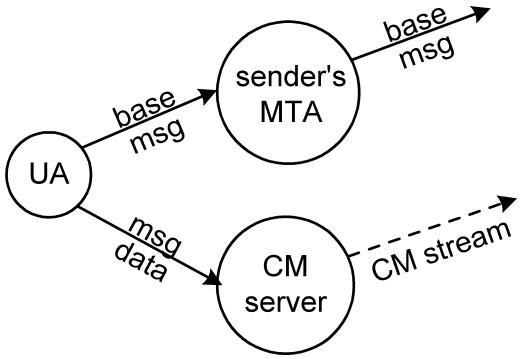

Figure 3.7: User agent formats sender-stored message

continuous media data from the body of the message, and saves it as a file in the storage of the media server. It then constructs a base message, which it delivers to the recipient. By using attachments, a simple version of sender-stored delivery could be implemented without changing existing standalone mail clients.

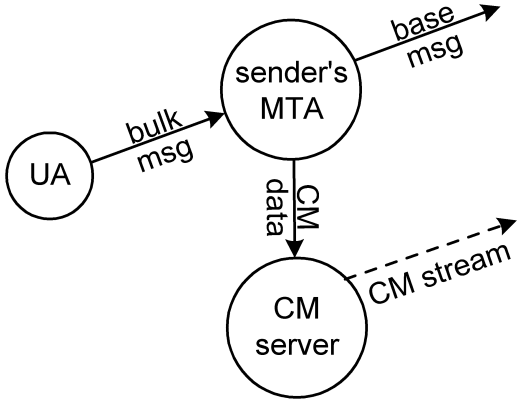

Figure 3.8: MTA formats sender-stored message

In the case of a Web-base user agent system, there is no real separation between the outgoing mail server and user agent, and thus the two design alternatives mentioned above do not apply. The difficulty with Web-based user agents is performing audio and video capture. An HTML interface for text email is enabled by the FORM and INPUT tags, which allow the user to enter text data, select checkboxes, etc. The user's input is conveyed back to the server with an HTTP POST request on execution of the FORM's SUBMIT function. An analogous process is not available for user audio or video input, because HTML tags have not been defined and implemented

to provide for audio/video capture and transmission to a remote server. A method of extending the HTML user input mechanism to include input from generic devices has been proposed in an Internet Draft[SALS]. If this mechanism were implemented in browsers, Web-based systems could be extended in a straightforward manner to enable audio and video message input. In the meantime, Web-based systems need to rely on the installation of capture software in the form of a plug-in, ActiveX control or Java applet. (Version 2 of the Java Media Framework enables audio and video capture from within Java applets. At the time of this writing, a beta-release of Version 2 is available[JMF].)

3.5.3 Benefits of Sender-Stored Delivery of Continuous Media Email

Sender-side storage combined with streaming solve the fundamental problems of Internet email described in Section 3.4 for supporting continuous media. Because the sender stores the continuous media 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. Thus, the problem of universal message delivery is solved.

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.

Because the continuous media is adaptively streamed, senders can deliver continuous media 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 continuous media messages.

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. Thus, bandwidth is not consumed in transmitting non-rendered continuous media, nor is disk storage wasted in holding such data.

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.

3.5.4 Problems with Sender-Stored Delivery of Continuous Media 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 continuous media 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 continuous media data to the recipient's MTA in the usual manner, and then the recipient streams the continuous media 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 continuous media 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 continuous media 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 continuous media 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 continuous media data from the origin sender's continuous media 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 continuous media data into the storage of his own continuous media server, and sends a reference to this copy rather than a reference to the continuous media 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.

3.6 Security and Privacy

Because of the ease by which data communications on the Internet can be intercepted, users are increasingly encrypting their email messages to ensure privacy. Additionally, users are becoming more wary of the potential for computer break-ins, which provide intruders with access to the contents of email messages. In this section, we describe a procedure of protecting sender-stored continuous media email from security threats such as break-ins, communication monitoring and man-in-the-middle attacks.

In the following discussion, we assume the reader is familiar with public key cryptography and methods of obtaining reliable public key certificates [KAUF]. Developers of sender-stored email systems can rely on the extensive support for security that is now provided by Web servers, browsers, user agents, and encryption libraries.

One way to secure sender-stored continuous media email is to encrypt the continuous media data with the public key of the recipient, and store the continuous media in encrypted form with the continuous media server. Then, only the recipient can decrypt the continuous media data with his private key. Such an approach would secure the message data from being rendered by an intruder that has broken into the continuous media server, and by attackers who have access to the communication channel between the continuous media server and message recipient. However, there are two drawbacks to this approach. First, decryption using an asymmetric key is time consuming, and would most likely introduce a significant delay when rendering the continuous media. Second, if there were multiple recipients of the message, then a separate copy of the continuous media data would need to be stored with the server for each recipient.

To avoid the problems that result from encrypting the continuous media with the recipient's public key, a symmetric key can be used instead. In this case, the sender's system locally generates a random secret symmetric key using a pseudo-random number generator seeded with a secret known only to the sender. This symmetric key is used to encrypt the data to be stored with the continuous media server. The recipient's public key is then used to encrypt the symmetric key for secure delivery to the recipient in the base message.

In the case that there are several recipients of the message, the symmetric key can be securely delivered to all of them using their individual public keys. Note that all forward recipients of the base message will also have access to

the continuous media data, because they will have the required symmetric key.

Assuming the secret symmetric key is long enough, possession of the continuous media data would be useless to a person without the decryption key. Thus, there is no need to authenticate access to the continuous media server. Nevertheless, requiring authentication may be prudent in that it adds one more barrier to a potential attacker. Also, it can control whether forward recipients are allowed access to the data through the sender's continuous media server. If the sender doesn't want to service requests from forward recipients of the base message, then the server can require that the client prove it is one of the originally intended recipients of the message. Alternatively, if the sender is willing to service requests from forward recipients of the base message, then the server can require that the client prove it has possession of the decryption key.

Other measures are required to fully secure the system described above. For instance, a virus could be running in the sender or recipient computer that intercepts the continuous media data in its non-encrypted form as it passes through the sender's media capture system or the recipient's media rendering system. Other potential threats include inadequate erasure of the decrypted symmetric key or the non-encrypted continuous media data from the memory or persistent storage of either the sender or recipient systems, and exposure of the sender's secret used to seed the pseudo-random number generator. These and other potential threats would also need to be guarded against by careful implementation of an overall security solution for the involved systems.

3.7 Integrated Recipient/Sender-Stored Email

3.7.1 Pure Recipient-Stored Delivery

Streaming continuous media message content from the sender's continuous media delivery system raises a QoS issue, because the network path between the sender's continuous media server and the recipient may be congested. Under congestion, the server will only be able to transmit a highly compressed version of the continuous media, 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 continuous media email, where the continuous media 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. 3.9, 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 continuous media data can be extracted from the message and given to a continuous media server under the control of the recipient's mail system. The message that is retrieved by the recipient's user agent via POP, IMAP or HTTP will be the base message referencing the separately stored continuous media 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.

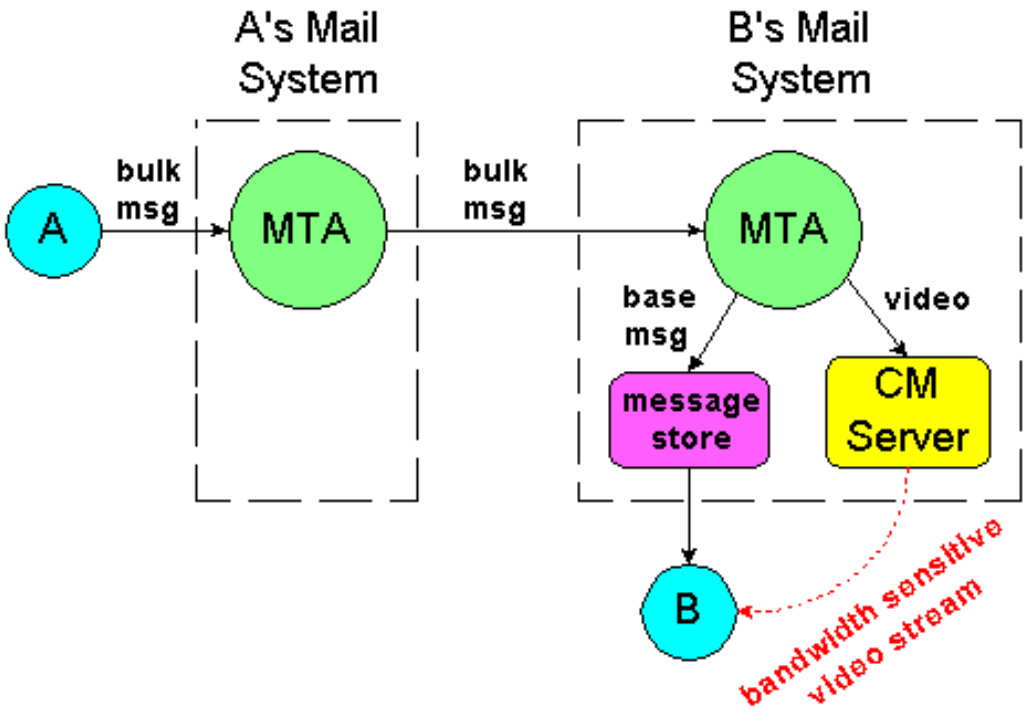

Figure 3.9: Recipient-stored continuous media email

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

within the recipient 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 continuous media, it extracts the continuous media data, removes any base-64 encoding, and then stores the resulting continuous media data in storage accessible by the system's continuous media server. To create the base message, the filter replaces the continuous media data in the original message with a reference to the continuous media 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 continuous media message, the message is streamed from the continuous media 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 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, 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 continuous media data in storage under the control of the recipient.

3.7.2 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 continuous media 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 most recipients are expected to render very little of the continuous media 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 *continuous media-aware*, that is, if it will stream the continuous media 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 continuous media-aware by using Extended SMTP [ESMTP]. Alice's MTA could have queried Bob's MTA with the ESMTP protocol exchange shown in Fig. 3.10. Bob's MTA responds that it supports the extended functionality identified by the keyword continuous mediaAWARE.

```
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 3.10: 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 continuous media-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 continuous media messages from known senders to pass into their allocated storage within the message store.

3.8 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 continuous media 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 continuous media data.

Even if the sender were to know how many outstanding references existed to a continuous media 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 continuous media 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 user agents, because the remote store can delete continuous media 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 continuous media. When a message in local storage is deleted by the user, the user agent does not inform the remote store of the deletion. Therefore, non-deterministic methods of continuous media deletion are also relevant for recipient-stored messages that are accessed through POP.

When continuous media 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, continuous media 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 continuous media. Thus, whenever a copy of a base message leaves the mail system, a non-deterministic policy of continuous media 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.

3.8.1 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 user agent, 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 continuous media she has created and makes available to recipients through her system's continuous media server. When she deletes a base message from her mailbox, her mail system also deletes the continuous media 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 continuous media 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 being used by the continuous media 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 continuous media (and incoming messages) the user deletes from her mailbox those messages she considers expendable. The user agent should respond by deleting both the base message and the continuous media 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 continuous media lifetime. If a recipient of

sender-stored continuous media wishes to be able to access the continuous media 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 continuous media into storage that is under his control. If the recipient desires to move the continuous media into his own storage, then the sender ought to provide a lossless mechanism of continuous media transport, so that the recipient can obtain a high quality copy of the message.

3.8.2 FIFO Deletion

One approach to message deletion is a simple first-in/first-out (FIFO) queue of continuous media data. Under this approach, the sender has a fixed amount of storage reserved for her outgoing continuous media 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.

3.8.3 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 continuous media 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. 3.11. 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, continuous media is retired from the expendable queue in the order of oldest first.

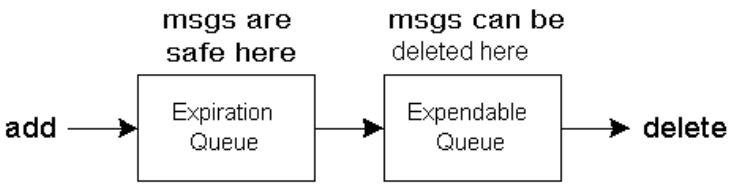

Figure 3.11: Message lifetime queues

Expiration information can be supplied in both human and machine-readable form. The human-readable expiration date allows recipients to manually copy continuous media 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 continuous media that has not yet been rendered—or continuous media 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 continuous media 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. 3.5:

```
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 continuous media. The second component of the header is the time the continuous media expires.

3.8.4 Continuous media access statistics

Both manual and automatic deletion mechanisms can be enhanced with the use of *continuous media access statistics*. When a recipient accesses the continuous media of a referencing message, the sender's mail system can make a record of this access. Because continuous media 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 continuous media data file was streamed. A less detailed mechanism might simply mark whether or not any part of the continuous media data was streamed.

In the manual deletion approach, the user agent can display the access statistics for continuous media 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 continuous media message. For example, after Alice's continuous media 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 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. 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 user agent, the command to delete the message is processed by the remote mail system, which can thus delete both the base message and its referenced continuous media. But if she is using a POP-based system, her user agent would only be able to delete the base message from its local storage; to delete the continuous media from the remote storage of the continuous media server, a new protocol between user agent and mail system would be required.

In the automatic deletion system, the access statistics can be used to order the continuous media in the expendable queue, so that the oldest continuous media is not necessarily the first to be deleted. continuous media that has not yet been retrieved could be given higher priority for retention over newer continuous media 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 continuous media.

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 continuous media. The continuous media server then maps each of these different URLs to the same continuous media 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 continuous media server will map requests for these files to the same object, 12345.mpg. When Bob reads his message, Alice's continuous media 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 continuous media 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 continuous media 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 continuous media file, and retain the continuous media 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.

3.9 Forwarding and Replying

3.9.1 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 example, suppose that Bob has a base message from Alice, and that he has just viewed its video by streaming it from Alice's continuous media 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 continuous media data; it is possible that the original author of the message deletes the continuous media before the forward recipient has a chance to render it. Fig. 3.12 illustrates unreliable forwarding. In this example, Alice first sends Bob a sender-stored continuous media message. When Bob instructs his user agent 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 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 continuous media, 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 continuous media data. Fig. 3.13 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 continuous media 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 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

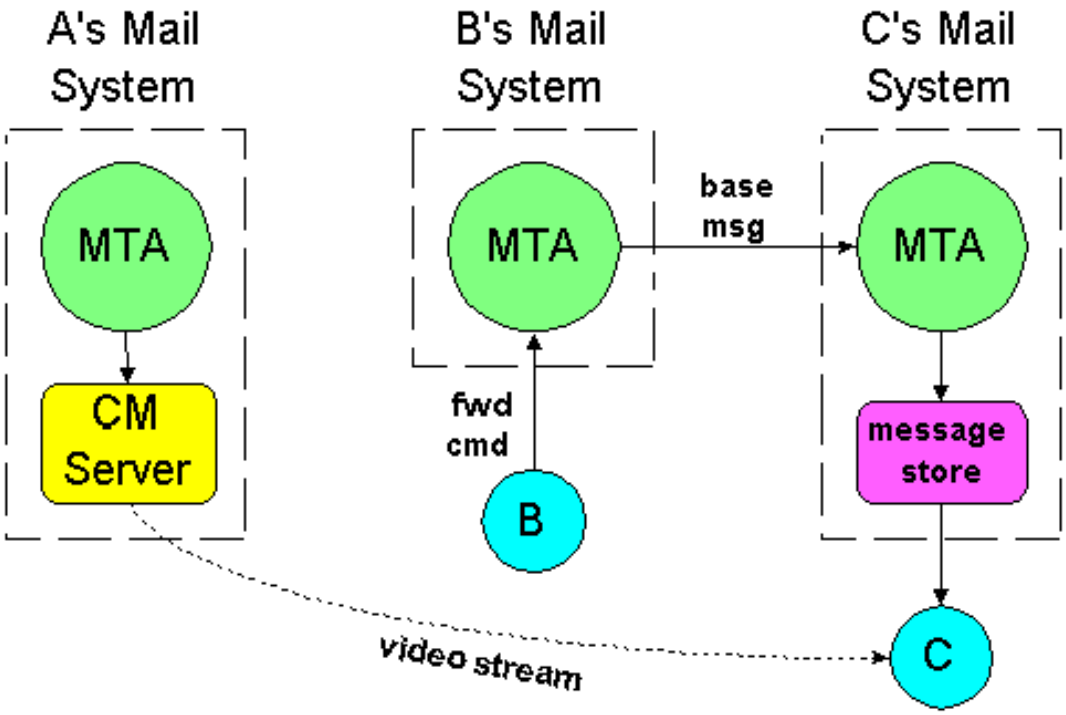

Figure 3.12: Unreliable forwarding of a continuous media message

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 continuous media 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 continuous media 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 continuous media 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 user agent to copy the continuous media referenced within the base message into his system's message store. When Bob's user agent issues this instruction, his mail service retrieves the

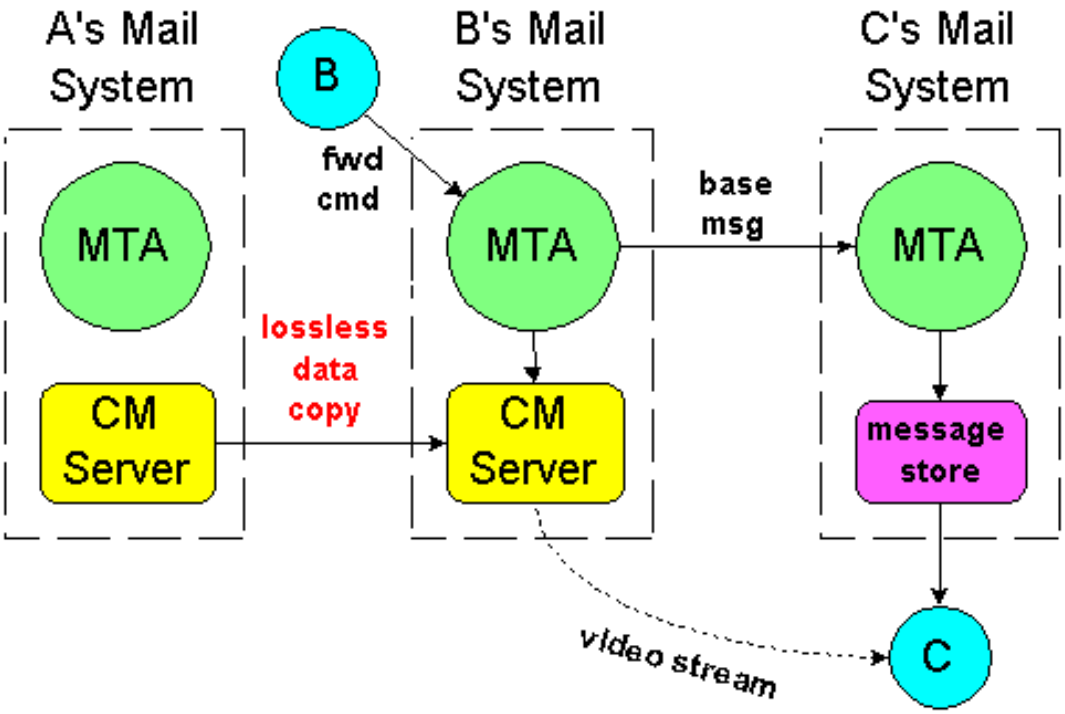

Figure 3.13: Reliable forwarding of a continuous media message

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 user agent as a base message, only it now resolves to continuous media data as it is stored in his mail system.

Adding new functionality that involves communication between user agent and mail systems, as 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 accomplished together. 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.

3.9.2 Replying

Frequently, recipients include the original message, or pieces of the original message, in their replies. It's possible to do the same with continuous me-

dia email. Let's look at an example. Suppose that Bob plays Alice's video message by streaming it from her continuous media 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 user agent 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[SMIL] file depicted in Fig. 3.14. Notice that the link pointing to the embedded video data is a reference into the storage of Alice's mail system. (See Appendix B for a short description of SMIL.)

```
<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 3.14: The SMIL document of a sender-stored annotated reply

3.10 Summary

In this chapter, we identified the major weaknesses of Internet email that obstruct the development of continuous media messaging. These include a faulty cost model, in which the recipient bears much of the cost of email delivery; the duplication of message data within mail systems, which occurs when email is sent to multiple recipients; the wasteful delivery of non-rendered message data; and a lack of sensitivity to the end user's access rate, which results in excessive delays in retrieving large messages. We showed how all of these problems are solved with a sender-stored message delivery architecture, where a small text message (base message) is sent that allows the recipient system to stream the continuous media message content from the sender's storage.

In order that our sender-stored email delivery architecture be backward compatible with existing systems, we described an implementation in which changes are only required to sender systems. We described the three major Internet user agent systems (POP, IMAP and Web-based), and we discussed how each of these systems can accommodate sender-stored email. We also described how existing security mechanisms based on public key cryptography can be use to ensure the privacy and integrity of sender-stored continuous media email.

Because sender-stored email is a major paradigm shift for existing email, it engenders several new problems. First, there is a QoS problem, which results from the streaming delivery of continuous media across a bandwidth-limited network path. Second, there is the problem of deciding when to delete message data from the sender's outbox, which the recipient may wish to access at an unknown point in time. Third, sender-stored email introduces complexities into the process of forwarding and replying with annotation (embedding pieces of the original message in the reply).

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 continuous media 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.

Chapter 4

Audio Message Boards

4.1 Introduction

There are currently a number of different forms of asynchronous conferencing on the Internet. Mailing lists are one form of conferencing tool, where participants submit email to a central server, which periodically compiles these comments into a single message and distributes it to the participants. News groups are another from of asynchronous conferencing, where participants place messages within a message tree. Newsgroups now take two different forms. There are the traditional newsgroups, based on the Network News Transfer Protocol (NNTP)[NNTP], and Web-based newsgroups, which are now commonly found as a service in the context of a larger information architecture, such as a corporate website. The first Web-based newsgroup was HyperNews[LALI], a CGI-based system in the public domain. When the research comprising this thesis was begun in February 1998, Hypernews was the only available Web-based news group. By the time of this writing, three years later (March 2001), Web-based newsgroups (or message boards) have become a common Web service.

These asynchronous messaging systems have so far been based on text. In particular, audio has yet to emerge as a medium for asynchronous conferencing on the Internet. However, because computer systems with audio playback and capture capabilities are now becoming common, it is now feasible to add audio to asynchronous conferencing. To better understand the technical issues involved with such development, and begin experimentation with its use, we developed a prototype audio-based asynchronous conferencing system called *aconf*.

Adding audio to asynchronous messaging adds a new dimension of possibility for participants to express themselves more completely. Although

emotion can be conveyed in text, its expression may be more complete or easier to achieve when using one's voice. For example, an asynchronous classroom where students are reading and discussing poetry would benefit from the use of voice, because the reader's voice can convey emotions that might not otherwise be experienced by a reader. Voice messages also can provide a more personal feeling that is difficult to accomplish with only text. Imagine a private conference set up for use by a family or group of friends. In such an environment, listening to the familiar voices of family or friends may help to evoke a feeling of closeness, which may be harder to achieve with text-only messages. In addition to the emotional and personal dimensions, voice messaging is highly desirable in the context of an asynchronous classroom for the instruction of a foreign language. In this context, students can replay spoken examples to study the sound and rhythm of the language, and submit their own responses for correction by the instructor.

In addition to improving the expressiveness of messages, audio messaging systems can be more efficient than text-only systems, because message creation can be done more rapidly. However, although message creation is more efficient, message consumption may be less efficient, because people can read text more quickly than the rate at which it is spoken. Also, readers can skim message text in order to omit content they feel is unimportant. But these problems are resolvable, because (1) audio messages can be played back at a faster rate than they were recorded, and (2) special audio rendering techniques, such as those used in SpeechSkimmer[ARON], provide users with functions that let them *skim* audio in a manner analogous to how they skim text.

Another advantage to audio messaging is that many people consider reading text on the computer screen to be a source of eyestrain, especially when reading through a long document. While the graphical user interface of the monitor may be the appropriate place for the user to process multiple choices presented in a single instant, many users tire when reading sequentially presented information, such as a document. Clicking on hyper links to get to the information one desires is probably the most efficient means of navigating through an information space, but after one has located the information desired, it may be easier for the user to listen to a document rather than read it.

Finally, we chose not to consider video at this time for three reasons. First, the bandwidth required for transmitting good quality video data is beyond what many users have available to them through a modem connection, and so the playback delay may be too great. Second, the storage requirement for video data is large, and would therefore consume excessive disk space on the server. Third, although many multimedia-equipped computers

have microphones, few have video capture devices. Of course, these factors will undoubtedly disappear over time, making video messaging more attractive; but the aim of our experiment was to implement a system that would be accessible to as many users as possible and function reasonably within the present environment.

4.2 Design Objectives

The guiding principle that we followed when designing our prototype system was to make it free of any special software installation, including going to a Web site to download and install a plug-in. If the user has speakers and a microphone connected to his system, then it should be possible for the user to simply follow a link to a conference, and begin listening to messages and leaving messages of his own without having to go through an elaborate installation procedure.

We were able to reach this goal within reasonable limits. The first time the user goes to a conference, he is presented with a pop-up window asking if he wishes to accept an ActiveX control, with a valid electronic signature. If the user clicks OK, the ActiveX control is loaded and installed. The user is never again confronted with the same question; any conference he may go to, at any Web site, will be able to interact transparently with the installed control.

As a second goal, we wanted the system to be accessible to as many people as possible, that is, to be operational in any browser and on any operating system. Since Java is platform independent, we naturally considered developing the client side of the system with a Java applet. However, at the time we developed the prototype, Java did not provide developers with audio capture functionality. Such functionality is now available with Java version 1.3, but is not yet available through the Java API exposed by browsers to java applets. However, if users install the Java 1.3 plug-in, an applet can be used to accomplish audio capture and playback. But this approach defeats the simplicity of our nothing-to-install objective.

There is also no mechanism in HTML that provides for audio capture and delivery via an HTTP POST to the server, as there is for text data. There have been Internet drafts by J. Salsman[SALS] proposing an extension to the INPUT element of HTML that provides this functionality, however, browser developers have not yet embraced the idea. Another possible solution is for browsers to support an OBJECT element that displays audio capture controls, and functions within the FORM element for submission of its data to the server. This mechanism is described in the W3C HTML 4.0

specification[HTML]. It seems likely that such functionality will eventually be provided across all platforms either through the HTML FORM element or through the OBJECT element.

Because a relatively platform independent solution was not available (and is still not available), we had to settle with a platform-specific solution at the client. We chose to use an ActiveX control to provide the audio capture and data submission to the server, thus limiting conference access to users of Internet Explorer running on a Windows operating system.

4.3 The Client Web Page

The standard format of a conference is the message tree. Messages are either placed on the top level or under a parent message. Messages are represented by lines containing message title, author and other information such as date. These lines are then presented as a vertical list in which child messages are indented under their parents. An icon appears to the left of the message, which when clicked, causes the list of the messages' children either to appear or disappear beneath the message. This is the standard interface for viewing the contents of a hierarchically ordered collection of objects. We experimented with a different approach, which was to present a vertical list centered on a selected message. Above the selected message are all of its ancestors, and below are all of its children. The initial state is for the "top level" pseudo message to be selected, as shown in Fig. fig:alt1. When the user selects message 3, as illustrated in Fig. fig:alt2, then message 3 becomes the current message, it's siblings disappear, and its children appear.

We found the alternative interface too confusing, because messages change their positions within the display window when selected. However, we feel this approach still warrants some continued evaluation as a possible alternative to the expanding branches of a tree.

We thus remained with the conventional tree with expanding branches. When the user first arrives at a conference, he sees a list of top-level messages. Each message is presented as a checkbox, followed by the message title and the author's name. When the checkbox is selected, the children of the message are displayed indented under the message. When the title of a message is clicked, the message text appears in a textbox at the bottom of the screen and the system's media player retrieves and plays the audio component of the message (Fig. 4.3). The user controls the playback of the audio through the playback controls of the media player.

Under the last child message in each exposed list is an "add comment" link. The user clicks on this link to insert a new message at that point in the

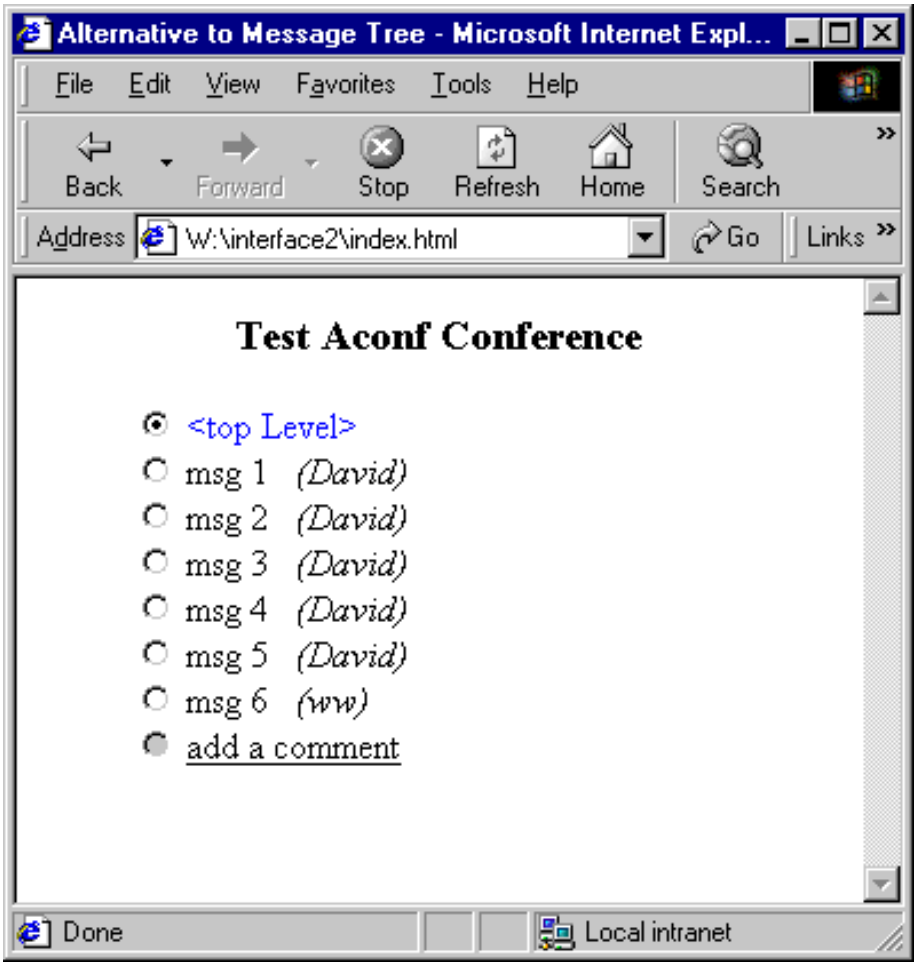

Figure 4.1: Vertical list interface, initial state

tree. After clicking on one of these add comment links, the message tree is replaced by a message creation interface (Fig. 4.4). This new screen includes textboxes for the users name, message title and message text. In addition to the text input boxes are audio capture controls in the form of buttons. These buttons are labeled RECORD, STOP, PLAY and ERASE. The user clicks RECORD to start audio capture, STOP to stop capture, PLAY to playback the contents of the capture buffer, and ERASE to erase the contents of the capture buffer. He can also click STOP to stop playback and RECORD to append onto data in the capture buffer. Once the user is satisfied with his message, he clicks the SEND button to submit the message to the conference server. If he decides not to submit his message, he clicks on the CANCEL button.

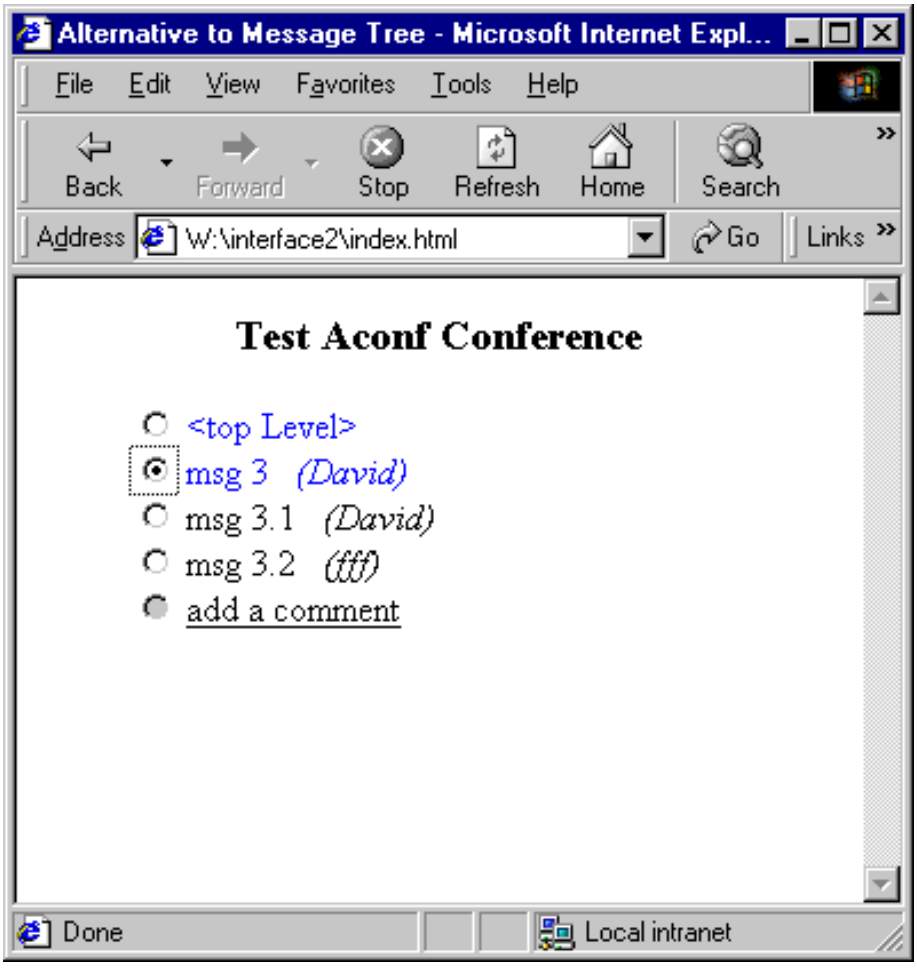

Figure 4.2: Vertical list interface, message 3 selected

The transition between the message tree of Fig. 4.3 and the message creation interface of Fig. 4.4 is done with dynamic HTML, which means that no interaction with the server is required. Although this technique provides for an extremely fast user interface, there is a problem regarding the browser's back button, which when clicked within the message creation window will not bring the user back to the message tree, but to the page from which the conference was originally entered. This behavior may not be understood by the user, because the two states of the unified Web page have the appearance of being two different Web pages. This problem exists for all such Web applications that utilize the document object model to provide a highly interactive Web page. The Web page designer may need to provide better clues than we do in our prototype so that the user perceives

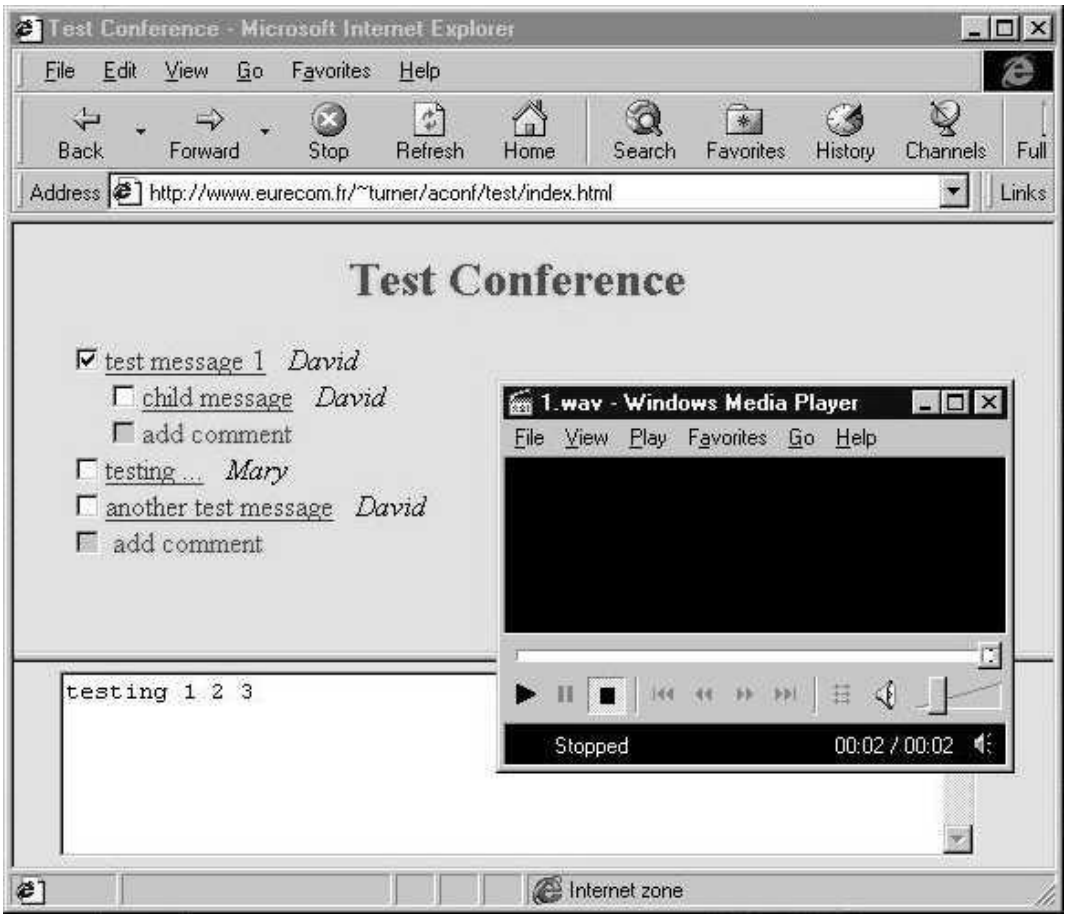

Figure 4.3: Message tree interface

the Web page as a single document, which will be exited when clicking the back button.

4.4 The Conference Server

System-level data is organized under a single directory; in our implementation we called the directory aconf. In this top-level directory we place all of the files which are common to all conferences, and subdirectories that contain files for specific conferences. The files in the top-level system directory include:

- aconf.cab — a signed cab file containing aconf.ocx, which is an ActiveX control.

Figure 4.4: Message creation interface

- AconfApplet.class — a Java applet whose purpose is to retrieve conference text data at Web page initialization.

- aconf.css — a cascading style sheet[HTTP] used to control the format of all conferences.

- aconf.js — a file of JavaScript functions that coordinate the separate system elements. The *onload* JavaScript function in this file interacts

with the Java applet to retrieve message text stored on the server, which it uses to build the message tree.

- TextBox.html — an HTML document that defines the scrollable text area to appear in the lower frame of the Web page.
- The conference subdirectories, which contain the following:
  - index.html — an HTML document that controls the frame layout of the Web page.
  - msgTree.html — an HTML document that defines the elements that appear in the upper frame of the Web page.
  - params.scr — a file containing conference-specific JavaScript variables.
  - data.txt — a text file that contains the text data for all messages in the conference.
  - 1.wav, 2.wav, 3.wav, etc. — the audio files named after their associated message numbers

The system requires a Web server, which may at any time be asked to deliver any one of the files identified in the previous list. In addition to the Web server, a conference server (written in Java) listens for TCP connection requests on some predetermined port number. This port number is configured in the JavaScript file Aconf.js, and passed to the activeX control during Web page initialization. The conference servers' job is to accept new messages sent from the client and add them to the public message tree. It does this in two steps. First, it assigns a message number to the new message and saves the audio data as a file with name equal to the message number. Second, it appends the textual data of the new message to the conference text data file, data.txt.

4.5 Web Page Initialization

The user enters a conference by loading the index.html file in the conference directory, which results in an initialization sequence that involves the loading of various supporting files, the installation of an ActiveX control, the running of a Java applet, and the execution of JavaScript that builds the message tree. Fig. 4.5 illustrates the components involved in initialization, and their functional relationships. When the browser loads the top-level HTML document, it retrieves the referencing documents, which include the upper frame, the

lower frame, the style sheet, the javascript file, the ActiveX control, and the Java applet. The browser builds the user interface, and runs the Javascript function referenced in the "onload" attribute in the BODY tag of the upper frame.

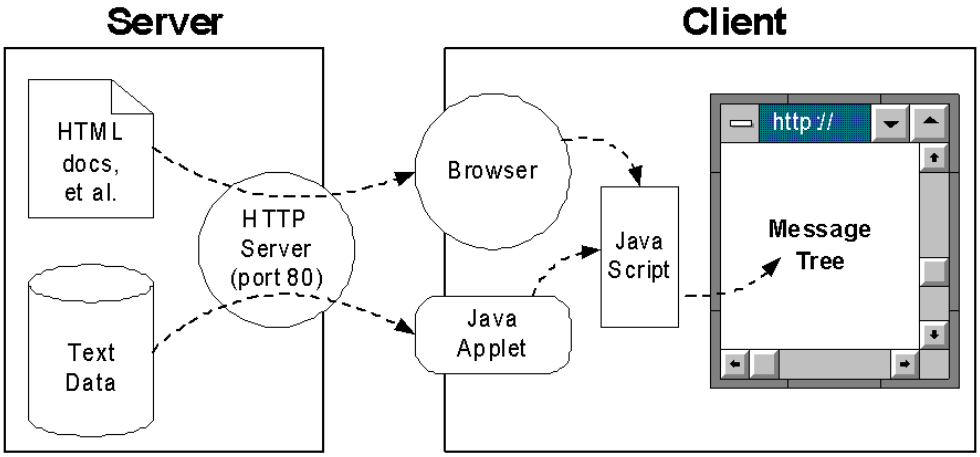

Figure 4.5: Web page initialization

The ActiveX control is referenced with the HTML <OBJECT> tag as follows:

```
<OBJECT id="Aconf" classid="CLSID:996F6602-7895-11d2-8F18-
    006008644547" codebase="http://www.eurecom.fr/~turner/aconf/
    aconf.cab#Version=1,0,0,4"></OBJECT>
```

The classid attribute is a 128-bit globally unique identifier (GUID) that Windows uses to identify components. When the browser encounters this element, it searches the system registry to check if a current version of this object is installed. If it either discovers that no such component with this GUID exists, or if it does exist but its version number doesn't match "1,0,0,4," then it will retrieve the cab file and install the control. When it does this,

the browser will display a window showing that the Web site is requesting the installation of an activeX control, and that the control has been signed by a valid certificate from "David A. Turner." If the user clicks the OK button to proceed, the user system extracts the file aconf.ocx from the cab file, and copies it into a subdirectory under the control of the browser. The file aconf.ocx is actually a Windows dynamic link library (DLL), and so the system has the control register itself by calling the exported function *DllRegisterServer()*.

DllRegisterServer() makes several calls to the Windows API to properly modify the Windows system registry to make the control accessible to JavaScript. One of the entries simply associates the 128-bit GUID with the pathname of the dll. After the control has registered itself, the browser will instantiate an instance of the control when it encounters a reference to its GUID in an OBJECT element. After instantiation, the javascript can call methods exposed by the controls interface through the id attribute specified in the OBJECT tag.

The contents of the CAB file is signed by a private key issued by Verisign, and a Verisign certificate is included with a public key that the user's browser uses to verify the authenticity of the code signer. When the user's browser loads the ActiveX control for the first time, the browser displays a security warning window in which the certificate is stated to be valid and signed by David A. Turner. The user is asked if he accepts the code to be installed on his machine. If he clicks on OK, the browser stores the ActiveX control in a subdirectory managed by the browser, and then calls the control's installation function, DllRegisterServer(). This function calls the Windows API to add the appropriate entries into the Windows registry. One of the entries associates the 128-bit GUID with the dll file stored in the browser's subdirectory of controls. Two other entries are also necessary in order to eliminate additional warning messages presented to the user: an entry that indicates the control is "safe for scripting," and another entry indicating it is "safe for initialization from persistent data."

If the user were to remove the control from his browser's environment, the system would call the DllUnregisterServer() function within the dll. This function removes the entries placed into the registry by the original call to DllRegisterServer(). Once the control has been installed, the OBJECT tag that references its GUID will result in the loading of the dll into memory, the initialization of the control contained in it, and Javascript will have access to the interface provided by the control.

The JavaScript initialization code is started by setting the onload attribute of the <BODY> tag, as in <BODY onload="buildMessageTree()" ...>. This function adds the HTML elements that comprise the message tree.

Before it starts adding elements, however, it first calls the loadData() method of the Java applet to retrieve the text data for the conference, which it does by issuing a GET request to the Web server. When this call returns, the script enters a loop in which it adds two <DIV> sections for each message in the message tree. The first pair of <DIV> sections for the first message in the conference will look as follows:

<DIV id="1" class="0" style="padding-left: 20; display:none">
<INPUT type="checkbox" onclick="expand('1')"> <A href="1.wav" onclick="play('1')"> test message 1 </A> <SPAN style='font-style:italic'> David </SPAN> </DIV>

<DIV id="r1" class="1" style="padding-left: 40; display: none">
<INPUT type="checkbox" disabled> <SPAN onclick="addComment('1')" style="cursor:hand; color:red"> add comment </SPAN> </DIV>

The second <DIV> section is used to provide a link that allows the user to add a comment at the bottom of the message's list of children. Each time a message needs to be added to the tree, a Javascript function uses the document object model (DOM) function insertAdjacentHTML() to insert two new <DIV> sections for the message just above the parent's ending <DIV> section.

4.6 User Interface

When the initialization script completes in the client browser, the client will have all of the text data, including message titles, author names and message text. However, the audio data will remain at the server. When the user wishes to listen to a message (and view its text content), he clicks the message title, which appears as a link. The message title link for message 1 of the test conference has the following form:

<A href="1.wav" onclick="showtext('1')"> test message 1 </A>

Clicking this link causes the browser to do two things. First, it executes the JavaScript function showtext(), which places the message text in the textbox in the lower frame. Second, it tells the system's media player to retrieve and render the wave file referenced in the href attribute. Fig. 4.3 shows one possible result of clicking on this <A> element.

At present, when retrieving audio files from a Web server, RealNetwork's G2 player will not render the audio file in streaming mode, that is, it will wait until it has retrieved the entire file before beginning playback. The

Microsoft Media Player introduces a fixed delay (settable by the user) before beginning playback of the audio file. If the audio playback rate is greater than the average bandwidth available over the TCP connection, and the length of the audio message is long enough, then the player will halt before the message has finished, while it rebuilds the playback buffer.

It is possible for these players to calculate an appropriate playback buffer size based on the average available bandwidth and file size. The file size is available from Web servers that include the Content-Length header in their response messages, which is the recommended behavior for HTTP 1.1 conformant Web servers (see §14.14 of [HTTP]). If the encoding rate is greater than the available bandwidth , then a relatively large amount of data needs to be buffered prior to commencing playback in order to avoid buffer starvation. On the other hand, if the playback rate is less than the available bandwidth, then the initial size of the playback buffer only needs to be large enough to overcome the delay that results from the TCP slow-start mechanism.

When the user wishes to expand a branch of the message tree (that is, look at the child messages of a particular message), he clicks the checkbox next to the message. Consider the first message of the test conference example of Fig. 4.3; the checkbox comes from the following HTML tag:

<INPUT type="checkbox" onclick="expand('1')>

When selected, the browser adds a check mark to the interior of the checkbox, and calls the JavaScript function expand() with the message number as its argument. The expand() function checks to see if the children of the message currently have the display attribute of their enclosing <DIV> tags set to "none," which makes them invisible, or "block," which makes them visible. If it finds that the children are invisible, it changes their display attributes to "block." If it finds that the children are visible, it changes their display attributes to "none" and also does the same for all descendents of the children. After performing this replacement operation, the browser re-renders the HTML document to reflect the modifications the script has made to its contents.

If the user wishes to add a message, he clicks on the "add comment" phrase that occupies the position in the tree he wishes to insert a new message. The add comment phrase for adding child messages to message number 1 is rendered from the following HTML:

<SPAN onclick="addComment('1')" style="cursor:hand; color:red">
add comment </SPAN>

When the user clicks on this text, the addComment() function is called, which is another one of the JavaScript functions contained in the script file msgtree.scr. This function sets the display attribute of the <DIV> containing the message tree to "none" and the display attribute of the <DIV> containing the message creation controls to "block." This transition from message tree display to message creation display (Fig. 4.3 to Fig. 4.4) is practically instantaneous, because the browser doesn't need to retrieve any data over the network.

At this point, the user sees text boxes for a name, message title and message text. Also, there are buttons for controlling the audio capture and message submission. Clicking any of these buttons results in the execution of a JavaScript function to perform the task, which it accomplishes by calling the associated methods on the ActiveX control. The process of mapping user generated Web page events to the ActiveX control is illustrated in Fig. 4.6.

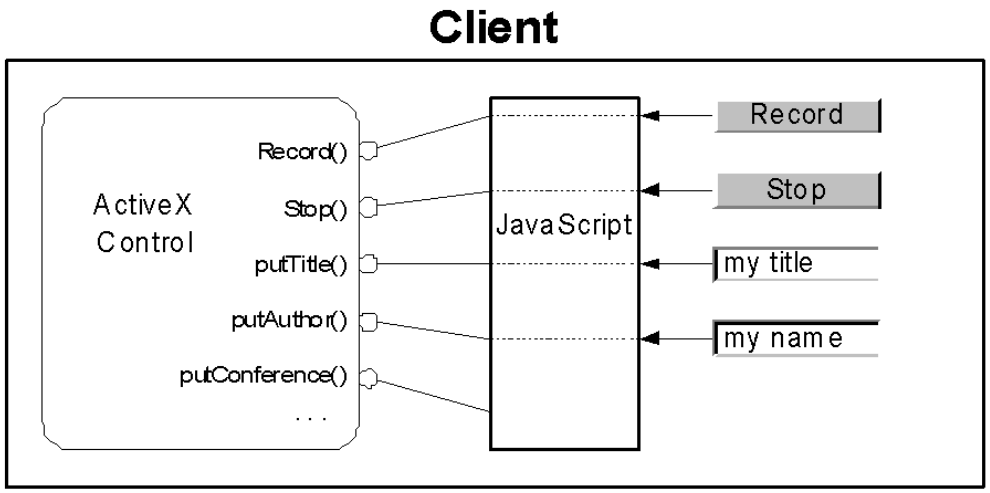

Figure 4.6: Message data and audio capture functions mapped to ActiveX control by JavaScript

After the user fills in the name and message title fields, and optionally the text field, he records a message. When he is satisfied with the recording,

he clicks on send, which passes execution to the JavaScript function send(). This function transfers the text data from the HTML text boxes to the ActiveX control, and then calls the send() method of the control. The control establishes a TCP connection with the conference server and submits the new message, which is sent as text data followed by the audio data. This process is illustrated in Fig. 4.7.

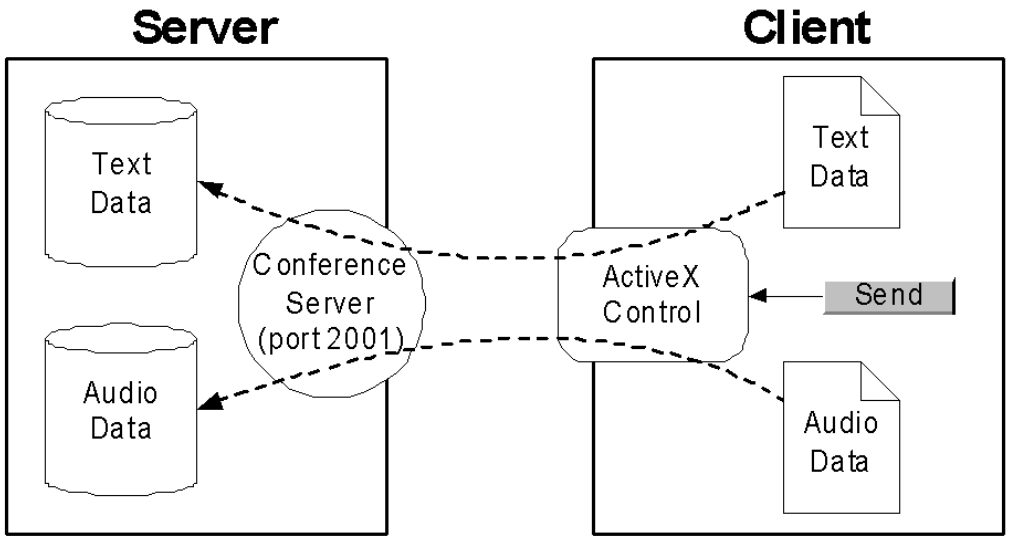

Figure 4.7: ActiveX control sends new message data to conference server

4.7 Summary

We have shown in our prototype that it is possible to construct a Web-based audio conferencing system that allows immediate user interaction without installation complexities. The user is presented only once with a window asking for permission to install an ActiveX control. Our approach, however, limits use to clients with Internet Explorer running under a Windows operating system.

Presently, it is possible to achieve platform and browser independence only by providing separate programming solutions for each targeted environment. However, when JavaSound becomes available, it will be possible to redesign the system so that it runs in all supporting browsers.

Another alternative is to extend the HTML <INPUT> element to include an audio value for its type attribute. When confronted with such an element, the browser would render an audio capture control to provide the user with a means of inputting audio. This is analogous to the way the browser renders a textbox control to allow the user to input text. A proposal for such an extension is currently in the form of an Internet draft, and thus may eventually be adopted as a standard and implemented. Such an extension would eliminate the need for the use of mobile code such as the ActiveX control used in our prototype, and would thus increase the security of user systems.

We have also demonstrated that dynamic HTML can be used to provide a complex Web page interface, which eliminates the delays associated with a CGI-style system that requires frequent retrieval of new HTML documents from the Web server.

Our prototype system relies on a standard Web server to deliver all of the objects to the client, and only implements a server component that performs a single function: to add new messages to a conference. This conference server component is platform independent, because it is written in Java.

Part II

Streaming Stored Multimedia

Chapter 5

Optimal Streaming of Layer-Encoded Presentations

5.1 Introduction

Many distributed multimedia systems deliver content across network paths with different levels of bandwidth. They may also be additionally constrained by a minimum start-up delay and limited client buffer. In order to meet these constraints within the context of available bandwidth, the content delivery system degrades the quality of the component elements of the presentation in order to reduce the size of their data representations. The motivation is to achieve on-time delivery, so that play-out does not stall because of lack of data.

Many of the proposed and implemented multimedia presentation streaming systems attempt to adapt their media flows in the context of changing bandwidth levels [CAND][REAL][RAMA]; however, there is mounting evidence that IP bandwidth is highly erratic, and properly modeled by probability distributions with infinite variance [PAX97a]. Under these conditions, even adaptive schemes breakdown at unpredictable times and for unpredictable durations. Although best-effort transmission works relatively well for asynchronous applications, such as email delivery and the propagation of news messages between news servers, users are more directly inconvenienced when media streams can not be delivered smoothly with short start-up delays. As a result, there has been growing interest in evolving the current best-effort Internet to one that is capable of providing QoS guarantees to media streams. In particular, there has been a lot of work done on Integrated Services (intserv) [RFC2212] and Differentiated Service (diffserv) [RFC2475]. It is likely that one or more of these technologies will provide end systems with

the ability to negotiate fixed bandwidth delivery channels for the duration of a multimedia presentation. We start by solving the problem of optimal scaling of multimedia data within the context of this expected environment. We then extend our fixed bandwidth solutions to the context of networks with unreliable QoS, which we do by embedding our solution within a two-phase decision process that responds to changing levels of bandwidth.

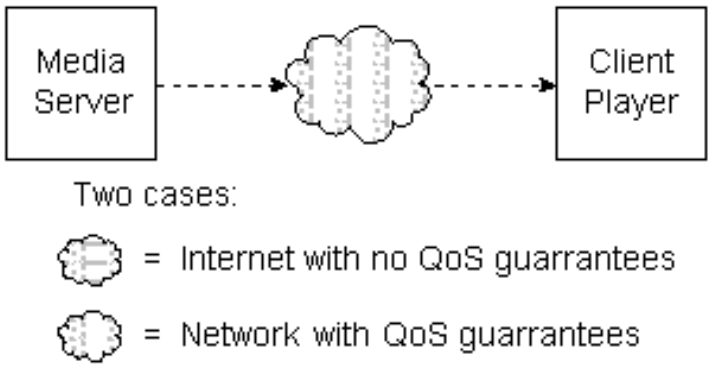

Figure 5.1: Network environments

When the network can not provide enough bandwidth to deliver the full resolution of the presentation, or if client memory is inadequate for storing pre-fetched data, the application must reduce the size of the data representation in order for the data to be delivered on time and rendered smoothly, and the end user must accept a presentation with a correspondingly degraded quality. To adapt to these constraints, multimedia data is sometimes structured with a layer-encoding scheme. The application adapts to the available bandwidth level by sending more or less layers, which results in more or less quality. For example, the current JPEG standard contains a progressive encoding option[JPEG]. By default, the libraries of the Independent JPEG Group encode color images into 10 layers, and black and white images into 6 layers. The progressive JPEG encoding scheme is widely supported by graphics libraries and popular applications such as Web browsers. Another example of scalable media is MPEG video, which is typically encoded into 2 or 3 layers.

Layer-encoded media is *progressive* in the sense that the data representation is ordered. An application can use data unit ℓ only if it combines it with units $1, \ldots, \ell - 1$. Consequently, a reliable transport protocol is needed, so that one lost packet at the beginning of the stream does not result in transporting useless data. We therefore assume a lossless transport channel, such as TCP.

All presentation data is considered to be representable in the form of

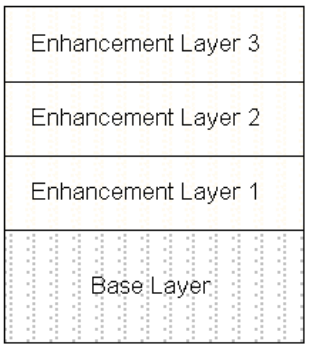

Figure 5.2: Layer encoded data

scalable discrete objects with associated rendering intervals. We take the start point of the rendering interval as the object's delivery deadline. For example, a static image composed of 2400K bits may have a rendering interval starting at $t = 10$ seconds, and ending at $t = 15$ seconds, resulting in a playout duration of 5 seconds, and a delivery deadline of $t = 10$ seconds. (All times in this thesis are relative to the start of presentation play out.)

We formulate the problem of scaling down the quality of a multimedia presentation as a constrained optimization. In the case of infinite client memory, we seek a transmission policy in the form of an N-dimensional vector that represents the number of layers to transmit of each unit of data, where the total number of data units is N. Of all possible transmission policies, we consider those that result in on-time delivery of the data units to be feasible. With a notion of overall presentation quality, we seek to find a feasible transmission policy that maximizes overall quality. We consider two different methods of comparing overall presentation quality: a total quality metric, and a refined max-min metric. In this context, we assume a generic object quality metric from which the overall presentation quality is determined. We demonstrate the superiority of the refined max-min metric for solving problems in this domain.

After solving the problem for limited bandwidth, we consider the problem of limited client memory. That is, non-rendered media data is not allowed to accumulate in client memory without bound. In other words, we consider a finite pre-fetch buffer. This problem is important in relation to small mobile devices or Internet appliances with limited memories. In this case, we need to determine not only the number of layers to send of each data unit, but also the rate at which we deliver these units to the client, so that what we decide to send does not overflow the client pre-fetch buffer.

In this chapter, we formulate the various problems in terms of discretely

scalable media. In the Chapter 6, we reconsider these problems under the assumption of continuously scalable media.

In the context of stored VBR-encoded video, several papers have studied the transmission of video from a server to a client with finite storage; a partial list includes [ZHAO99, FENG99, MCMA96, MCMA98, SALE96, SALE98, REIS97, REIS98]. All of these papers assume that the video is encoded with one layer. These papers have examined a variety of smoothing and pre-fetching schemes that minimize bandwidth usage for fixed start-up delay and finite client storage capacity.

Recently, Zhao, Willebeek-LeMair and Tiwari have considered "malleable multimedia presentations" [ZHAO98, ZHAO99]. Their model is similar to ours in that the objects are assumed to be encoded in layers. A central assumption in their work is that client storage is finite. They investigate the tradeoff between the size of client storage and the number of layers that can be transmitted, assuming that the transmission bandwidth is limited. In [ZHAO99], the authors assume all objects have the same number layers and only consider policies that transmit the same number of layers for each object. They propose a binary search algorithm to search through the number of layers in order to find the maximum number of layers that can be sent while satisfying the bandwidth constraint. In [ZHAO98] they provide an enhancement algorithm that adds one layer to a subset of the objects while remaining feasible.

Our work differs from [ZHAO98] and [ZHAO99] in many respects. First, we place the problem within the context of a two-phase decision problem in which start up delay is minimized, which is not done in [ZHAO98] and [ZHAO99]. Second, we allow for general quality values for each layer for each object. References [ZHAO98] and [ZHAO99] only consider the special case where quality equals number of rendered layers, which is in many circumstances not an accurate measure of quality. Third, we examine optimization criteria that are different from simply finding the maximum number of layers that can be transmitted for all objects. We consider the natural total quality criterion and show how optimizing total quality can be formulated as a dynamic programming problem. We also propose a refined max-min criterion, which makes efficient use of the available bandwidth while striving to maximize the worst-case quality for rendered objects. We present numerical testing with real progressive JPEG data to investigate and compare the different quality value definitions and optimization criteria. We also introduce progressive rendering of static objects into the optimization objective, and study how progressive rendering influences the optimal policy. Finally, we solve the problem of optimal streaming of continuously scalable media.

Table 5.1: Slide show elements

element	rendering interval
image 1	[0, 15]
image 2	[15, 30]
image 3	[30, 50]
audio	[0, 50]

5.2 Transmission Policies

To illustrate basic concepts and notation, we will refer to the sample slide shown in Table 5.2, which is comprised of four elements: three images and an audio stream. This slide show has a total rendering period of 50 seconds. The first image has a rendering interval of $[0, 15]$, which means it is displayed at the client at time $t = 0$ seconds, and continues to be rendered until time $t = 15$ seconds. The second image has rendering interval $[15, 30]$, and the third image, $[30, 50]$. There is an audio stream underlying the presentation, which extends from time $t = 0$ seconds to time $t = 50$ seconds.

Not all elements in the presentation can be treated the same. The data for the discrete media elements—namely the images—must arrive at the client before the start of their rendering intervals, because all of the data is needed at the start of their rendering intervals. However, the continuous media elements do not have the same restriction. For example, playout of the audio data can commence before all of the data arrives. We would be adding unnecessary delay to the playout of the presentation by requiring all the audio data to arrive at the client before time $t = 0$ seconds. To solve this problem, we divide the audio stream into many small contiguous rendering intervals as shown in Table 5.2, and we refer to units comprising the presentation's data representation with the term "object" rather than "element" after applying this transformation.

Now, all of the data units comprising the presentation are treated as discrete elements with delivery deadlines equal to the start of their rendering intervals. Table 5.2 illustrates our current view of the presentation's data.

Our final transformation is to sort the objects by their delivery deadlines and assign an index. The result of applying this transformation is illustrated in Table 5.2. We designate the delivery deadline of object i by t_i.

Note that the objects comprising the presentation can be a mixture of various media type, such as video frames, still images, audio segments, etc. Object i is composed of L_i layers, and the j^{th} layer of object i contains x_{ij} bits. Fig. 5.3 provides an example illustration of the notation developed so

Table 5.2: Slide show objects

object	rendering interval
image 1	[0, 15]
image 2	[15, 30]
image 3	[30, 50]
audio 1	[0, 1]
audio 2	[1, 2]
⋮	⋮
audio 50	[49, 50]

Table 5.3: Delivery deadlines

object	delivery deadline
image 1	0
image 2	15
image 3	30
audio 1	0
audio 2	1
⋮	⋮
audio 50	49

Table 5.4: Object numbering

object	delivery deadline	object index
image 1	0	1
audio 1	0	2
audio 2	1	3
⋮	⋮	⋮
audio 14	13	15
image 2	15	16
⋮	⋮	⋮
audio 50	49	53

far.

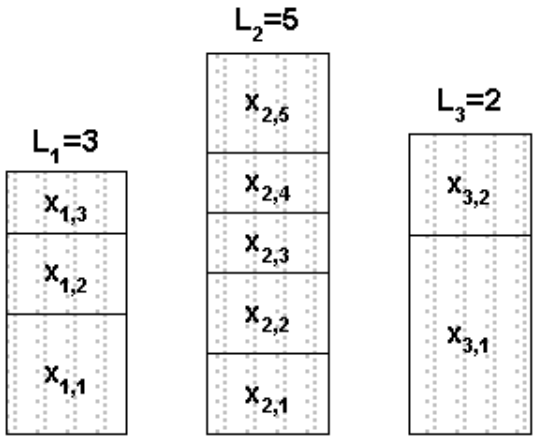

Figure 5.3: Layer-encoded multimedia

The presentation timeline (Fig. 5.4) starts at time $t = -d$, where d is the number of seconds the application spends pre-fetching data prior to starting playout. For notational convenience, we let $t_0 = -d$. Note that the deadline for the first object t_1 is illustrated as different than the start of presentation playout $t = 0$. This is shown this way for the sake of generality. Normally, $t_0 = 0$, because the start of playout would correspond to something being rendered.

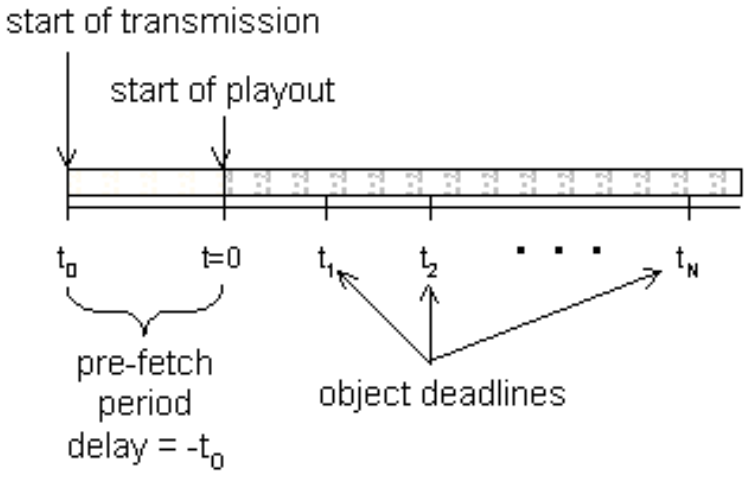

Figure 5.4: The presentation timeline

Initially, we make the assumption that bandwidth is known, and thus we can determine the bits that can be transferred in each interval on the presentation timeline defined by the object deadlines. Let b_i represent the number of bits that can be transferred on interval i, which is the interval with starting point t_{i-1} and ending point at t_i. Because the bandwidth may not be adequate to transport the full resolution of all the objects before their deadlines, the application needs to determine how many layers it can transport of each object so that all the bits delivered will arrive before their deadlines. We call the number of layers to send of each object the transmission policy, and represent it by an N-dimensional vector $(j_1, \ldots, j_N)$. We use y_i to represent the number of bits selected by the transmission policy for the i^{th} object. Fig. 5.5 provides a graphic illustration of these variables.

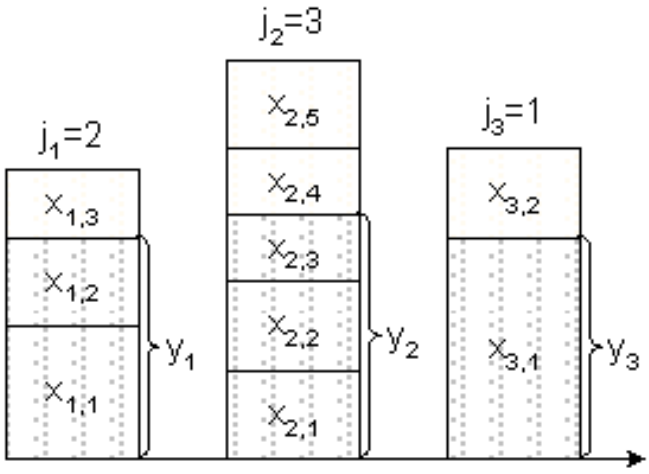

Figure 5.5: Transmission policy

We call a transmission policy $P = (j_1, \ldots, j_N)$ *feasible* if all the bits selected arrive at the client before their deadlines. Because each object has a hard deadline for the delivery of all of its bits, we can assume that the application will always transmit bits for objects with earlier deadlines before objects with later deadlines. This assumption of sequential delivery provides a way to determine if a particular transmission policy is feasible. Fig. 5.6 illustrates how this works. We must have that the total number of bits transmitted of object 1 be less than or equal to the total number of bits that can be transmitted over the first interval, that is, the interval between the start of transmission t_0 and the first deadline t_1. Since the bits of object 2 are sent immediately after the bits of object 1, we must have that the sum of the bits for objects 1 and 2 be less than or equal to the total number of bits that can be transmitted before the deadline of the second object t_2.

The continuation of this logic should be clear from Fig. 5.6, and the system of inequalities representing these transmission constraints are thus given in (5.1).

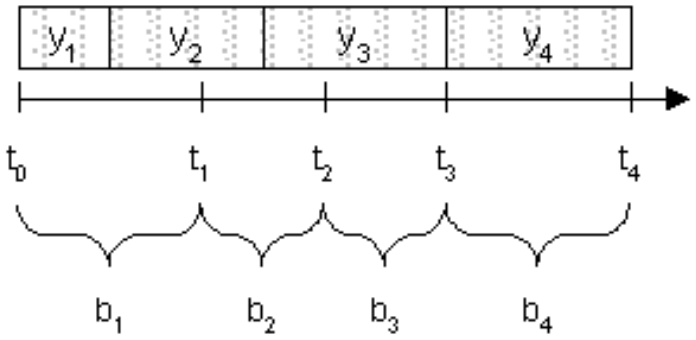

Figure 5.6: Transmission constraints

$$\begin{aligned} & y_1(j_1) \leq b_1 \\ & y_1(j_1) + y_2(j_2) \leq b_1 + b_2 \\ & \vdots \\ & y_1(j_1) + \ldots y_N(j_N) \leq b_1 + \ldots + b_N \end{aligned} \tag{5.1}$$

5.2.1 Presentation Quality

In order to determine an optimal transmission policy from the set of feasible policies, we need to have an objective function, which naturally is to maximize a measure of overall presentation quality. For this purpose, we assume that $Q(P)$ is a measure of overall presentation quality resulting from policy P, and we say that a transmission policy is *optimal* if it is feasible and there is no other feasible policy with greater overall quality.

To specify $Q(P)$, we need to have a measure of individual object quality. For this purpose, we assume a generic object quality metric $q_i(j)$ for each object, which is a function of the number of layers rendered. We assume that q is non-decreasing, and maps onto $[0,1]$, where $q_i(0) = 0$ and $q_i(L_i) = 1$. Our object quality metric will take on $L_i + 1$ different values for object i, since there are that many different renderings (including the trivial rendering of zero bits).

There are many possibilities for q. One possibility is to let q be the *resolution* of the rendered object, that is, the percentage of bits rendered from the object's total representation. We also refer to the resolution quality metric as the *bit-oriented* quality metric. Another possibility is to equate rendered quality with the percentage of layer's rendered, which we refer to as the *layer-oriented* quality metric. Another possibility is to equate quality with the root mean square of the difference between the pixels (or samples) of the image (or audio) rendered under the policy and the pixels (or samples) of the image (or audio) rendered with all possible layers. Still another possibility is that human testing is used as a basis of assigning effective quality scores to the L_i possible values the metric can take on for object i. In any case, the optimal algorithms we develop will work with any possible quality assignment, and so we can assume a general quality metric $q_i(j)$.

5.3 The Total Quality Criterion

In this section, we use dynamic programming to develop an efficient algorithm for the determination of the optimal transmission policy with respect to the *total quality* metric, which equates the overall quality of a presentation with the sum of the qualities of its component objects. That is,

$$Q(P) = \sum_{i=0}^{N} q_i(j_i)$$

A brute-force determination of $Q(P)$ requires $L_1 L_2 \ldots L_N$ calculations, which is not polynomial-bounded in the number of objects. However, there exists a recursive formulation of the problem that yields an algorithm that terminates in polynomial time. To do this, we first develop some terminology, which we use to state a theorem. Then we show how the theorem provides an efficient method of solving the optimization problem. We end the section with a proof of the theorem.

Suppose we only want to send objects i through N, and that we have s surplus bits of bandwidth available to do this, in addition to the $b_i, \ldots, b_N$ bits that are available in the intervals terminating at deadlines $t_i, \ldots, t_N$. We say the *sub-policy* $(j_i, \ldots, j_N)$ is feasible if it satisfies the following deadline constraints:

$$\begin{aligned} & y_i(j_i) \leq s + b_i \\ & y_i(j_i) + y_{i+1}(j_{i+1}) \leq s + b_i + b_{i+1} \\ & \vdots \\ & y_i(j_i) + \ldots + y_N(j_N) \leq s + b_i + \ldots + b_N \end{aligned} \tag{5.2}$$

Let $\Omega_i(s)$ be the set of all such feasible sub-policies, that is, all sub-policies that satisfy (5.2). We define $f(i,s)$ to be the maximum total quality attainable over the set of feasible sub-policies. That is,

$$f(i,s) = \max_{\Omega_i(s)} \Big(q_i(j_i) + \ldots + q_N(j_N) \Big). \tag{5.3}$$

By definition, $f(1,0)$ is the maximum quality for the presentation. We develop a recursive expression for f that will allow us to compute $f(s,N)$, $f(N-1,s)$, ..., $f(2,s)$, $f(1,0)$, in that order. For this purpose, define $\mathcal{X}_i(s)$ to be the set of all possible values for the number of layers of object i that can be sent within a bandwidth of s (surplus) bits plus the bits that can be transmitted between deadlines t_{i-1} and t_i. That is,

$$\mathcal{X}_i(s) = \Big\{ j \mid y_i(j) \leq s + b_i \Big\}. \tag{5.4}$$

Theorem 1 *$f(i,s)$ can be expressed recursively as follows:*

$$f(N,s) = \max_{j \in \mathcal{X}_N(s)} \Big(q_N(j) \Big), \tag{5.5}$$

and for $i = N-1, \ldots, 1$,

$$f(i,s) = \max_{j \in \mathcal{X}_i(s)} \Big(q_i(j) + f\Big(i+1, s + b_i - y_i(j) \Big) \Big). \tag{5.6}$$

This theorem provides a way to compute an optimal policy using the following procedure:

1. Use (5.5) to compute and store $f(N,s)$ for values of s that range from 0 to the total number of presentation bits (incrementing s by some reasonable value, such as 100).

2. Similarly, for varying values of s, use (5.6) to compute and store $f(N-1,s)$, $f(N-2,s)$,..., $f(2,s)$, in that order.

3. Compute $\widehat{j}_1$ that maximizes (5.6) for $f(1,0)$, that is, $\widehat{j}_1$ maximizes $q_1(j) + f(2, b_1 - y_1(j))$ over $\mathcal{X}_1(0)$. Set $\widehat{s}_1$ to the unused bandwidth to be carried into the next interval, that is, $\widehat{s}_1 = b_1 - y_1(\widehat{j}_1)$.

4. Repeat the following for $i = 2, \ldots, N-1$:

   - Compute the value $\widehat{j}_i$ that maximizes $q_i(j) + f(i+1, \widehat{s}_{i-1} + b_i - y_i(j))$ over $\mathcal{X}_i(\widehat{s}_{i-1})$.

- Let $\widehat{s}_i$ represent the unused bandwidth to carry into the next interval, that is, $\widehat{s}_i = \widehat{s}_{i-1} + b_i - y_i(\widehat{j}_i)$.

5. Compute the value $\widehat{j}_N$ that maximizes $q_N(j)$ over $\mathcal{X}_N(\widehat{s}_{N-1})$.

The optimal transmission policy will be $(\widehat{j}_1, \ldots, \widehat{j}_N)$. This algorithm has complexity $O(LNS)$, where L is an upper bound for the number of layers in each object, N is the number of objects, and S is the number of presentation bits.

Proof Let $f(s,i)$ be defined as in (5.3), and let $g(s,i)$ be defined as in (5.5) and (5.6). We use an inductive argument to show that f and g are identical. For the case of $i = N$, note that since $\Omega_N(s) = \Big\{(j) \mid y_N(j) \leq s + b_N\Big\}$ and $\mathcal{X}_i(s) = \Big\{j \mid y_N(j) \leq s + b_N\Big\}$, j ranges through the same values in the maximizations of $q_N(j)$ over $\Omega_N(s)$ in (5.3) and $\mathcal{X}_N(s)$ in (5.5). Thus,

$$f(N,s) = \max_{\Omega_N(s)} \Big(q_N(j_N)\Big) = \max_{\mathcal{X}_N(s)} \Big(q_N(j)\Big) = g(N,s).$$

This establishes equality for the case $i = N$.

Now, we make the inductive assumption that $g(i+1,s) = f(i+1,s)$. We demonstrate that $g(i,s) = f(i,s)$ by first showing $g(i,s) \geq f(i,s)$, and then $g(i,s) \leq f(i,s)$. By definition of g as satisfying (5.6), we have

$$g(i,s) = \max_{j \in \mathcal{X}_i(s)} \Big(q_i(j) + g\Big(i+1, s + b_i - y_i(j)\Big)\Big).$$

Combining this with our inductive assumption, we have

$$g(i,s) = \max_{j \in \mathcal{X}_i(s)} \Big(q_i(j) + f\Big(i+1, s + b_i - y_i(j)\Big)\Big),$$

and then by definition of f in (5.3),

$$g(i,s) = \max_{\mathcal{X}_i(s)} \Big(q_i(j) + \max_{\Omega_{i+1}(s+b_i-y_i(j))} \Big(q_{i+1}(j_{i+1}) + \ldots + q_N(j_N)\Big)\Big) \tag{5.7}$$

Suppose that $(j_i(s), \ldots, j_N(s))$ maximizes $q_i(j_i) + \ldots + q_N(j_N)$ over $\Omega_i(s)$. Then by definition of f in (5.3), we have

$$f(i,s) = q_i\Big(j_i(s)\Big) + \ldots + q_N\Big(j_N(s)\Big). \tag{5.8}$$

Additionally, the first inequality of (5.2), which defines $\Omega_i(s)$, gives

$$y_i\Big(j_i(s)\Big) \leq s + b_i.$$

Thus, $\mathcal{X}_i(s)$, as defined in (5.4), contains $j_i(s)$. This means the outer maximization in (5.7) covers $j_i(s)$. Thus we have:

$$g(i,s) \geq q_i\Big(j_i(s)\Big) + \max_{\Omega_{i+1}(s+b_i-y_i(j_i(s)))} \Big(q_{i+1}(j_{i+1}) + \ldots + q_N(j_N)\Big) \qquad (5.9)$$

Substituting $j_i(s), \ldots, j_N(s)$ into the other inequalities in (5.2) (omitting the first inequality) produces the following system:

$$\begin{aligned}
&y_i\Big(j_i(s)\Big) + y_{i+1}\Big(j_{i+1}(s)\Big) \leq s + b_i + b_{i+1}\\
&y_i\Big(j_i(s)\Big) + y_{i+1}\Big(j_{i+1}(s)\Big) + y_{i+2}\Big(j_{i+2}(s)\Big) \leq s + b_i + b_{i+1} + b_{i+2}\\
&\vdots\\
&y_i\Big(j_i(s)\Big) + y_{i+1}\Big(j_{i+1}(s)\Big) + \ldots + y_N\Big(j_N(s)\Big) \leq s + b_i + b_{i+1} + \ldots + b_N
\end{aligned}$$

Shifting the first term $y_i\Big(j_i(s)\Big)$ onto the right side of these inequalities gives us the system:

$$\begin{aligned}
&y_{i+1}\Big(j_{i+1}(s)\Big) \leq s + b_i - y_i\Big(j_i(s)\Big) + b_{i+1}\\
&y_{i+1}\Big(j_{i+1}(s)\Big) + y_{i+2}\Big(j_{i+2}(s)\Big) \leq s + b_i - y_i\Big(j_i(s)\Big) + b_{i+1} + b_{i+2}\\
&\vdots\\
&y_{i+1}\Big(j_{i+1}(s)\Big) + \ldots + y_N\Big(j_N(s)\Big) \leq s + b_i - y_i\Big(j_i(s)\Big) + b_{i+1} + b_N
\end{aligned}$$

By definition of Ω, this system implies that

$$\Big(j_{i+1}(s) \ \ldots, j_N(s)\Big) \in \Omega_{i+1}\Big(s + b_i - y_i\Big(j_i(s)\Big)\Big).$$

Therefore, the maximization in (5.9) covers the point $\Big(j_{i+1}(s), \ldots, j_N(s)\Big)$, and so we have from (5.9) and (5.8) that

$$g(i,s) \geq q_{i+1}\Big(j_{i+1}(s)\Big) + \ldots + q_N\Big(j_N(s)\Big) = f(i,s).$$

Now we complete the proof by showing $g(i,s) \leq f(i,s)$. For this purpose, suppose that $j_i(s)$ is an element in $\mathcal{X}_i(s)$ that maximizes the outer maximization in (5.7). Then we can rewrite g as follows:

$$g(i,s) = q_i\Big(j_i(s)\Big) + \max_{\Omega_{i+1}(s+b_i-y_i(j_i(s)))} \Big(q_{i+1}(j_{i+1}) + \ldots + q_N(j_N)\Big)\Big). \qquad (5.10)$$

Since $j_i(s) \in \mathcal{X}_i(s)$, we have

$$y_i\Big(j_i(s)\Big) \leq s + b_i. \tag{5.11}$$

Now suppose that $\Big(j_{i+1}(s), \ldots, j_N(s)\Big)$ solves the maximization in (5.10), so that

$$g(i,s) = q_i\Big(j_i(s)\Big) + \ldots + q_N\Big(j_N(s)\Big), \tag{5.12}$$

and

$$\Big(j_{i+1}(s), \ldots, j_N(s)\Big) \in \Omega_{i+1}\Big(s + b_i - y_i\Big(j_i(s)\Big)\Big). \tag{5.13}$$

Statement (5.13) implies the following system of inequalities:

$$\begin{aligned}
&y_{i+1}(j_{i+1}(s)) \leq s + b_i - y_i(j_i(s)) + b_{i+1} \\
&y_{i+1}(j_{i+1}(s)) + y_{i+2}(j_{i+2}(s)) \leq s + b_i - y_i(j_i(s)) + b_{i+1} + b_{i+2} \\
&\vdots \\
&y_{i+1}(j_{i+1}(s)) + \ldots + y_N(j_N(s)) \leq s + b_i - y_i(j_i(s)) + b_{i+1} + \ldots + b_N
\end{aligned} \tag{5.14}$$

If we shift the term $-y_i(j_i(s))$ appearing in each inequality to the left side, we see that (5.11) and (5.14) imply that $(j_i(s), \ldots, j_N(s))$ satisfies (5.2), and thus is a member of $\Omega_i(s)$. Therefore, $(j_i(s), \ldots, j_N(s))$ is covered by the maximization in (5.3) that defines f, and so we have

$$f(i,s) \geq q_i\Big(j_i(s)\Big) + \ldots + q_N\Big(j_N(s)\Big).$$

This result combined with (5.12) gives us $f(i,s) \geq g(i,s)$. ■

We will use the algorithm developed in this section to compute, in Sec. 5.5, optimal transmission policies for two different object quality functions at varying levels of bandwidth. We will compare these policies with those generated from a different algorithm that we develop in the next section based on a different presentation quality metric, called the max-min quality metric. Both of these algorithms generate optimal policies under their respective criteria, and both can be used for scheduling the delivery of presentation layers before and after play out has begun.

5.4 The Refined Max-Min Criterion

One possible measure of overall presentation quality $Q(P)$ is simply to take the worst object quality across all objects comprising the presentation. A feasible policy would then be optimal if it maximizes the minimum quality

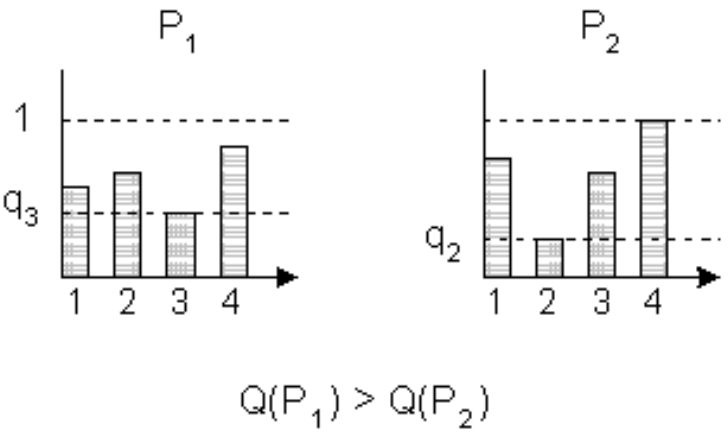

Figure 5.7: Max-min criterion

across all objects in the presentation. We call this the *max-min criterion.* (See Fig. 5.7).

Although the max-min criterion is a natural choice for a criterion of optimality, it typically provides policies that do not allocate all the available bandwidth. Consequently, sending additional layers for some objects improves the overall quality of the presentation, although the minimum quality remains fixed. In general, less cumulative bandwidth is available for transmitting objects with earlier deadlines, thus the minimum attainable quality will be dominated by bandwidth available for the early objects. This result will be especially problematic when the start-up is short. We now consider a *refined max-min criterion* that overcomes these inadequacies.

With the refined max-min criterion, we represent the overall quality of the delivered presentation by a vector of object qualities with components that are sorted in increasing levels of quality.

Definition 4 (Refined Max-Min Quality Vector) *Let $P = (j_1, \ldots, j_N)$ be a transmission policy. If a_i is the object with the i^{th} worst quality under P, then we call $Q(P) = (q_{a_1}(j_{a_1}), \ldots, q_{a_N}(j_{a_N}))$ the refined max-min quality vector of P.*

For example, suppose we have a presentation with three objects, a transmission policy of $P = (2, 4, 3)$, and quality values $q_1(2) = 16.3$, $q_2(4) = 30.4$, and $q_3(3) = 20.5$. In this case, we have that $a_1 = 1, a_2 = 3, a_3 = 2$, and $Q(P) = (16.3, 20.5, 30.4)$. Note that if there are quality ties, then there is more than one assignment of objects to $a_1, \ldots, a_N$. However, these various assignments will result in the same quality vector, and thus the refined max-min quality vector remains well defined in the case of quality ties.

The transformation of transmission policies to refined max-min quality vectors is a mapping from a domain of N-dimensional integer-valued vectors to a range of N-dimensional real-valued vectors whose components are non-decreasing in their index: $n < m \Rightarrow q_n(j_n) \leq q_m(j_m)$. We define a total ordering over this range set, so that we can distinguish between policies with greater or lesser quality.

Definition 5 (Ordering of Sorted Quality Vectors) *Let* $Q(P) = (q_1, \ldots, q_N)$ *and* $Q(P') = (q'_1, \ldots, q'_N)$. *We say* $Q(P) > Q(P')$ *if and and only if there exists an integer* $k \in \{1, \ldots, N\}$ *such that* $q_i = q'_i$ *for* $i = 1, \ldots, k-1$, *and* $q_k > q'_k$.

Fig. 5.8 provides a graphical illustration of how Definition 5 is used to compare two 4-dimensional policies P_1 and P_2.

Definition 6 (Optimal Refined Max-Min Policy) *We say that a feasible transmission policy is optimal if there exists no other feasible policy with greater quality.*

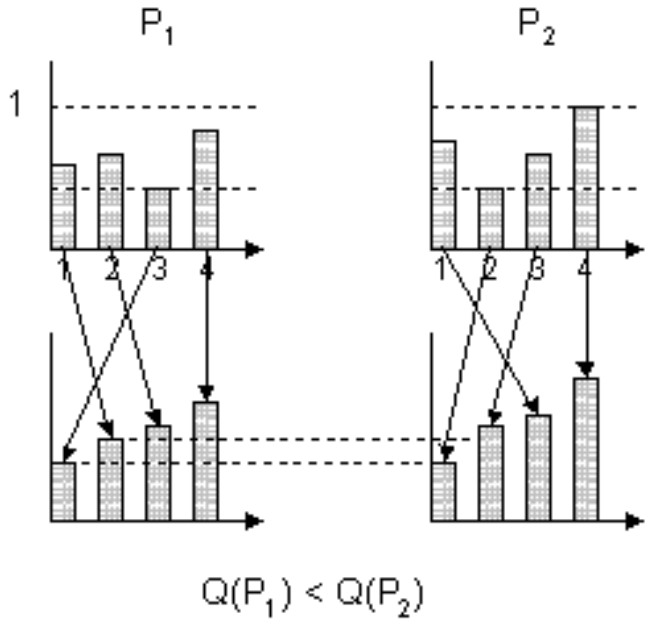

Figure 5.8: Refined max-min criterion

It is important to note that if a policy is optimal for the refined max-min criterion, then it is also optimal for the max-min criterion. However,

the converse is not generally true. Thus, the refined max-min criterion is a more sensible measure for the overall quality of a presentation, because in addition to satisfying the max-min criterion, it better exploits the available bandwidth to improve the quality of the presentation.

We now present an algorithm that determines the optimal policy under the assumption that the quality values $q_i(j)$ are distinct for all values of i and j. Quality measures that utilize the length of time rendered, root mean square, or number of bits, typically fulfill this assumption. The algorithm (Fig. 5.9) initializes a policy vector $(j_1, \ldots, j_N)$ to zeros, and increases in stages the number the layers to be sent for each object. The algorithm enters a loop in which it tries to add a layer to the object with lowest quality. If adding a layer to this object results in a feasible policy, then the layer counter for this object is incremented. On the other hand, if adding a layer results in a non-feasible policy, then its layers are held fixed, and no longer considered (by removal from S) for a possible quality improvement. Thus, the algorithm has the form of a basic greedy algorithm.

```
for i ← 1, ..., N
        j_i = 0
S = {1, ..., N}
while S is not empty
        find k ∈ S s.t. q_k(j_k) ≤ q_i(j_i) for all i in S
        if (j_1, ..., j_k + 1, ..., j_N) is feasible then
                j_k ← j_k + 1
        else
                remove k from S
```

Figure 5.9: Refined max-min algorithm

Fig. 5.10 is a graphical illustration of how the algorithm works. The upper left-hand graph within the illustration shows the state of the algorithm at some arbitrary point in the execution of its outer loop. There are four objects in the set S, each with a resulting quality that is represented by the height of the bar relative to the Y-axis. The first step from this point is for the algorithm to select the fourth object for a quality improvement, since it has lowest quality. After increasing the number of layers of the fourth object by one, the algorithm determines that the resulting policy is feasible, i.e. it checks to see that the inequalities in equations (5.1) are satisfied. In the second step, the algorithm selects the second object for a quality improvement, because it has the least quality of objects in S. However, incrementing the

number of layers for the second object results in a transmission policy that is not feasible. In response, the algorithm goes to step 3 in which it removes the second object from S, leaving it's layer count fixed for the duration of the algorithm. In step 4, the algorithm selects object 3 from S and increases its layer count.

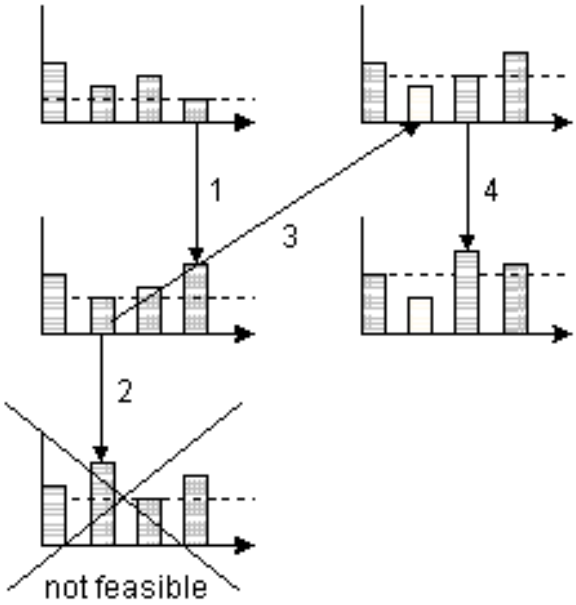

Figure 5.10: Greedy refined max-min algorithm illustrated

We now proceed to show that the algorithm in Fig. 5.9 indeed produces an optimal policy for the refined max-min criterion. To do this, we first establish the following lemma.

Lemma 1 *When the algorithm removes an object k from S, we have that $q_i(j_i - 1) < q_k(j_k)$ for $i \in S$.*

Proof Suppose we are at the point in time when the algorithm removes k from S. Relative to this point, let i be an arbitrary element in S, and let j'_k and j'_i be the number of layers assigned to objects k and i, respectively. We demonstrate the lemma by showing:

$$q_i(j'_i - 1) < q_k(j'_k). \tag{5.15}$$

Case 1 $j'_i = 1$.

It is easy to see that (5.15) holds by observing that $q_i(j'_i - 1) = q_i(0) = 0$, and $q_k(j'_k) \geq q_k(1) > 0$.

Case 2 $j_i' > 1$.

Consider the point prior to the removal of k from S when the algorithm last incremented the layer counter for object i, that is, when j_i was equal to $j_i'-1$. Relative to this point, let j_k'' be the number of layers assigned to object k. Because the layer counter for object i was being incremented, the quality of object i was less than the quality of all objects in S, and in particular, the quality of object k. Thus,

$$q_i(j_i' - 1) < q_k(j_k''). \tag{5.16}$$

Since this time precedes the point when k is removed from S, we must have that $j_k'' \leq j_k'$, which implies, since q is increasing, that

$$q_k(j_k'') \leq q_k(j_k'). \tag{5.17}$$

Taken together, (5.16) and (5.17) demonstrate (5.15). ■

Theorem 2 *The refined max-min algorithm converges to an optimal policy for increasing quality metrics that take on distinct values over the set of presentation objects.*

Proof Let $\widehat{P} = (\widehat{j}_1, \ldots, \widehat{j}_N)$ be an optimal policy, that is, a feasible policy whose quality is greater than or equal to all other feasible policies. Let $\widehat{a}_i$ represent the object with the i^{th} best quality under $\widehat{P}$, which means that the i^{th} component of $Q(\widehat{P})$ is $q_{\widehat{a}_i}(\widehat{j}_{\widehat{a}_i})$. At an arbitrary point during the execution of the algorithm, let $P = (j_1, \ldots, j_N)$ represent the current policy, and a_i the object with the i^{th} best quality under P. Thus, the i^{th} component of $Q(P)$ is $q_{a_i}(j_{a_i})$. Each time an object is removed from S, its assigned layers (and thus its quality) becomes fixed. At the point when the m^{th} object is removed from S, the following inequalities will hold for the duration of the algorithm:

$$q_{a_1}(j_{a_1}) < q_{a_2}(j_{a_2}) < \ldots < q_{a_i}(j_{a_m})$$

Additionally, since the objects remaining in S have qualities higher than $q_{a_m}(j_{a_m})$, and since the number of layers assigned to them are never decreased, we must have that the following holds true after the m^{th} removal from S:

$$q_{a_m}(j_{a_m}) < q_{a_k}(j_{a_k}) \text{ for } k = m+1, \ldots, N$$

We prove the theorem inductively. In step 1, we demonstrate that when the algorithm removes its first object from S, we have

$$q_{a_1}(j_{a_1}) = q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1}).$$

In step 2, we make the inductive assumption that after m removals from S,

$$q_{a_i}(j_{a_i}) = q_{\widehat{a}_i}(\widehat{j}_{\widehat{a}_i}) \text{ for } i = 1, \ldots, m, \tag{5.18}$$

and then show that when the algorithm removes its next object from S, the following must hold:

$$q_{a_{m+1}}(j_{a_{m+1}}) = q_{\widehat{a}_{m+1}}(\widehat{j}_{\widehat{a}_{m+1}}).$$

Thus, when the algorithm removes the N^{th} object from S and terminates, we have that $Q(P) = Q(\widehat{P})$.

Before beginning step 1, note that since the quality values are assumed to be unique across all layers and objects within the presentation, (5.18) is equivalent to the following:

$$a_i = \widehat{a}_i \text{ and } j_{a_i} = \widehat{j}_{\widehat{a}_i} \text{ for } i = 1, \ldots, m \tag{5.19}$$

Therefore, we have as a corollary to the theorem that the optimal refined max-min policy is unique.

Step 1 *When the algorithm removes its first object from S, we have that $q_{a_1}(j_{a_1}) = q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$.*

Suppose the algorithm is removing its first object from S. As we noted before, this object will be permanently assigned to a_1, and it will have lowest quality in all remaining policies computed by the algorithm. We demonstrate equality by showing that both $q_{a_1}(j_{a_1}) > q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$ and $q_{a_1}(j_{a_1}) < q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$ lead to contradictions.

Assume that $q_{a_1}(j_{a_1}) > q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$. Because a_1 has been removed from S, all other objects have qualities greater than $q_{a_1}(j_{a_1})$ under P, which means that $q_{a_i}(j_{a_i}) > q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$ for $i = 1, \ldots, N$. Thus, $Q(P) > Q(\widehat{P})$, which contradicts the fact that $\widehat{P}$ is optimal.

Now assume $q_{a_1}(j_{a_1}) < q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$. Since $q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$ is less than all other values of $q_{\widehat{a}_i}(\widehat{j}_{\widehat{a}_i})$, we have

$$q_{a_1}(j_{a_1}) < q_{\widehat{a}_i}(\widehat{j}_{\widehat{a}_i}) \text{ for } i = 1, \ldots, N.$$

Since $\{\widehat{a}_1, \ldots, \widehat{a}_N\} = \{1, \ldots, N\}$, the above is equivalent to

$$q_{a_1}(j_{a_1}) < q_i(\widehat{j}_i) \text{ for } i = 1, \ldots, N. \tag{5.20}$$

Since $a_1 \in \{1, \ldots, N\}$, (5.20) gives $q_{a_1}(j_{a_1}) < q_{a_1}(\widehat{j}_{a_1})$. Because q is increasing, we must have that $j_{a_1} < \widehat{j}_{a_1}$, and then, since j_i takes only integer values,

$$j_{a_1} + 1 \leq \widehat{j}_{a_1}. \tag{5.21}$$

Combining our assumption that $q_{a_1}(j_{a_1}) < q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$ with the lemma, we have

$$q_i(j_i - 1) < q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1}) \text{ for } i \in S.$$

Since $q_{\widehat{a}_1}(\widehat{j}_{\widehat{a}_1})$ is less than all other values of $q_{\widehat{a}_i}(\widehat{j}_{\widehat{a}_i})$, we can state that $q_i(j_i - 1) < q_i(\widehat{j}_i)$, which leads to $j_i - 1 < \widehat{j}_i$, and then

$$j_i \leq \widehat{j}_i \text{ for } i \in S. \tag{5.22}$$

Taken together, (5.21) and (5.22) imply that $(j_1, \ldots, j_{a_1}+1, \ldots, j_N)$ is component-by-component less than $\widehat{P}$, and therefore must be feasible (because the terms on the left-hand side of the feasibility are positive). But this contradicts the fact that a_1 is being removed from S.

Step 2 *If the first m removals from S result in (5.18), then the next removal from S results in $q_{a_{m+1}}(j_{a_{m+1}}) = q_{\widehat{a}_{m+1}}(\widehat{j}_{\widehat{a}_{m+1}})$.*

Similar to step 1, we show that $q_{a_{m+1}}(j_{a_{m+1}}) > q_{\widehat{j}_{\widehat{a}_{m+1}}}(\widehat{a}_{m+1})$ and $q_{a_{m+1}}(j_{a_{m+1}}) < q_{\widehat{a}_{m+1}}(\widehat{j}_{\widehat{a}_{m+1}})$ lead to contradictions.

Assume that $q_{a_{m+1}}(j_{a_{m+1}}) > q_{\widehat{j}_{\widehat{a}_{m+1}}}(\widehat{a}_{m+1})$. Since the objects remaining in S have qualities greater than $q_{a_{m+1}}(j_{a_{m+1}})$, we have that $q_{a_i}(j_{a_i}) > q_{\widehat{a}_{m+1}}(\widehat{j}_{\widehat{a}_{m+1}})$ for $i = m+1, \ldots, N$. Thus, $Q(P) > Q(\widehat{P})$, which contradicts the assumption that $\widehat{P}$ is optimal.

Now, assume that $q_{a_{m+1}}(j_{a_{m+1}}) < q_{\widehat{a}_{m+1}}(\widehat{j}_{\widehat{a}_{m+1}})$. Because $q_{\widehat{a}_k}(\widehat{j}_{\widehat{a}_k})$ is increasing in k, we have that

$$q_{a_{m+1}}(j_{a_{m+1}}) < q_i(\widehat{j}_i) \text{ for } i \in \{\widehat{a}_{m+1}, \ldots, \widehat{a}_N\} \tag{5.23}$$

Because our inductive assumption is equivalent to (5.19), we have that $\{a_1, \ldots, a_m\} = \{\widehat{a}_1, \ldots, \widehat{a}_m\}$, and then,

$$\{a_{m+1}, \ldots, a_N\} = \{\widehat{a}_{m+1}, \ldots, \widehat{a}_N\}. \tag{5.24}$$

Thus, 5.23 holds for $i \in \{a_{m+1}, \ldots, a_N\}$, and in particular, for $i = a_{m+1}$. Therefore, we have that $q_{a_{m+1}}(j_{a_{m+1}}) < q_{a_{m+1}}(\widehat{j}_{a_{m+1}})$. Because q is strictly increasing, we have $j_{a_{m+1}} < \widehat{j}_{a_{m+1}}$, and then

$$j_{a_{m+1}} + 1 \leq \widehat{j}_{a_{m+1}}. \tag{5.25}$$

The lemma guarantees that $q_i(j_i - 1) < q_{a_{m+1}}(j_{a_{m+1}})$ for $i \in S = \{a_{m+2}, \ldots, a_N\}$. But this is also true for $i = a_{m+1}$, because q is increasing. Combining this with our assumption that $q_{a_{m+1}}(j_{a_{m+1}}) < q_{\widehat{a}_{m+1}}(\widehat{j}_{\widehat{a}_{m+1}})$, we get

$$q_i(j_i - 1) < q_{\widehat{a}_{m+1}}(\widehat{j}_{\widehat{a}_{m+1}}) \text{ for } i \in \{a_{m+1}, \ldots, a_N\}. \tag{5.26}$$

Because $q_{\widehat{a}_k}(\widehat{j}_{\widehat{a}_k})$ is increasing in k, and because of (5.24) we have that

$$q_i(j_i - 1) < q_{\widehat{a}_k}(\widehat{j}_k) \text{ for } i \in \{a_{m+1}, \ldots, a_N\} \text{ and } k \in \{\widehat{a}_{m+1}, \ldots, \widehat{a}_N\}$$

But since $\{a_{m+1}, \ldots, a_N\}$ and $\{\widehat{a}_{m+1}, \ldots, \widehat{a}_N\}$ are identical, we have that $q_i(j_i - 1) < q_i(\widehat{j}_i)$ for $i \in \{a_{m+1}, \ldots, a_N\}$, and thus $j_i - 1 < \widehat{j}_i$, for $i \in \{a_{m+1}, \ldots, a_N\}$. This implies

$$j_i \leq \widehat{j}_i \text{ for } i \in \{a_{m+2}, \ldots, a_N\}. \tag{5.27}$$

Equations (5.19), (5.25) and (5.27) taken together imply that $(j_1, \ldots, j_{a_{m+1}} + 1, \ldots, j_N)$ is component-by-component less than or equal to $\widehat{P}$, and therefore must be feasible. But this contradicts the fact that a_{m+1} is being removed from S. ■

For object quality measures that map into a relatively small range, such as $q_i(j) = j$, the refined max-min algorithm may not converge to an optimal policy, because it does not properly resolve quality ties. We propose (and evaluate in Sec. 5.5) the following tie-breaking heuristic: in the presence of a quality tie, choose the object whose next layer has the smallest number of bits. Intuitively, this heuristic makes sense, because we are improving the quality by one layer with the least expenditure of bandwidth.

In the next section, we use a sample presentation to compute the optimal policies under the refined max-min criterion for two different object quality functions, and for varying levels of bandwidth. We also do this for the total quality criterion discussed in Sec. 5.3, and use the results to compare the two approaches. Then in Sec. 5.7 we look at a method to extend the refined max-min approach to better utilize available bandwidth by permitting progressive rendering of static objects.

5.5 Experimental Results

In this section we report some of our experimental results regarding the difference between the policies generated by our total quality and refined max-min algorithms. We will see how well the refined max-min algorithm performs under the tie-breaking heuristic, and we will compare the policies generated by the to algorithms using different individual object quality functions. We are not reporting on the two-phase decision problem.

We assembled a slide show presentation with 2 black and white and 8 color JPEG images in order to compare the refined max-min criterion with the total quality criterion under two different object quality measures. We

Table 5.5: Image data for slide show presentation

image	deadline	bytes by layer number										total
	(secs)	1	2	3	4	5	6	7	8	9	10	
1	0	1321	1956	457	306	1926	2899	246	492	415	5210	15228
2	18	2966	8026	7479	14341	566	25803					59181
3	36	11118	14782	1594	2635	14471	28445	2556	3060	3830	73974	156465
4	51	11223	22622	12931	35286	2106	66897					151065
5	62	10536	17380	1007	1600	14285	30474	2550	3521	4839	81178	167370
6	76	3473	3666	602	569	1700	6880	826	978	922	19360	38976
7	95	4596	7312	852	629	2375	7988	952	1616	1645	14893	42858
8	107	9253	7322	818	1200	3068	16640	2587	2025	2482	68278	113673
9	124	4424	4969	1554	1153	2284	6597	829	1613	1473	14705	39601
10	133	13221	28030	2438	4026	45614	58405	2621	5035	8037	116863	284290

encoded the images using the Independent JPEG Group's library [JPEG], which by default encodes color images into 10 layers and black and white images into 6 layers. The two object quality measures were a layer-oriented measure, which equates quality with the ratio of layers rendered to the total number of layers in the object, and a bit-oriented quality measure, which equates quality with the ratio of bits rendered to total bits from all layers. Note that under the bit-oriented measure, convergence to an optimal refined max-min policy is guaranteed. We chose rendering times that reflect a quick-paced slide-show presentation. Table 5.5 shows the number of bytes in each layer of each image, and the deadlines for the arrival of each image relative to the presentation timeline.

The first object quality measure is the ratio of layers rendered to the total number of layers in the object, i.e., $q_i = j/L_i$. We call this the *layer-oriented quality measure*. We use this rather than the number of layers, because the objects in the presentation are not encoded into the same number of layers. The layer-oriented quality measure generally results in many ties, because many objects in the presentation will be encoded with the same number of layers. Thus, the policy that the refined max-min algorithm produces may be sub-optimal.

The second object quality measure is the ratio of bits rendered to total bits across all layers. We call this the bit-oriented quality measure. For the bit-oriented quality measure, we have:

$$q_i(j) = \frac{\sum_{k=1}^{j} x_{ij}}{x_i} \quad \text{, where} \quad x_i = \sum_{j=1}^{L_i} x_{ij}$$

This is a somewhat natural measure in that the number of encoded bits, in a loose sense, represents the "information" in the layers; we are therefore associating quality with rendered information.

We suppose that there is approximately a constant 24 Kbps of bandwidth

available for the presentation, and we fix the start up delay at 5 seconds. Fig. 5.11 shows the percentage of layers sent for each of the ten objects in the presentation for the refined max-min and total quality criteria under the layer-oriented quality measure. The resulting policies of the two algorithms appear to agree in general regarding which images should be weak (in terms of percentage of layers rendered) and which should be strong. However, the refined max-min algorithm produces a presentation with more uniform image qualities.

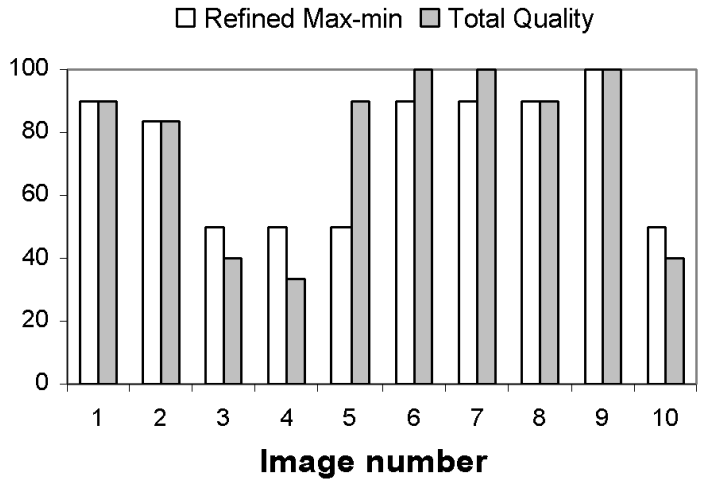

Figure 5.11: Percentage of layers sent by criterion under the layer-oriented quality measure

We can also see from Fig. 5.11 that the worst case object quality is 50 percent for the refined max-min criterion. The ordinary max-min criterion would have stopped at this point, generating a policy that transmits 50% of the layers for each object. The refined max-min criterion enables the presentation to display more layers than an ordinary max-min criterion while still respecting the max-min philosophy.

Fig. 5.12 shows the percentage of bits sent for each object under using the bit-oriented quality measure. Here the two algorithms give strikingly different policies. While the refined max-min algorithm continues to distribute relatively equal importance across all objects, the total quality algorithm selects a highly non-uniform distribution. The worst case object quality for the refined max-min criterion is approximately 30 percent of the object's bits, while that of the total quality criterion is less than 10 percent.

In order to further examine the differences between the two criteria and the two quality measures, we computed the optimal policies for the sample

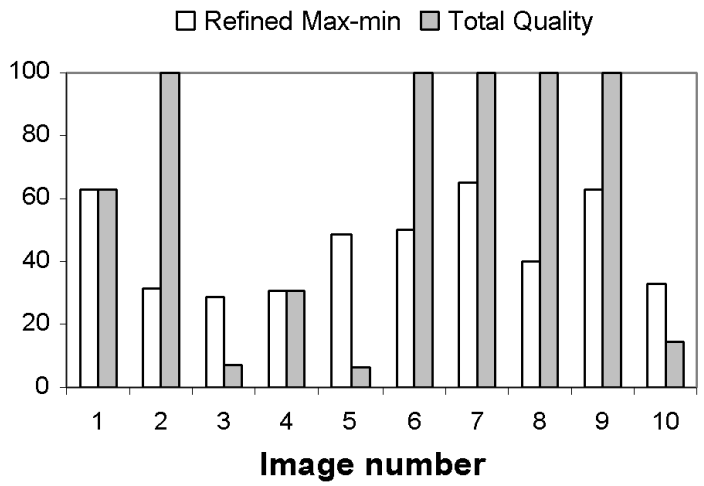

Figure 5.12: Percentage of layers sent by criterion under the bit-oriented quality measure

presentation while varying the level of bandwidth. We plotted three different summary statistics related to presentation quality: minimum percentage of layers rendered by bandwidth (Fig. 5.13), bandwidth utilization (Fig. 5.14), and average percentage of layers rendered by bandwidth (Fig. 5.15). Note that the legend appearing in Fig. 5.13 applies to all three figures.

Fig. 5.13 demonstrates that the refined max-min criterion is superior to the total quality criterion with respect to minimizing the worst quality, which isn't surprising, because this objective is its primary motivation. It should also be noted that the layer-oriented quality measure performs better than the bit-oriented quality measure. There are a few points where the total quality criterion with the layer-oriented measure performs better than the refined max-min criterion with the bit-oriented measure, but in general the refined max-min criterion performs better with both quality measures.

After maximizing the minimum object quality, our second motivation was to improve the presentation quality by using as much of the additional bandwidth as possible, while trying to minimize the worst case of those images that could be helped. Fig. 5.14 shows the bandwidth utilization (bandwidth consumed $\div$ available bandwidth) for the two criteria with the two quality measures. Here, all methods show increased variance in the upper bandwidth region, which is explained by the large amount of data concentrated in the final layer of each image. The plot shows that the refined max-min criterion works slightly better with the bit-oriented measure.

Fig. 5.15 shows the average percentage of layers rendered by bandwidth

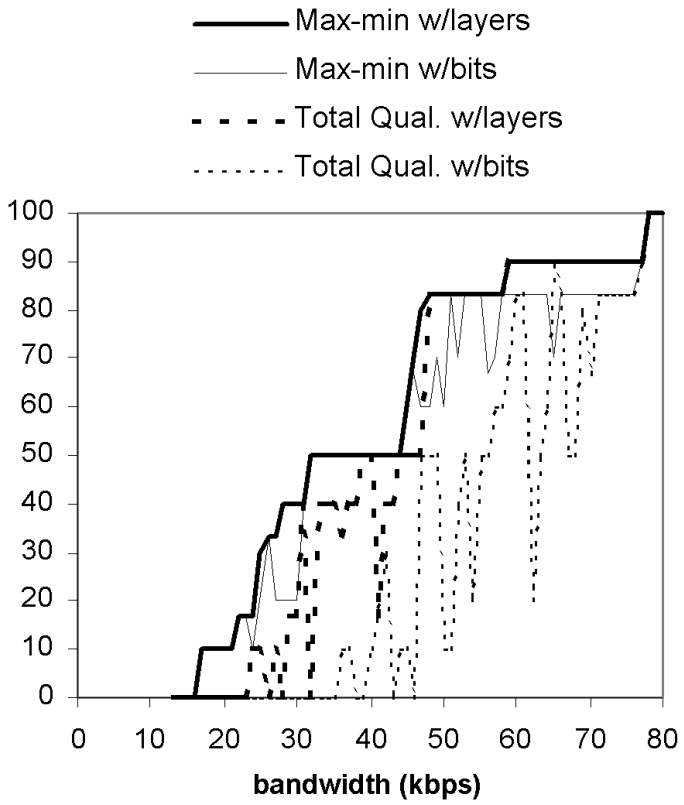

Figure 5.13: Minimum percentage of layers rendered by bandwidth

levels. Here the total quality criterion based on the layer-oriented quality measure is superior to the other methods, especially for the low- and mid-range bandwidths. The total quality criterion with the bit-oriented quality measure also appears to do well in the lower bandwidths, but gives weaker results for higher levels of bandwidth. At high levels of bandwidth, the various methods converge, but the refined max-min criterion based on the layer-oriented quality measure converges the most quickly. The refined max-min criterion with the bit-oriented quality measure is the clear loser in this comparison.

In summary, the four objectives can produce rather different optimal policies. We believe that the refined max-min criterion is superior to the total quality and max-min criteria. Nevertheless, in order to make a more definite conclusion, subjective testing with human subjects is needed.

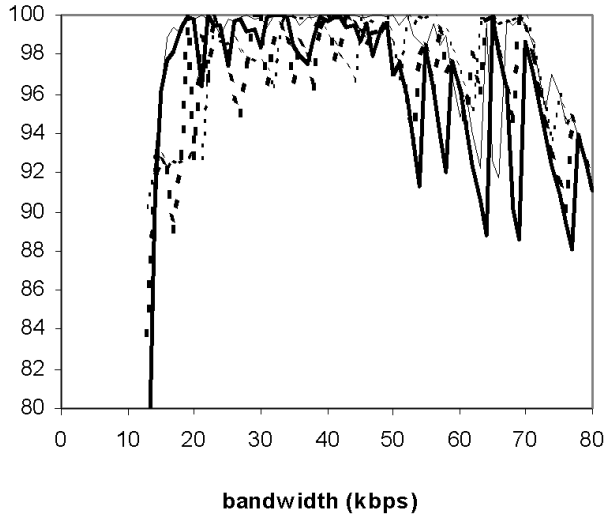

Figure 5.14: Bandwidth utilization by criterion/quality measure

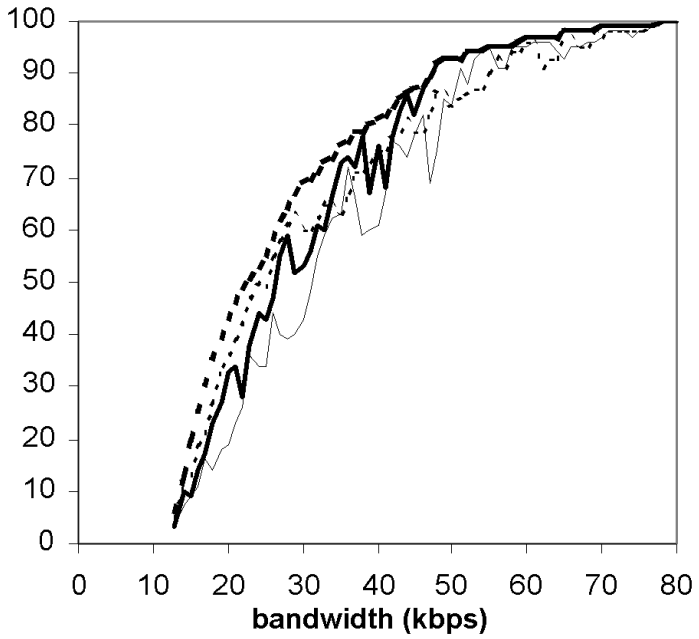

Figure 5.15: Average percentage of layers rendered by bandwidth

5.6 Adaptation to Changing Bandwidth

In this section we consider the problem of streaming a layer-encoded multimedia presentation from a server to an arbitrary client over a communication

network with no QoS guarantees, such as the global IP network. Studies of TCP bandwidth on the Internet [PAX97a] demonstrate the non-stationarity of TCP throughput. For this reason, the application must continuously monitor packet loss and delay characteristics, and use this in adjusting its expectation of future bandwidth. We assume the application has such an estimator, which will necessarily be related to the transport protocol used, whether it be TCP or an application specific protocol implemented over UDP.

The advantage of layer-encoding multimedia presentation data is that some layers can be dropped to reduce the size of the data representation, which will increase the speed at which the presentation can be transported over the network. With faster transmission, the end user is saved from waiting for presentation data to be pre-fetched into a playback buffer. However, fewer layers means lower quality, so we would like to send as many layers as possible while keeping the start up latency to a minimum.

Examples of layer-encoded multimedia data include progressive JPEG and GIF, and MPEG 1 and 2. The literature on layer-encoded video tends to use the terms *base layer* and *enhancement layers* to describe the order of layering. (See Fig. 5.2.) The intent is to define a minimum quality presentation, which is composed of only the base layers. If there is enough bandwidth to transport enhancement layers, then the application does so, and the resulting rendering at the client is correspondingly improved.

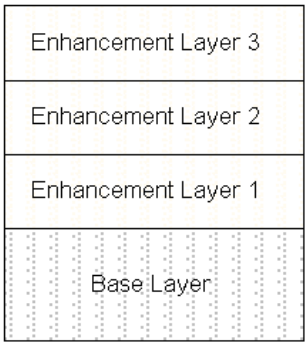

Figure 5.16: Base and enhancement layers

We assume that a presentation comprised of only the first layer of each object represents a presentation of minimum acceptable quality, and we use the term *base layer* to refer to the first layer, and *enhancement layer* to refer to higher layers, which contribute to quality, but are not required for achieving the minimum quality level.

Our first priority is to minimize the start up delay for playout of a presentation comprised only of base layers. After establishing the minimum start

up delay, our second priority is to improve the quality of the presentation by sending enhancement layers. Corresponding to these two priorities, we have two decision phases through which the application passes. In the first phase, the application simply sends base layers for objects in the order of their first appearance in the presentation. While it is sending base layer data, it collects information about the available bandwidth and uses it to decide when to start playing the presentation. After the presentation starts playing, the application enters into the second phase in which it loops on the decision of which layer to send next.

Because we are transmitting data over a network with no QoS guarantees, after a communication channel is established, the application does not know the rate at which it can transmit data from source to destination. For this reason, the application begins by transmitting only base layer data, which it stores in a pre-fetch buffer at the client until it determines that it can safely begin playout of the presentation without the threat of the presentation stalling.

While the application is transmitting base layers, it performs two other tasks in parallel. First, it records a packet transmission history, which it uses for predicting future bandwidth. Second, it loops on the decision whether to begin playing the presentation.

We do not assume any particular transport protocol, such as TCP or a particular UDP-based scheme. However, for whatever transport mechanism is employed, we assume the application has a reasonable method of estimating from the packet transmission history a lower bound for expected future bandwidth. We let $B(a, b)$ represent a lower bound for the number of bits the application expects can be delivered from server to client in the time interval $[a, b]$.

We maintain the notation developed in preceding sections. For the reader's convenience, we repeat these definitions in Fig. 5.17.

i	index used to label the presentation objects
N	the number of objects in the presentation
L_i	the total number of layers for object i
j_i	layers of object i to be transmitted (and rendered)
$B(a, b)$	estimated lower bound on bandwidth on $[a, b]$
t_i	deadline for the arrival of layers of object i

Figure 5.17: Notation

Suppose the application has delivered the base layers of objects 1 through $q-1$ to the client, and is now considering whether to start playout. Let $x_{i,j}$

be the number of bits in the j^{th} layer of object i. If the application starts play out now, all of the unsent base layers will arrive on-time if the cumulative bits needed at each deadline is less than or equal to the cumulative bits that can be delivered. Thus, the application starts play out if the following inequalities are satisfied:

$$\begin{aligned} & x_{q,1} \leq B(0, t_q) \\ & x_{q,1} + x_{q+1,1} \leq B(0, t_{q+1}) \\ & \vdots \\ & x_{q,1} + \ldots + x_{N,1} \leq B(0, t_N) \end{aligned}$$

If this system of inequalities is not satisfied, the application does not begin play out, but continues to pre-fetch additional base layers.

After the application gives the command to start playback, it transitions into phase two, in which it loops on the decision of which layer to send next. To make this decision, the application continues to record transmission statistics and refines its estimate of a lower bound on future bandwidth. With this estimate, it determines a sequence of layers P (the transmission policy) that can be delivered on-time (feasible) and that maximizes an objective measure of presentation quality $Q(P)$. The first layer in the sequence comprising this policy is chosen to be transmitted next.

Suppose that the rendering of the first r objects have already started, so the application only needs to concern itself with scheduling the delivery of layers from objects $r+1$ to N. To simplify notation, we re-label these objects as 1 through M.

A transmission policy specifies the number of layers to send for each object, which we represent as an M-dimensional vector P whose i^{th} component j_i represents the number of layers of object i to send to the client. We call a policy *feasible* if all of the bits it sends arrive at the client prior to their deadlines.

Let $b_i = B(t_{i-1}, t_i)$, the application's estimate of the number of bits that can be transmitted between the deadlines t_{i-1} and t_i. At each deadline, the cumulative number of bits needed must be less than or equal to the cumulative bits that will be transmitted. Thus, if we let $y_i(j)$ equal the unsent bits of object i that appear in layers 1 through j, then policy P is feasible if the following system of M inequalities hold.

$$\begin{aligned} & y_1(j_1) \leq b_1 \\ & y_1(j_1) + y_2(j_2) \leq b_1 + b_2 \\ & \vdots \\ & y_1(j_1) + \ldots y_M(j_M) \leq b_1 + \ldots + b_M \end{aligned}$$

Now that we have defined the set of feasible policies, we need to determine which of these policies are optimal, that is, which policies result in presentations with the best quality. The application now applies either the total quality criterion or the refined max-min criterion to find an optimal policy. It then sends the next unsent layer of the optimal policy.

5.7 Progressive Rendering

Now we consider the benefits and methodology of progressive rendering of static object data, such as JPEG images. In progressive rendering, the object layers are permitted to arrive after the beginning of the object's rendering period, but before the end of the rendering period. In this way, the client can improve the object's quality by rendering additional layers that arrive late.

To see that a transmission policy that includes progressively rendered objects has value, consider a slide show presentation of one minute duration that includes three images. The first image is displayed immediately and is rendered for the entire 60-second length of the presentation. The second image is rendered 20 seconds later and is rendered for the remaining 40 seconds. The third image is displayed 40 seconds into the presentation and is rendered for the remaining 20 seconds. Suppose the images are encoded into 6 layers.

Because the first image is to be displayed at the start of the presentation, the length of time needed for the transport of its base layer will determine the presentation start up delay. If the images are of equal size, and the bandwidth in 10 seconds is adequate to send all 6 layers of an image, then play out of the presentation will be $(1, 6, 6)$, that is, 1 layer of image 1, 6 layers of image 2, and 6 layers of image 3.

However, if we allowed an image to be progressively rendered in increasing degrees of quality, we could play out the presentation $(6, 6, 6)$. In this scenario, image 1 is rendered with 1 layer at time $t = 0$, and then is rendered with increasing quality as additional layers arrive during the next 10 seconds. At time $t = 10$ the transmission of layers of image 2 begins. All layers of images 2 and 3 arrive before the starting points of their rendering intervals, and thus are rendered with full quality. Clearly, this presentation has better overall quality than the one without progressive rendering. In general, presentations with a static object whose rendering begins at the beginning of the presentation will only have its base layer rendered.

Besides rendering more layers of static objects near the beginning of the presentation, another benefit of progressive rendering is to increase band-

width utilization near the end of the presentation. After the last object is delivered, it is possible to improve the quality of static objects that remain displayed by sending additional enhancement layers.

To see that progressive rendering is applicable for objects other than the first and the last, consider a presentation with three objects, each having 10 layers with 10K bits per layer. The presentation starts with a period of silence in which 30K of bandwidth is available. Then, object 1 is rendered for a length of time in which 30.01K of bandwidth is available. At the end of this period, the rendering of object 1 ceases and the rendering of object 2 begins. Then, there is 200K of bandwidth available during the rendering period of object 2. When the rendering period of object 2 ends, the rendering period of object 3 begins. (In our example, it is not necessary to specify the length of the rendering period of object three.) Without progressive rendering, the optimal refined max-min policy will be to send 3 layers of the first object (consuming the 30K of bandwidth in the first interval), 3 layers of the second object (consuming the 30K of bandwidth in the second interval), and 10 layers of the third object (consuming 100K of bandwidth in the third interval). In summary, the optimal refined max-min policy without progressive rendering will be (3, 3, 10).

On the other hand, if we permit progressive rendering, it is possible to render the policy (6, 10, 10). With this policy, the server transmits 26 layers back-to-back with the available bandwidth. First, it transmits the six layers of object 1, then the 10 layers of object 2, and finally the 10 layers of object 3. The first 3 layers of the first object are rendered at the beginning of its rendering interval, followed by 3 additional layers that are progressively rendered throughout the interval. When the rendering interval of the second object begins, no layers will be available, but each of the object's 10 layers will be transmitted and progressively rendered during the initial one third of the interval. The remaining bandwidth will then be used to transmit 10 layers of the third object, which will be rendered on time.

One drawback of policy (6, 10, 10) over (3, 3, 10) is that now the first three layers of the second object will arrive late relative to the start of their rendering interval. Thus it could be argued that this delay makes the progressive rendering policy inferior. In this case, one could use the policy (3, 10, 10), in which the first 3 layers of the second object are available at the start of its rendering interval, as in policy (3, 3, 10), and the 7 additional layers are progressively rendered.

However, a more serious objection to policy (6, 10, 10) is that layer 6 of the first object is used for an insignificant amount of time, but results in a significant delay in the rendering of the second object. Intuitively, policy (5, 10, 10) is better than (6, 10, 10), because it avoids sending a layer which

is rendered for an insignificant amount of time and reduces the rendering delay for the layers of the second object. The algorithm we present does not recognize this trade-off, and converges to policy (6, 10, 10). However, if we modify the definition of feasibility so that it rejects layers that arrive excessively late — such as 1 second prior to the end of their rendering interval — then the algorithm will converge to policy (5, 10, 10) in this example.

For progressive rendering, we now consider a policy P to be feasible if the bits sent arrive prior to the end, rather than the start, of their rendering intervals. For illustrative purposes, we consider the following natural definition of quality for progressive rendering:

$$q_i(j_1, \ldots, j_i) = \frac{1}{L_i} \sum_{j=1}^{L_i} \frac{v_{ij}}{w_i}, \text{ where}$$

v_{ij} is the rendering time of layer j of object i, and w_i is the length of the rendering interval for object i. The calculation of v_{ij} depends on the number of layers chosen for object i and the objects preceding object i, which is why we express q_i as a function of these layers.

The refined max-min algorithm *without* progressive rendering converges, because adding a layer to the object with the worst quality will not degrade the quality of the other objects. But this may not be the case with our new definitions of feasibility and quality, because at some point within the execution of the algorithm, it may be possible to add a layer to the object with minimum quality that results in the delayed arrival of succeeding layers, thus degrading their qualities. We propose the heuristic of testing for degradation in overall quality before adding a layer. The algorithm for progressive rendering with our proposed heuristic is shown in Fig. 5.18.

We used the same presentation data in Table 5.5 in Sec. 5.5 to compare the layer-oriented refined max-min algorithm with and without progressive rendering. Fig. 5.19 shows the percentage of layers rendered for each object under the two methods. Note that the policy with progressive rendering is better than the policy without progressive rendering in the sense of the refined max-min criterion. Interestingly, the progressive rendering algorithm decreases the number of layers in images 1 and 2 in order to send more layers of image 5. It also decreases the number of layers in images 6 through 9 in order to send more layers of image 10. The reason for this is that the algorithm delivers some of the layers in images 5 and 10 late, which then contribute less than a full unit towards the quality of their images, so the algorithm works harder to send additional layers of these two objects. Image

do for $i = 1, \ldots, M$
    j_i = number of layers sent of object i
    if $j_i == 0$ then $j_i = 1$
$S = \{1, \ldots, M\}$
do while S is not empty
    find $k \in S$ s.t. $q_k(j_k) \leq q_i(j_i)$ for all i in S
    $P = (j_1, \ldots, j_M)$
    $P' = (j_1, \ldots, j_k + 1, \ldots, j_M)$
    if P' is feasible AND $Q(P') > Q(P)$ then
        $j_k = j_k + 1$
    else
        remove k from S

Figure 5.18: Refined max-min algorithm with progressive rendering

10 dominates the transmission policy, because it is very large in size (almost twice as large as the next largest image) and it is preceded by a very short interval (half as long as the next shortest interval).

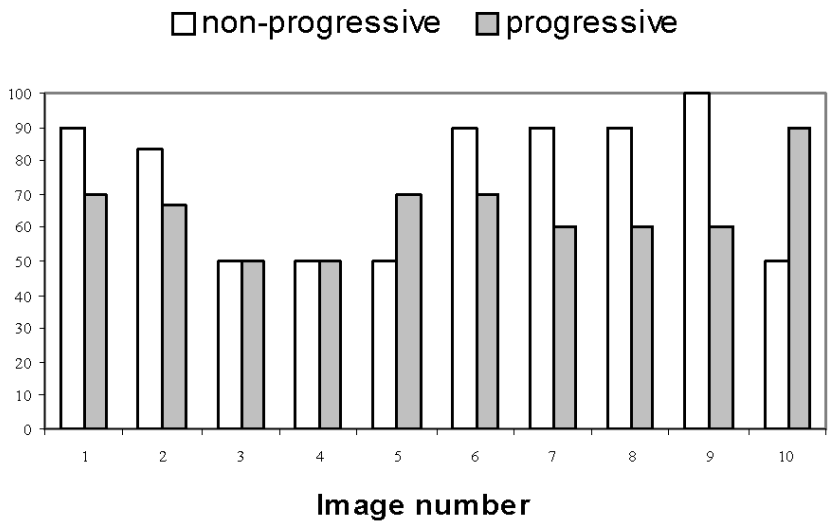

Figure 5.19: Percentage of layers sent by the refined max-min algorithm with and without progressive rendering

5.7.1 The Problem with Sequential Delivery

Underlying our work on optimal streaming of layer-encoded multimedia presentations is an assumption of sequential delivery. When each of the objects to be delivered has a hard deadline to meet, then this is a logical simplification of the problem, since there would be no reason to deliver an object with an given deadline after an object with a later deadline. However, in the case of progressive rendering of static images, there is an undesirable consequence to the restriction that objects be delivered sequentially, as ordered by their index. To make this point clear, suppose we take a simple presentation composed of two continuously scalable images with rendering intervals that are approximately equal to $[0, 20]$. Assume a startup delay of 10 seconds, and that the two images have the same number of bits, and the bandwidth allows for the transmission of half the bits over 10 seconds. With sequential delivery, and assuming that object quality is equal to the average resolution of an image across its rendering interval, 50% of the first image will be delivered over the pre-fetch interval, and then 100

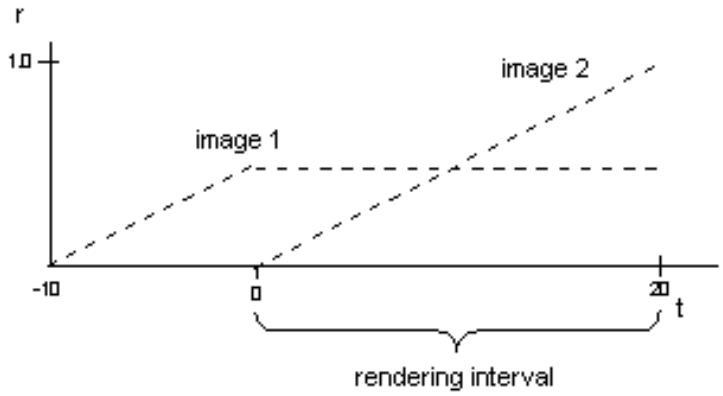

Figure 5.20: The problem with sequential delivery

Both images have the same rendered quality, namely 50

5.8 Limtied Client Memory

In this section we consider the problem of limited client memory. We develop a system of constraints that express both limitations in client memory and

transmission capacity, and present an algorithm to determine a transmission policy that is optimal in the refined max-min sense.

With an infinite client buffer, the application can transmit data at the maximum rate available, and so the bits selected to send to the client are constrained only by the available bandwidth. However, with a finite client buffer, the application needs to control the rate at which it transmits to avoid overflowing the client buffer. For this purpose, we introduce a new variable z_i that represents the number of bits the server transmits over the i^{th} interval. (See Fig. 5.21.) A transmission policy P is now represented by a pair of N-dimensional vectors $(\vec{j}, \vec{z})$, where j_i is the number of layers to send of the i^{th} object, and z_i is the number of bits to transmit over the i^{th} interval. The values j_i are integers, and the values z_i are reals.

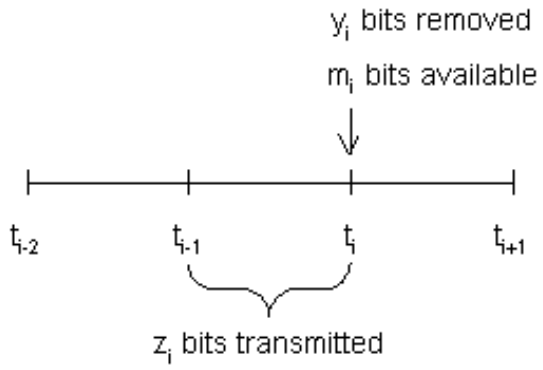

Figure 5.21: The i^{th} interval

As before, we assume the application sends the presentation data sequentially, that is, it sends the bits of object 1, then object 2, and so on. There is no loss of generality in making this assumption, because the objects are ordered by their deadlines, and all bits of an object must arrive before it's deadline. Thus, the z_i replace the b_i as the constraining resource on the bit sums, giving the following system of bit sum constraints: (Note that y_i is a function of j_i.)

$$\begin{aligned} & y_1 \leq z_1 \\ & y_1 + y_2 \leq z_1 + z_2 \\ & \vdots \\ & y_1 + \cdots + y_N \leq z_1 + \cdots + z_N \end{aligned}$$

The number of bits the application can transmit over the i^{th} interval must be less than or equal to the maximum number of bits the transmission channel allows over this interval, which we have already defined as b_i. Thus, we have the following transmission rate constraints:

$$z_i \leq b_i \text{ for } i = 1, \ldots, N.$$

We must also constrain the rate of transmission to avoid overflowing the client buffer. We assume that the bits of the i^{th} object are removed from the client buffer at the beginning of the object's playout interval, t_i. We also assume that all incoming bits pass through the client buffer. For these reasons, the amount of free storage in the client buffer increases by y_i at time t_i. (See Fig. 5.21.)

We let m_i represent the amount of free memory in the buffer at time t_i, after the removal of the i^{th} object. We let M represent the capacity of the buffer, and for notational convenience, we define $m_0 = M$. The application will not overflow the client buffer if the number of bits it transmits on the i^{th} interval is less than or equal to the amount of free memory in the buffer after removal of object $i-1$, which equals m_{i-1}. Thus, we have the following memory constraints:

$$z_i \leq m_{i-1} \text{ for } i = 1, \ldots, N,$$

The application needs to know both the number of layers j_i to transmit, and the transmission rates z_i. Thus, our new search space is $(\vec{j}, \vec{z})$, and the set of feasible solutions is defined by the three systems of inequalities given above, and by the fact that j_i and z_i are all non-negative.

The memory available at the end of the i^{th} interval m_i (at time t_i) equals the memory that was available at the beginning of the i^{th} interval, m_{i-1}, minus the number of bits transfered over the interval, z_i, plus the number of bits removed from client memory at time t_i. Thus, for $i = 1, \ldots, N$, we can write:

$$m_i = m_{i-1} - z_i + y_i.$$

By induction, we have:

$$m_i = M + (y_1 - z_1) + \cdots + (y_i - z_i),$$

for $i = 1, \ldots, N$, and so the memory constraints can be rewritten as follows:

$$z_i \leq M + (y_1 - z_1) + \cdots + (y_{i-1} - z_{i-1}).$$

Thus, feasibility can thus be expressed solely in terms of the variable that make up our transmission policy, $(\vec{y}, \vec{z})$, without relying on the intermediate variables m_i.

We present two algorithms that compute optimal transmission policies. We can classify the first algorithm as a brute force method, and the second algorithm as a greedy method. Because transmission can sometimes be arbitrarily delayed, optimal policies are not generally unique. However, our brute force and greedy algorithms converge to the same solution, which is to send the bits are early as possible. From a practical perspective, this would be favorable to delaying transmission, in the case that unexpected delays are introduced into the system that might result in missing a deadline.

The brute force algorithm is shown in Fig. 5.22. It produces a feasible transmission policy in $O(L_1 L_2 \cdots L_N)$ time. This algorithm is not absolutely brute force in the sense that it only ranges through all possible integer assignments to $(j_1, \ldots, j_N)$, and does not try every possible real assignment to $(z_1, \ldots, z_N)$. An exhaustive search is made only of the finite N-dimensional space of $\{0, \ldots, L_1\} \times \cdots \{0, \ldots, L_N\}$. Theoretically, we can not do a brute force search of the uncountable set $\{0, \ldots, x_{1.}\} \times \cdots \times \{0, \ldots, x_{N.}\}$, over which $\vec{z}$ ranges. Instead, with each assignment to $\vec{j}$, we assign the largest possible values to $z_1, \ldots, z_N$, in increasing order, permitted by the bandwidth and memory constraints on $\vec{z}$. This assignment is accomplished by executing the following for $k = 1, \ldots, N$:

$$z_k \leftarrow min(b_k, m_{k-1})$$
$$m_k \leftarrow m_{k-1} - z_k + y_k$$

After making the assignment to z_k and m_k within the k^{th} step of the loop above, we check that the cumulative bit sum of presentation objects is less than or equal to the cumulative number of bits allowed to be transmitted up to the k^{th} deadline. If this check fails, we know that all policies $P = (\vec{j}, \vec{z})$ with $\vec{j} = (j_1, \ldots, j_N)$ are not feasible. In this case, the algorithm goes to the next assignment to $\vec{j}$.

if $\sum_{i=0}^{k} y_i > \sum_{i=0}^{k} z_i$
    try next assignment to $(j_1, \ldots, j_N)$

If the cumulative bit sum check succeeds, then we know $(j_1, \ldots, j_N, z_1, \ldots, z_N)$ is feasible. Thus, we check for a quality improvement over the previously saved assignment:

if $Q(j_1, \ldots, j_N) > Q(\hat{j_1}, \ldots, \hat{j_N})$ then
    $(\hat{j_1}, \ldots, \hat{j_N}) \leftarrow (j_1, \ldots, j_N)$

The entire algorithm is given in Fig. 5.22. Note that $(\hat{j_1}, \ldots, \hat{j_N})$ represents the highest quality feasible assignment to $\vec{j}$ over the set of tested assignments. When the algorithm terminates, $(\hat{j_1}, \ldots, \hat{j_N})$ will contain an optimal assignment to $\vec{j}$. Although this is not shown, also when the algorithm completes, the assignment to $\vec{z}$ is determined with the aid of the intermediate variables $m_1, \ldots, m_N$.

```
// main processing loop
(ĵ_1, ..., ĵ_N) ← (0, ..., 0)
for the L_1 L_2 ··· L_N possible assignments to (j_1, ..., j_N)
    m_0 ← M
    for k = 1, ..., N
        z_k ← min(b_k, m_{k-1})
        m_k ← m_{k-1} − z_k + y_k
        if Σ_{i=0}^{k} y_i > Σ_{i=0}^{k} z_i
            try next assignment to (j_1, ..., j_N)
    if Q(j_1, ..., j_N) > Q(ĵ_1, ..., ĵ_N) then
        (ĵ_1, ..., ĵ_N) ← (j_1, ..., j_N)
```

Figure 5.22: Brute force algorithm for finding optimal memory-constrained transmission policies for layered multimedia presentations

The brute force algorithm works with any presentation quality metric $Q(P)$. However, if we restrict ourselves to the refined max-min metric defined in Section sec:, then we can replace the exhaustive search through $\{0, \ldots, L_1\} \times \cdots \{0, \ldots, L_N\}$ with a greedy search. In this case, we start with a null assignment to $\vec{j}$, and successively increment the component of $\vec{j}$ with minimum quality in set S. We remove elements from S when they can no longer be feasibly increased or when they have reached their layer limit L_i. We also continue the simple assignment to $\vec{z}$ via the intermediate memory variables $\vec{m}$ that satisfies the bandwidth and memory constraints placed on $\vec{z}$. Fig. 5.23 shows the entire algorithm.

The greedy algorithm is significantly more efficient than the brute force version, completing in time bounded by $O(LN^2)$. Table 5.8 shows the difference in the magnitudes of the time it takes for these algorithms to run in Java on various sample presentations, and with different bandwidth and

$S = \{1, \ldots, N\}$
$m_0 \leftarrow M$
$z_1 \leftarrow min(b_1, m_0)$
while S is not empty
    find $k \in S$ with lowest quality
    $j_k \leftarrow j_k + 1$
    $y_k \leftarrow y_k + x_{k,j_k}$
    // assign values to $\vec{z}$ that respect bandwidth and memory constraints
    for $i = 1$ to $N - 1$
        $m_i \leftarrow m_{i-1} - z_i + y_i$
        $z_{i+1} \leftarrow min(b_{i+1}, m_i)$
    // check for feasibility
    if $(j_1, \ldots, j_N, z_1, \ldots, z_N)$ is not feasible then
        $y_k \leftarrow y_k - x_{k,j_k}$
        $j_k \leftarrow j_k - 1$
        remove k from S
    else if $j_k = L_k$ then remove k from S

Figure 5.23: Greedy algorithm for finding optimal memory-constrained transmission policies for layered multimedia presentations

memory capacities. In these sample presentations, version "b" has greater bandwidth larger memory than the corresponding version "a."

Although we do not provide a proof that the greedy algorithm produces an optimal policy, this fact is believable, since the transmission policies it generates are identical with those generated by the brute force algorithm for presentations with object counts of 10 and lower.

5.9 Summary

A *multimedia presentation* is a collection of multimedia data with rendering intervals relative to the start of the presentation. In order to reduce the delay between user request for playout and the start of playout, the application begins playout before transmitting all of the data to the client. Once playout of the presentation has begun, the unsent data at the server assume delivery deadlines in accordance with the start of their rendering intervals. Starting the playout as early as possible and still meeting the future delivery deadlines is the process of *media streaming.*

If the multimedia data has a fixed representation, and the client memory

Table 5.6: Execution times in milliseconds for brute force and greedy algorithms

presentation	N	brute force	greedy
P4a	4	60	20
P4b	4	110	20
P8a	8	2123	20
P8b	8	5768	20
P10a	10	67277	20
P10b	10	188431	20
P16a	16	∞	30
P16b	16	∞	40

is sufficiently large, then the problem of delivering and rendering the presentation is a matter of determining the amount of start-up delay needed to allow enough data to be pre-fetched in order to ensure smooth playout. This will necessarily be a function of the bandwidth of the transmission channel. In the case this rate is known, the problem of start-up delay is a simple calculation. In the case that the bandwidth is unknown at the outset of transmission, the application must begin transmitting data immediately, but introduce a delay before deciding when to start playout. Its decision will necessarily be based on an estimate of future bandwidth from very recent transmission history.

If the multimedia presentation is composed of N data units, each with a finite number of data representations of varying sizes, then the application can trade off between quality and start up delay by varying the amount of data to be transmitted to the client. We consider presentations that are scalable through a layer encoding. Layer-encoded data is a storage-friendly data representation of multiple versions of a single multimedia presentation.

With layer encoded data, the application needs to deliver layers over a lossless transmission channel, because the usefulness of higher layers is dependent on the presence of the lower layers. Relative to a fixed start-up delay, there are many possible scaled versions of the presentation that can be delivered on-time to the client, which we refer to as *feasible transmission policies*. A feasible transmission policy is *optimal* relative to a presentation quality metric if no other feasible policy exists with better quality. For generic presentation quality metrics, selecting an optimal transmission policy is not polynomial bounded in the number of layered data units. We consider two different presentation quality metrics: a *total quality* metric, and a *refined max-min* metric. We present algorithms that produce optimal polices under

these metrics in polynomial time. We argue that the refined max-min metric is superior for streaming scalable presentations.

In the case of small Internet devices, such as mobile phones, PDAs, and other Internet appliances with limited client memory, it may not be possible to store the amount of pre-fetched data required under an optimal transmission policy. To solve this problem, we extend the notion of a transmission policy to include a specification of the scaling to be applied to the data, and a specification of the rate at which to transmit this data to the client. We presented an efficient algorithm that produces an optimal transmission policy under the refined max-min presentation quality metric.

Chapter 6

Optimal Streaming of Continuously Scalable Presentations

6.1 Introduction

Progressive JPEG and GIF, and MPEG-2 are multimedia encodings that possess the property of coarse grained scalability (CGS); their data representations are layered, and each successive layer contributes to increased resolution of the rendered media. JPEG-2000, MPEG-4, and Ogg Vorbis are more recently developed encoding schemes, which possess the property of fine grained scalability (FGS), which means the size of the data layers can be made either very small, or arbitrarily small. For the analytical purposes of this thesis, it is sufficient to assume the property of *continuous scalability*. With a continuously scalable object, when r percentage of bits are delivered, then r percentage of bits are used for its rendering.

As with discretely scalable media, continuously scalable media should be transported trough a reliable transport channel, such as TCP. Otherwise, one lost packet at the beginning of the stream will invalidate all packets delivered after that point related to the containing packet's object.

We assume that the same sequence of transformations are applied to the presentation data as were applied in the case of discretely scalable presentations in Section 5.2. Thus, we have a sequence of N continuously scalable discrete objects with progressively later delivery deadlines $t_1 \leq \ldots \leq t_N$.

We solve the problem of optimally scaling the presentation data as a constrained optimization under the refined max-min criterion, having argued in Chapter 5 for the superiority of the refined max-min criterion over the Total

Quality criterion for scheduling the delivery of layer-encoded data. In the current chapter, we consider the fixed bandwidth case. In environments with irregular QoS, the adaptive scheme described in Chapter 5 can be employed in an identical manner to the case of continuously scalable media.

This chapter is organized as follows. In Section 6.2, we define the set of feasible scaling vectors that satisfy the bandwidth constraints. In Section 6.3, we develop notation for the rendered quality of individual presentation elements (which we refer to as objects), and define the refined max-min quality metric, which is an overall presentation quality metric. In Section 6.4, we present an algorithm that computes a feasible scaling vector with a maximum refined max-min quality, under the identity quality metric for individual components. In Section 6.5 we describe the problem of finite client pre-fetch buffer, and show how an optimal transmission policy can be computed by solving N linear programming problems.

6.2 Feasibility

We are interested in the case in which available bandwidth is insufficient for transporting the full resolution of all presentation objects. Thus, we must determine the degree to which the data representations of each object should be scaled down so that all delivery deadlines are met, while quality is kept at a maximum.

A multimedia presentation may be composed of several different media types, such as still images, audio, video, animation, etc. For the purposes of our analysis, all of the presentation data can be organized into N continuously scalable objects with rendering intervals relative to the start of the presentation, which we take to be time $t = 0$ seconds.

As in the previous chapter, we use the term *object* to refer to a unit of scalable data with an associated delivery deadline. For instance, suppose an audio stream were composed of a sequence of 10 ms scalable segments. Each of these segments would be considered a separate object with a corresponding delivery deadline. We number the objects 1 through N, in the order of the time of the beginning of their rendering intervals, $t_1, \ldots, t_N$.

We would like to deliver (and render) the full resolution of the i^{th} object, which requires x_i bits. However, to satisfy bandwidth and client memory limitations, and an initial playout delay constraint, the application may only be able to transmit r_i percent of the i^{th} object. We refer to the vector $R = (r_1, \ldots, r_N)$ as the resolution vector. Fig. 6.1 illustrates the data model.

We say that a resolution vector R is *feasible* if transmitting the scaled data representation $(r_1 x_1, \ldots, r_N x_N)$ results in each object arriving at the

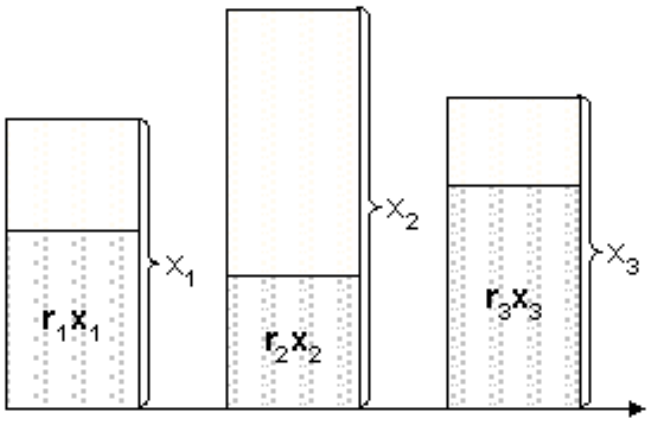

Figure 6.1: Continuously scalable multimedia objects

destination before the beginning of its rendering interval. We thus take $t_1, \ldots, t_N$ as the delivery deadlines of objects 1 through N.

Let B represent the bandwidth available to the application, and let d represent the initial delay between the start of transmission and the start of playout. As a convenience, we designate the start of transmission by t_0. Thus, $t_0 = -d$. If the deadline of the first object is at time $t_1 = 0$, we would naturally set d to some non-zero positive value to ensure that some version of the first object is rendered. (See Fig. 6.2.)

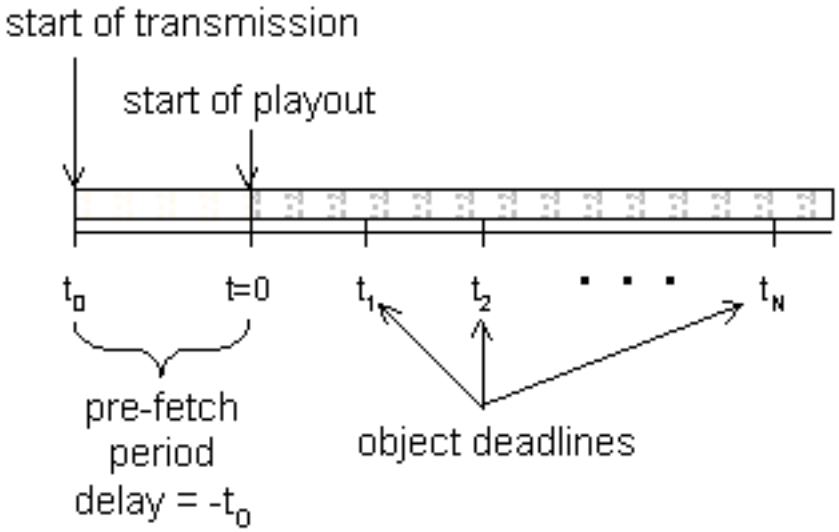

Figure 6.2: The presentation time-line

We define b_k as the number of bits the application can transfer from source to destination over the time interval $[t_{k-1}, t_k]$. Thus, $b_1 + \cdots + b_k$ represents the cumulative number of bits the application can transfer from the start of transmission until the beginning of the rendering interval of the k^{th} object. Under transmission policy R, all the deadlines will be met if the cumulative bits needed at each deadline is less than or equal to the cumulative bits that can be delivered by that deadline. Thus, the resolution vector $(r_1, \ldots, r_N)$ is feasible if the inequalities in Fig. 6.3 are satisfied. Feasibility means the resulting scaled data representation of the presentation results in smooth playout at the destination.

$$
\begin{aligned}
& r_1 x_1 \leq b_1 \\
& r_1 x_1 + r_2 x_2 \leq b_1 + b_2 \\
& \vdots \\
& r_1 x_1 + \ldots + r_N x_N \leq b_1 + \cdots + b_N
\end{aligned}
$$

Figure 6.3: Bandwidth constraints

These constraints specify the set of feasible resolution vectors, or transmission policies. Our goal is to determine a method of selecting an element from this set that maximizes the overall presentation quality.

6.3 Quality and the Refined Max-Min Criterion

We use the refined max-min criterion as described in Chapter 5 as a basis for selecting an optimal scaling vector, that is, a feasible scaling vector that maximizes the overall presentation quality. If the transmission of the resulting data representation does not violate the deadline constraints, then presentation playout is smooth, and the quality of the poorest quality object can not be improved. We assume the rendered quality of object n equals r_i, the percentage of its total bits rendered.

Table 6.3 shows 3 possible renderings of a presentation with 3 components. The max-min criterion chooses rendering A as having higher overall quality than rendering B, because the minimum object resolution in A is 0.5, where as the minimum object resolution in rendering B is 0.4. However, the max-min criterion doesn't distinguish between renderings A and C, because

Table 6.1: Scaling factors for three different presentation renderings

	rendering		
image	A	B	C
1	0.6	0.8	0.5
2	0.7	0.9	0.5
3	0.5	0.4	0.5

they have the same minimum object resolutions, even though rendering A is superior in the sense of providing higher resolutions for images 1 and 2. The *refined max-min criterion* overcomes this weakness by respecting the philosophy of the max-min criterion, but distinguishes between renderings such as A and C by fully using the available bandwidth to improve individual object resolutions beyond their minimums.

Quality under the refined max-min criterion is represented by a vector whose first component is the lowest object quality score in the rendering, the second component the second lowest object quality score, and so on. If we let a_i represent the object that takes the i^{th} position in the quality vector, then the overall presentation quality is given by

$$Q(P) = (r_{a_1}, \ldots, r_{a_M}).$$

The renderings in Table 6.3 result in the following quality vectors:

$$Q(A) = (0.5, 0.6, 0.7)$$
$$Q(B) = (0.4, 0.8, 0.9)$$
$$Q(C) = (0.5, 0.5, 0.5)$$

Now, rendering A has higher overall quality than rendering C, because their quality vectors have the same first component, but $Q(A)$ has a higher second component than $Q(C)$.

In the previous chapter, we considered a *total quality criterion*, which defines an optimal scaled version of the presentation by maximizing the sum of the qualities of individual components of the presentation. We argued that the refined max-min criterion is superior to the total quality criterion, because it (1) produces presentations with more uniform object qualities, (2) has lower storage and CPU resource requirements, and (3) satisfies the max-min philosophy while utilizing all available bandwidth to improve quality. For these reasons, we restrict our investigation to algorithms that rely on the refined max-min criterion for determining an optimal scaling vector.

6.4 Redistribution Algorithm

We say that a transmission policy P is optimal in the refined max-min sense if it is feasible and its refined max-min quality $Q(P)$ is greater than or equal to the refined max-min quality of all other feasible policies. In this section, we present an algorithm, called the redistribution algorithm, that converges to an optimal policy. However, before presenting the algorithm, we first demonstrate that the components of an optimal resolution vector are non-decreasing. We state this fact as a lemma, because we will use it when proving the optimality of the redistribution algorithm.

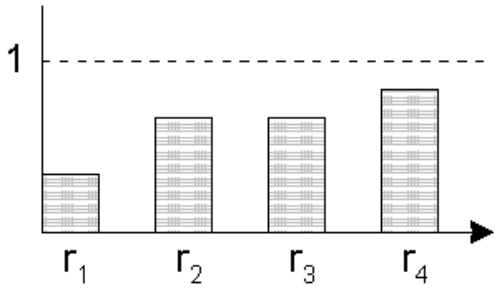

Figure 6.4: Nondecreasing lemma

Lemma 2 *Relative to a system of bandwidth constraints, the components of an optimal resolution vector are non-decreasing in their index.*

Proof
For a given presentation, let $R = (r_1, \ldots, r_N)$ be an optimal resolution vector relative to the bandwidth constraints in (6.3). We show that for all $i < j$, $r_i \leq r_j$.
Suppose there exists i and j such that $i < j$ and $r_i > r_j$. We show that this contradicts the optimality of R by constructing a feasible resolution vector R' with quality better than $Q(R)$.

Clearly, we can choose $\delta > 0$ sufficiently small so that

$$r_i > r_i - \frac{\delta}{x_i} > r_j + \frac{\delta}{x_j} > r_j.$$

Choose R' such that $r'_k = r_k$ for $k \neq i, j$. For i and j, take $r'_i = r_i - \frac{\delta}{x_i}$, and $r'_j = r_j + \frac{\delta}{x_j}$. R' satisfies feasibility constraints 1 through $i-1$, because they are identical to those of R. Constraints i through $j-1$ are identical with those of R, except for the i^{th} term, which equals $(r_i - \frac{\delta}{x_i}) \cdot x_i$. Because δ

is positive, this term is less than the i^{th} term in the corresponding constraint of R. Constraints j through N differ from those of R only in the i^{th} and j^{th} terms, which equal $(r_i - \frac{\delta}{x_i}) \cdot x_i$ and $(r_j + \frac{\delta}{x_j}) \cdot x_j$, respectively. Simplification of these terms leads to $r_i \cdot x_i + r_j \cdot x_j$, and so constraints j through N are identical for R and R'. Thus, R' is feasible.

Now we show that $Q(R') > Q(R)$. R and R' share the same components except in positions i and j. Let n be the number of these common components that are less than or equal to r_j. (The remaining $N - n - 2$ common components are strictly greater than r_j.) $Q(R)$ and $Q'(R)$ must match in the first n places. The $(n+1)^{st}$ place of $Q(R)$ contains r_j, and the $(n+1)^{st}$ place of $Q(R')$ contains a value greater than r_j. ■

This lemma provides us with an insight into how we might efficiently determined an optimal policy. We pass through the objects from 1 to N. In pass k we initialize the resolution of object k to what can be achieved by using the bandwidth following the deadline of object $k-1$, that is, the bandwidth within interval i. If this resolution is less than the resolution of any of its predecessor objects, then we re-allocate bandwidth from one or more earlier intervals so that the resolution of object k is equal to the resolution of its predecessor object $k-1$. This algorithm, appearing in Fig. 6.5, converges to an optimal refined max-min resolution vector that respects the bandwidth constraints.

```
for k = 1 to N
        b ← b_k
        x ← x_k
        r ← b ÷ x
        j ← k
        while (j ≥ 2) and (r_{j-1} > r)
                j ← j − 1
                b ← b + b_j
                x ← x + x_j
                r ← b ÷ x
        for i = j to k
                r_i ← r
```

Figure 6.5: The redistribution algorithm

Fig. 6.6 graphically illustrates the functioning of the algorithm through several steps of execution, starting from the first pass in the outer loop. In the

first pass, the algorithm allocates all the bandwidth in the first interval for delivery of the first object. The result is some value for r_1, as depicted in the figure. The next pass through the outer loop selects the second object, and allocates all the bandwidth in the second interval to the delivery of the bits of object 2. The result, r_2, is a resolution greater than the first object, and so the algorithm continues in the outer loop. It allocates all the bandwidth in the third interval to the third object, with resulting resolution better than previous objects, and so the algorithm goes on to the fourth object. But when the algorithm uses only the bandwidth in the fourth interval for delivering object 4, the result is a resolution value that is inferior to the previous object. The algorithm then enters the inner loop in which it steps backwards in time, reallocating bandwidth in order to improve the resolution of the fourth object by decreasing the resolution of earlier objects. In the first step of the inner loop, the algorithm adjusts the allocation of bandwidth between objects 3 and 4, resulting in identical resolutions between these two objects. However, the resulting resolution is still lower than an earlier object, namely object 2. Thus, the algorithm remains in the inner loop, and decreases the number of bits delivered of the second object in order to increase the number of bits sent of objects 3 and 4. The result is identical resolutions for objects 2, 3 and 4. At this point, the resolutions now form a non-decreasing sequence in time, and the algorithm exits the inner loop in order to continue with the outer loop, in which it considers object 5, and so on.

At step k in the outer loop, the algorithm sets the resolution of the k^{th} object so that equality holds in the k^{th} constraint, using the values of $r_1, \ldots, r_{k-1}$ set during previous passes. The algorithm then checks the resolution of preceding object $k-1$; if its resolution r_{k-1} is greater than r_k, the algorithm goes into a inner loop in which it redistributes bandwidth so that r_k is increased and r_{k-1} is decreased to the same value. It continues this process of redistribution until the preceding resolution is less than or equal to r_k. Thus, by design, the algorithm converges to a resolution vector in which the components are non-decreasing in their index.

At intermediate points in its execution, the algorithm may set components of the resolution vector to values greater than 1. When this occurs, it represents excess capacity of the transmission channel to transport bits of succeeding objects. When the algorithm in Fig. 6.5 terminates, the application should set to 1 any resolutions that exceed 1.

Theorem 3 *Under a system of bandwidth constraints, the redistribution algorithm converges to an optimal resolution vector in the refined max-min sense.*

Proof

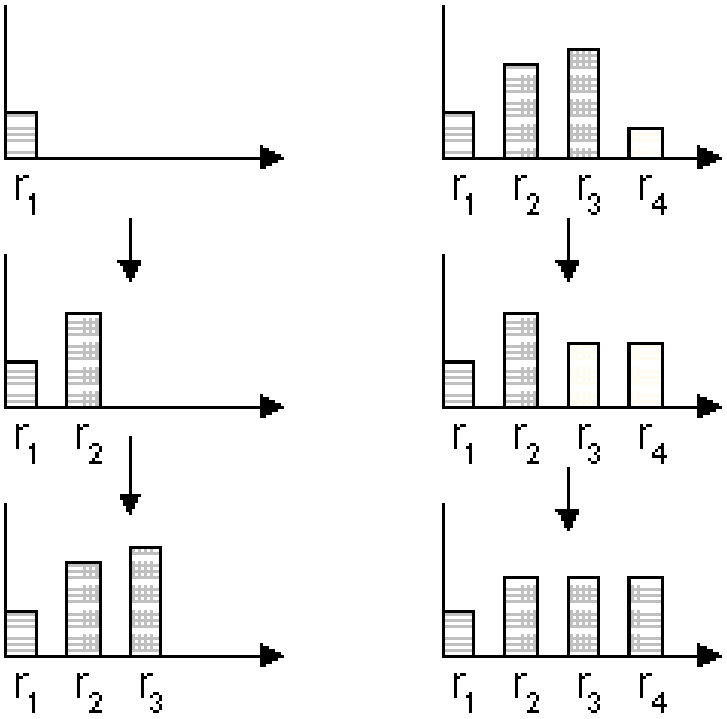

Figure 6.6: Redistribution algorithm

For a given presentation, suppose the optimal resolution vector is $\hat{R} = (\hat{r}_1, \ldots, \hat{r}_N)$, and the redistribution algorithm converges to resolution vector $R = (r_1, \ldots, r_N)$. We show by induction that $R = \hat{R}$.

Step 1: We show $r_1 = \hat{r}_1$. If $r_1 > \hat{r}_1$, then $\hat{R}$ would not be optimal. We demonstrate equality by showing that $r_1 < \hat{r}_1$ leads to a contradiction. Assume that the algorithm makes a final assignment to r_1 in step k. The algorithm redistributes the available bandwidth evenly between objects 1 through k as follows:

$$r_1 = \cdots = r_k = \frac{b_1 + \cdots + b_k}{x_1 + \cdots + x_k}. \tag{6.1}$$

Since the values for r_1 through r_k are identical, we have:

$$r_1 x_1 + \cdots + r_k x_k = r_k x_1 + \cdots + r_k x_k = r_k(x_1 + \cdots + x_k)$$

Substituting (6.1) into the right-hand side of this expression yields:

$$r_1 x_1 + \cdots + r_k x_k = b_1 + \cdots + b_k,$$

and thus the k^{th} bandwidth constraint holds for equality for R.

From Lemma 2 we have that $\hat{r}_i \geq \hat{r}_1$ for $i = 1, \ldots, k$. Combining this with our assumption that $\hat{r}_1 > r_1$, and our observation that $r_1 = \cdots = r_k$, gives $\hat{r}_i > r_i$ for $i = 1, \ldots, k$. Therefore, in the k^{th} bandwidth constraint for $\hat{R}$,

$$\hat{r}_1 x_1 + \cdots + \hat{r}_k x_k \geq b_1 + \cdots + b_k,$$

we can decrease each term $\hat{r}_i x_i$ by a positive value to arrive at $r_i x_i$, demonstrating

$$r_1 x_1 + \cdots + r_k x_k < b_1 + \cdots + b_k.$$

This contradicts our finding that the k^{th} bandwidth constraint holds for equality for R.

Step 2: Assume that $r_i = \hat{r}_i$ for $i = 1, \ldots, k-1$; we show $r_k = \hat{r}_k$. If $r_k > \hat{r}_k$, then, according to Lemma 2, $Q(\hat{R}) < Q(R)$, which contradicts the optimality of $\hat{R}$. We demonstrate equality by showing that $r_k < \hat{r}_k$ leads to a contradiction. Assume that the algorithm makes a final assignment to r_k in step j, and when it does, it redistributes available bandwidth evenly between objects i through j as follows:

$$r_i = \cdots = r_k = \cdots = r_j = \frac{b_i + \cdots + b_j}{x_i + \cdots + x_j}. \tag{6.2}$$

Because redistribution stops at i, the algorithm only made a single assignment to r_{i-1}, which results in the i^{th} constraint holding for equality with values $r_1, \ldots, r_{i-1}$. Therefore, we know

$$r_1 x_1 + \cdots + r_{i-1} x_{i-1} = b_1 + \cdots + b_{i-1}. \tag{6.3}$$

Since the values for r_i through r_j are identical, we have:

$$r_1 x_1 + \cdots + r_{i-1} x_{i-1} + r_i x_i + \cdots + r_j x_j = r_1 x_1 + \cdots + r_{i-1} x_{i-1} + r_j (x_i + \cdots + x_j)$$

Substituting (6.2) and (6.3) into the right-hand side of this expression yields:

$$r_1 x_1 + \cdots + r_j x_j = b_1 + \cdots + b_j,$$

and thus the j^{th} bandwidth constraint holds for equality for R.

Our inductive assumption that $\hat{r}_n = r_n$ for $n = 1, \ldots, i-1$ allows us to rewrite the j^{th} bandwidth constraint for $\hat{R}$ as follows:

$$r_1 x_1 + \cdots + r_{i-1} x_{i-1} + \hat{r}_i x_i + \cdots + \hat{r}_j x_j \geq b_1 + \cdots + b_j. \tag{6.4}$$

From Lemma 2 we have that $\hat{r}_n \geq \hat{r}_i$ for $n = k, \ldots, j$. Combining this with our assumption that $\hat{r}_k > r_k$, and our observation that $r_i = \cdots = r_k = \cdots =$

r_j, gives $\hat{r}_n > r_n$ for $n = k, \ldots, j$. Thus, we can decrease the terms $\hat{r}_n x_n$ in (6.4) by a positive value to arrive at $r_n x_n$, demonstrating

$$r_1 x_1 + \cdots + r_j x_j < b_1 + \cdots + b_j.$$

This contradicts our finding that the j^{th} bandwidth constraint holds for equality for R. ■

6.5 Client Memory Constraints

In this section we consider the problem of limiting client memory. We develop a system of constraints that express both limitations in client memory and transmission capacity, and show how an optimal transmission policy can be arrived at using linear programming.

The application can no longer transmit data at the maximum available rate, because it may overflow the client buffer if data is not removed from it quickly enough. We let z_i represent the rate at which the server transmits data in the i^{th} interval (Fig. 6.7).

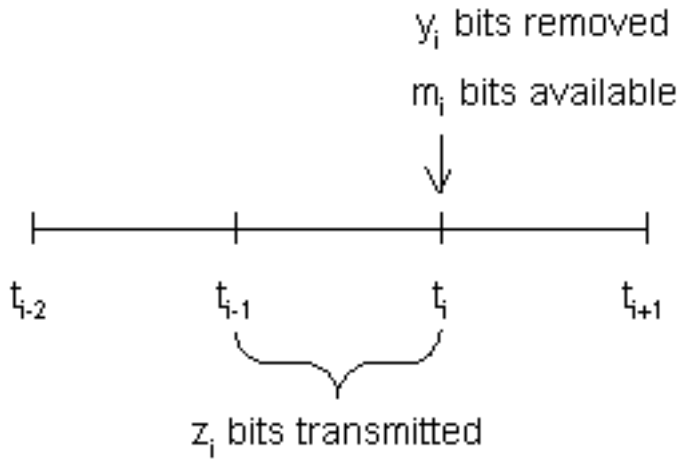

Figure 6.7: The i^{th} interval

As before, we assume the application sends the presentation data sequentially, that is, it sends the bits of object 1, then object 2, and so on. There

is no loss of generality in making this assumption, because the objects are ordered by their deadlines, and all bits of an object must arrive before the object's deadline. Thus, the z_i replace the b_i as the constraining resource on the resolution factors, giving the following constraints on the resolution factors:

$$\begin{aligned} & r_1 x_1 \leq z_1 \\ & r_1 x_1 + r_2 x_2 \leq z_1 + z_2 \\ & \vdots \\ & r_1 x_1 + \cdots + r_N x_N \leq z_1 + \cdots + z_N \end{aligned}$$

Figure 6.8: Resolution factor constraints

The transmission rate z_i is constrained by the maximum transmission rate on the i^{th} interval, and by the unused space in the client memory buffer at the beginning of the interval. Thus, the transmission rate constraints are

$$z_i \leq b_i \text{ for } i = 1, \ldots, N,$$

and the memory constraints are

$$z_i \leq m_{i-1} \text{ for } i = 1, \ldots, N,$$

where m_i is the number of available (unused) bits in client memory just after the removal of object i at time $t = t_i$. (See Fig. 6.7.) For convenience, we let m_0 represent the capacity of the client buffer M.

The application needs to know both the resolution factors r_i, and the transmission rates z_i. Thus, our new search space is $(\vec{r}, \vec{z})$, and the set of feasible solutions is defined by the three systems of inequalities given above, and by the fact that r_i and z_i are all non-negative. Our next step is to eliminate the m_i from the constraints.

At each deadline t_i the application removes $r_i x_i$ bits from the pre-fetch buffer for rendering. This means the free space in the buffer just after this removal has increased by $r_i x_i$. But the free space in the buffer just before removal is the free space made available by the removal of object $i-1$ at time t_{i-1}, minus the number of bits transmitted across the i^{th} interval z_i. Thus, for $i = 1, \ldots, N$, we can write:

$$m_i = m_{i-1} - z_i + r_i x_i.$$

By induction, we have:

$$m_i = M + (r_1 x_1 - z_1) + \cdots + (r_i x_i - z_i).$$

Thus, the memory constraints can be rewritten as follows:

$$z_i \leq M + (r_1 x_1 - z_1) + \cdots + (r_i x_i - z_i).$$

It is easy to see that the three sets of constraints have the form:

$$\sum_{i=1}^{N} c_{ij} z_i \leq M_j,$$

for $j = 1, \ldots, J$, and $z_j \geq 0$.

We can determine the max-min resolution factor relative to these constraints by introducing a dummy variable y, and solving the following linear programming problem:

$$\begin{array}{l} \text{maximize } y \text{ such that} z_i \geq y, \text{ and} \\ \sum_{i=1}^{N} c_{ij} z_i \leq M_j,\ j = 1, \ldots, J \end{array}$$

After this problem is solved, we will know the first component of the refined max-min quality vector. The procedure is then to fix the value of the minimum resolution factor, thus reducing the space by one, and apply the procedure again to obtain the next minimum. This procedure is continued until all of the resolution factors are determined.

6.6 Summary

In this chapter, we considered multimedia presentations that are composed of *continuously scalable* media, that is, data encodings that support fine grained scalability (FGS). We use the term *continuous*, because in the transmission policy we represent the scaling of a data unit with a real number rather than an integer, and object resolutions vary continuously across $[0, 1]$. We first present an efficient algorithm that produces an optimal policy for clients with infinite pre-fetch buffers. Then, as in the discrete case, we extend the notion of a transmission policy to include a specification of the transmission rate as a function of time, and we presented an algorithm that produces a corresponding optimal policy.

One advantage of continuous scalable data is that it avoids the problem of breaking quality ties when computing an optimal policy, which is an issue in the case of discretely scalable data. In the case of discretely scalable

data, transmission policies may result in playouts that "flicker" in the sense that quality levels between contiguous objects may change often. This problem is avoided in the case of continuously scalable media, because optimal transmission policies will always be composed of non-decreasing sequences of resolution scores. In fact, the results in this chapter suggest that an intelligent tie breaking strategy in the discrete case would be to always pick the latest object for a quality improvement.

Chapter 7

Conclusion

7.1 Summary

7.1.1 Continuous Media Email

In Chapter 3, we identified the major weaknesses of Internet email that obstruct the development of continuous media messaging. These include a faulty cost model, in which the recipient bears much of the cost of email delivery; the duplication of message data within mail systems, which occurs when email is sent to multiple recipients; the wasteful delivery of non-rendered message data; and a lack of sensitivity to the end user's access rate, which results in excessive delays in retrieving large messages. We showed how all of these problems are solved with a sender-stored message delivery architecture, where a small text message (base message) is sent that allows the recipient system to stream the continuous media message content from the sender's storage.

In order that our sender-stored email delivery architecture be backward compatible with existing systems, we described an implementation in which changes are only required to sender systems. We described the three major Internet UA systems (POP, IMAP and Web-based), and we discussed how each of these systems can accommodate sender-stored email. We also described how existing security mechanisms based on public key cryptography can be use to ensure the privacy and integrity of sender-stored continuous media email.

Because sender-stored email is a major paradigm shift for existing email, it engenders several new problems. First, there is a QoS problem, which results from the streaming delivery of continuous media across a bandwidth-limited network path. Second, there is the problem of deciding when to delete message data from the sender's outbox, which the recipient may wish

to access at an unknown point in time. Third, sender-stored email introduces complexities into the process of forwarding and replying with annotation (embedding pieces of the original message in the reply).

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 continuous media 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.

7.1.2 Voice Message Boards

In Chapter 4 we discussed the Web as a medium for hosting asynchronous conferences that allow participants to leave voice messages in addition to text within a hierarchically structured tree. After discussing the advantages of adding audio to message boards, we describe a prototype system that we implemented, which relies on a combination of relatively new Web technologies, including ActiveX controls, Java and dynamic HTML. We made special efforts to keep our implementation free of any special software installation, and thus allow users to begin participating in the conference immediately upon arrival to the conference site. While the current state of Web infrastructure supports the submission of text data from client to server, by the HTML FORM element or through a Java applet, there does not yet exist an analogous method for capturing audio data at the client and delivering it to the server. Our prototype system relies on an ActiveX control to provide this missing functionality, which limits the system's accessibility to users of Internet Explorer under Windows. We describe two proposed technologies that would provide developers with a platform independent means of capturing audio data and submitting it to a server.

7.1.3 Optimal Streaming of Stored Media

An asynchronous multimedia message is an instance of what we call a *multimedia presentation*, which is a collection of multimedia data with rendering

intervals relative to the start of the presentation. In order to reduce the delay between user request for playout and the start of playout, the application begins playout before transmitting all of the data to the client. Once playout of the presentation has begun, the unsent data at the server assume delivery deadlines in accordance with the start of their rendering intervals. Starting the playout as early as possible and still meeting the future delivery deadlines is the process of *media streaming*.

If the multimedia data has a fixed representation, and the client memory is large enough to store the entire presentation, then the problem of delivering and rendering the presentation is a matter of determining the amount of start-up delay needed to allow enough data to be pre-fetched in order to ensure smooth playout. This will necessarily be a function of the bandwidth of the transmission channel. In the case this rate is known, the problem of start-up delay is a simple calculation. In the case that the bandwidth is unknown at the outset of transmission, the application must decide when to start playout based on its estimate of future bandwidth.

If the multimedia presentation has a scalable data representation, then the application can trade off resolution (or quality) with start up delay. We consider presentations that are scalable through a layer encoding. With layer encoded data, the application needs to deliver layers over a lossless transmission channel, because the usefulness of higher layers is dependent on the presence of the lower layers. Relative to a fixed start-up delay, there are many possible scaled versions of the presentation that can be delivered on-time to the client, which we refer to as *feasible transmission policies*. A feasible transmission policy is *optimal* relative to a presentation quality metric if no other feasible policy exists with better quality. For generic presentation quality metrics, selecting an optimal transmission policy is not polynomial bounded in the number of layered data units. We consider two different presentation quality metrics: a *total quality* metric, and a *refined max-min* metric. We present algorithms that produce optimal polices under these metrics in polynomial time. We argue that the refined max-min metric is superior for streaming scalable presentations.

In the case of small Internet devices, such as mobile phones, PDAs, and other Internet appliances with limited client memory, it may not be possible to store the amount of pre-fetched data required under an optimal transmission policy. To solve this problem, we extend the notion of a transmission policy to include a specification of the scaling to be applied to the data, and a specification of the rate at which to transmit this data to the client. We presented an efficient algorithm that produces an optimal transmission policy under the refined max-min presentation quality metric.

Finally, we considered multimedia presentations that are composed of

continuously scalable media, that is, data encodings that support fine grained scalability (FGS). We use the term *continuous*, because in the transmission policy we represent the scaling of a data unit with a real number rather than an integer. We first present an efficient algorithm that produces an optimal policy for clients with infinite pre-fetch buffers. Then, as in the discrete case, we extend the notion of a transmission policy to include a specification of the transmission rate as a function of time, and we presented an algorithm that produces a corresponding optimal policy.

7.2 Future Work

7.2.1 Continuous Media Email

Sender-stored email is currently a reality in the marketplace. In fact, the conclusions regarding sender-stored continuous media email were made during the initial phases of research on this thesis, and were publicly presented and published before the technique became widespread. However, the sender stored delivery systems that are currently being employed have not implemented the extended functionality described in this research. Specifically, the message deletion policy remains primitive: message data is preserved for a fixed interval of time, such as two weeks, and then deleted from sender-side storage. The two types of forwarding have not yet been implemented, and neither has any advanced form of replying that includes annotation. Thus, there exists the opportunity to implement prototype systems that incorporate these functions. Prototype development of these advanced features are intended for a Web-based email system.

7.2.2 Voice Message Boards

Following the implementation and demonstration of Aconf, voice message boards have since become a reality in the marketplace. However, several problems still remain unsolved. One problem is to provide a more reliable and universal voice capture mechanism. One way to do this is to provide a mixed mode user interface that combines the graphical user interface of a desktop PC, Web TV, etc. with the audio channel of a telephone. In fact, mixed communication channels comprise an area of distributed multimedia applications that warrant research and study.

Another area that needs work is in providing continuous media compression and skimming mechanisms to the user interface (and modifying the underlying media transport service accordingly). This will reduce the time

it takes for message consumers to process stored continuous media messages, and thus make the processing of audio and video messages more efficient, and possibly more efficient than processing text.

Multimedia indexing and searching are also important technologies that need to be brought to voice message boards, or multimedia message boards in general.

Voice user interfaces (VUI) is also a natural area to consider in extending the usefulness of voice message boards. Methods of navigating trees or other structures need to be implemented within voice recognition systems for searching archived voice messages.

7.2.3 Streaming Stored Multimedia

The refined max-min metric is a candidate for the best playout quality metric in the context of multimedia data sets that are comprised of media units with short playout durations and hard delivery deadlines. But in the case of media units with long playout durations and soft delivery deadlines, the refined max-min metric may need to be modified or replaced. Research on this problem is ongoing.

In this thesis, efficient algorithms that can be used by implementers were demonstrated for three of four important cases. In the case of continuously scalable media with a client pre-fetch buffer constraint, a more efficient algorithm may be possible. It is believed that a modified form of the redistribution algorithm may be a solution to this problem.

This thesis paves the way for developing a more general theory of streaming stored multimedia. The theorems presented can be placed in a more general context, and possibly used to solve related problems involving the delivery of a sequence of data units with delivery deadlines through a bandwidth-limited transmission channel to a destination with a memory constraint. Possibly, the resulting theorems would have relevance in the fields of content distribution and optimal streaming of synchronous (live) media.

More experimentation with emerging multimedia encodings such as MPEG-4, JPEG-2000 and Ogg Vorbis are needed to further validate the algorithms presented in this thesis, and to accelerate their use in real world applications. Also, sensible object quality metrics need to be determined by user testing. These quality metrics will necessarily be a function of the specific multimedia encoding and end system characteristics.

In the case of presentations with a finite number of different representations, this thesis assumed a layered encoding scheme, which conserves storage resources. Other methods exist for providing different data representations to attain different bit rates and playout qualities. For example, suppose

someone creates a video message. This message could be stored in three different encodings: video, audio, and text. The multimedia delivery system could have passed the speech signal through a speech recognition system to generate the text. Alternatively, the message author could enter the text in addition to the video. Or perhaps a hybrid form of the two methods could be used: as the author records her video message, she watches a corresponding stream of text generated by the speech recognition system. Currently, such a system would require her to train her system to decrease recognition errors. In the case of this example, it is clear that the number of layers transmitted should not fluctuate greatly, which might occur if playout intervals are short. This concern is related to the issue of quality ties. One way the algorithm can break a tie is by choosing the least fluctuating transmission policy. Thus, we could make a decision between ties by applying a secondary optimization criterion to a quality metric that expresses "playout flicker". For example, we could represent fluctuation of a transmission policy $F(P)$ as follows:

$$F(P) = \sum_{n=1}^{N-1} |j_{n+1} - j_n|. \tag{7.1}$$

In the greedy refined max-min algorithm, we can choose the largest k in S (with minimum quality across S), and increment its layer count. That this rule minimizes F across an interesting set of feasible policies is not determined in this thesis, and is therefore stated as an open problem.

Appendix A

SMTP

A.1 Basic Text Email

Internet email works because there is universal agreement regarding the format of messages and their method of transport: arbitrary senders can send messages to arbitrary recipients. In general, an Internet email message is 7-bit ASCII text data that conforms to the specification given in RFC 822[RFC822], and the transport of such data is accomplished using the Simple Mail Transport Protocol (SMTP)[SMTP]. That all participants of Internet email support the RFC 822 format and SMTP is the reason why Internet email is universal.

An RFC-822 message is a string of ASCII characters, which are divided into lines that are no longer than 1000 characters in length. The line delimiter is a carriage return character (CR) followed by a line feed character (LF). These lines are further separated into two sections: a header section, followed by a body section. The header section is separated from the body by a blank line, which is a line composed of only its terminating CRLF sequence. The header section contains one line for each header. The headers provide meta-information, such date of message, subject, sender's return address, etc. The body section contains the message text. The remaining details of the RFC-822 message format can be found in [RFC822].

To deliver an email message, the sending entity needs to have the email address of the recipient. This email address is composed of a user ID, followed by "@," followed by an Internet domain name. With the domain name of the address, the sender queries the Internet domain name system (DNS) for the IP address of a mail server responsible for handling mail for the domain. After the sender has this IP address, it initiates a TCP connection with the host at this IP address through port 25. After the TCP connection

is established, the mail server begins a dialog with the sender using the Simple Mail Transfer Protocol (SMTP), as illustrated in Fig. A.1. Within this dialog, the sending entity plays the role of controlling client and the mail server plays the role of subordinate server. The server begins the dialog by sending a greeting message, which is a machine-readable 3-digit response code optionally followed by a space and some human-readable text.

```
C: (establishes TCP connection with server through port 25)
S: 220 aaa.com mail server ready
C: HELO alice.aaa.com
S: 250 aaa.com
C: MAIL FROM: <alice@aaa.com>
S: 250 OK
C: RCPT TO: <bob@bbb.com>
S: 250 OK
C: DATA
S: 354 Start transmission
C: (client now sends message)
C: .
S: 250 OK
```

Figure A.1: SMTP message transport

After receiving the server's greeting, the client sender is required to issue the HELO command in which it identifies itself to the server. If the server accepts the sender, it returns a response with code 250 followed by its domain name. At this point, the sender has established a communication channel with the server. The next step is for the client to announce the email address of the sender with the MAIL FROM command. If the server accepts the sender, it replies with code 250. After receiving agreement with regard to the sender, the client next announces the address of the recipient. If the server replies affirmatively, it is ready to accept the message data. The next step is for the client is to issue the DATA command to announce its intention to begin transmission of the message data. After issuing the DATA command, the client then waits for the *start transmission* response from the server. Upon receipt of the start transmission response, the client begins sending the RFC 822 message data, which is guaranteed to be composed of lines of ASCII text that are terminated by CRLF and are less than 1000 characters in length. The client indicates the end of message data by sending a line that contains a single period. The server acknowledges receipt of the message data by returning code 250.

Because it is feasible that within the user's message there is a line that includes a single period, a "transparency procedure" is used, as follows. If the sender encounters a line of text that begins with a period, then it inserts an additional period at the beginning of the line before sending it to the server. When the server receives a line that begins with a period, it checks to see if there are any more characters in the line. If it finds no other characters, then it assumes the client is indicating that it has finished sending the message data. If there are other characters in the line, then the server deletes the initial period.

A.2 Transport of MIME-Formatted Email

The RFC-822 format of messages and the SMTP protocol that enables the transport of messages into recipient mailboxes form the basis of Internet email. However, the RFC-822 message format is inadequate for messages that contain anything other than a single body of ASCII text. To provide for multiple-body messages composed of any type of data, the Multipart Internet Mail Extension (MIME) was developed[MIME].

In order to allow incremental adoption by email systems, the MIME format was designed so that MIME-compliant messages also comply with the syntax rules of RFC 822 messages. This was accomplished by encoding non-ASCII data into ASCII, and providing a way of dividing the single body of an RFC-822 message into multiple parts. Special headers are used to identify subdivisions of the body, the type and format of the data contained within each subdivision, and the ASCII encoding scheme that was applied to the data. For binary data (such as images, audio and video), the base 64 encoding is used. The algorithm maps groups of three bytes into four bytes by dividing the 24 contiguous bits of the three-byte groups into four 6-bit pieces, which are then mapped onto a set of 64 ASCII characters. The result of the encoding is to increase the size of data representation by 33%. Thus, continuous-media messages, which are already large, become even larger when transported in ASCII encoded form.

One problem with SMTP is the waste encountered by having to encode native binary data into lines of ASCII text. One way to eliminate this waste is to use the extended commands CHUNKING and BINARYMIME, which are defined in RFC 1830[RFC1830]. These two commands enable the sender to transfer messages that contain binary data in their original form. However, for an MTA to accomplish this, it must be prepared to reformat a message it handles if the message source and destination processes do not both support the extended functionality. For example, if the source of a message supports

the transfer of binary objects, but the destination of the message does not, then the MTA must base 64 encode those body parts containing binary data, and appropriately modify the relevant body part headers before transferring the message to its destination. Similarly, if the source sends base 64 encoded data within a message, and the destination is prepared to accept binary message data, the MTA will need to decode the embedded data and alter body part headers in order to use the binary transport mechanism.

Let's consider an example in which the MIME message format is used in conjunction with SMTP to deliver a video message. Suppose that Alice recorded herself speaking for one minute using MPEG-1 at 1.5 Mbps. She saves the file with the name hibob.mpg. In her user agent she types a short text message to supplement her video message and attaches the video file to her message. She addresses the message to Bob and presses the send button. First, we will examine the structure of the message that her UA creates, and then we will examine how it is transported to Bob's mailbox.

Fig. A.2 contains the message that Alice's UA created. Notice that it follows the organization scheme of RFC-822; because there is a header section, followed by a blank line, followed by a body. The header section contains the various pieces of meta-information, such as an identification of the sender, the recipient, message date, etc. In particular, note the presence of the Content-Type header, which declares that the body contains a MIME multipart/mixed object. Also within the main header section, a boundary attribute is defined, which specifies a string of characters used to demarcate the beginning and ending of the various body parts.

In our example, there are two separate parts within the main body, besides a small area of human-readable text preceding the first boundary string, which is ignored by MIME parsers. Each part follows the basic structure of a body part; it has a header section, followed by a blank line, followed by a body section. The boundary string is used to begin each part. The final part is also followed by the boundary string, but it is appended with two dashes, "–."

The Content-Type of the first body part is text/plain, indicating that the body contains plain text. The charset attribute is defined within the Content-Type header to indicate the character set used. The Content-Transfer-Encoding indicates the body contains 7-bit data. The body contains the text created by Alice.

The Content-Type of the second body part is video/mpeg, indicating that the body contains an MPEG data file. The Content-Transfer-Encoding indicates the data has been transformed into simple ASCII text using the base 64 algorithm. The Content-Disposition header indicates the recipient UA should treat the data carried by this body part as an attachment, which

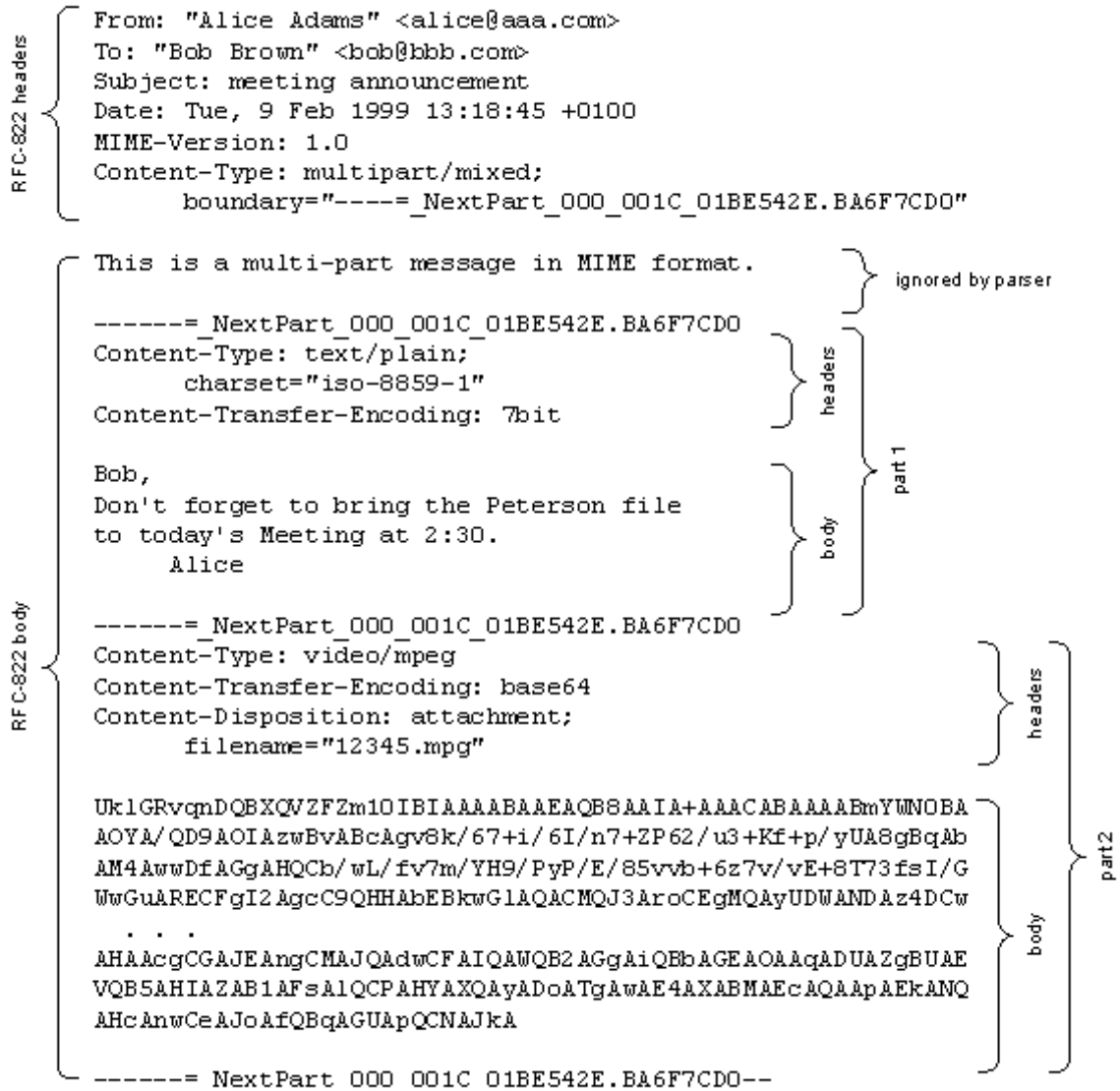

Figure A.2: Message with base-64 encoded data

means the UA should not attempt to interpret the data for immediate display, but should present to the user an option to open the attached data. The filename attribute is defined within the Content-Disposition header as a suggested name for the UA to display to the user. Actually, when the Content-Type is given as application/octet-stream, some UA infer the data format from the extension specified in the filename, and launch an appropri-

ate media player based on this inference.

Notice that the body of the second part contains only the upper and lower alphanumeric characters, the forward slash, and the plus sign. These are the 64 ASCII characters used in the base 64 encoding scheme.

Now let's consider one possible scenario for the route that this message takes to Bob's mailbox. Suppose that Alice is using a UA that runs as a stand-alone application on her desktop computer, and that it is configured to transfer outgoing messages to a nearby mail transfer agent (MTA), which is provided by her mail service provider. (Frequently, this MTA is the same process that accepts incoming messages and places them in Alice's mailbox.) This MTA relays her messages to their next destination, which is usually the server that manages the recipient's mailbox. Alice's UA uses SMTP to transfer her messages to the MTA in her domain, and this MTA also uses SMTP to transfer these messages to the MTA in the recipient's domain (Fig. A.3).

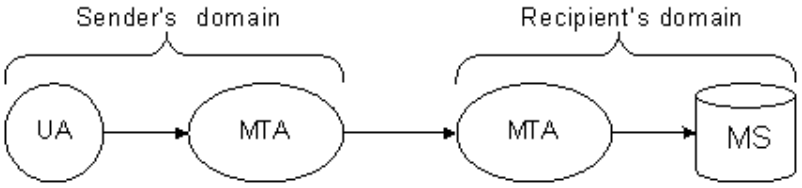

Figure A.3: SMTP message transport

Because the message in our example is a one-minute 1.5-Mbps MPEG video clip, its size, after base 64 encoding, is 15 MB. In today's environment, such a message is frequently too large to pass into the recipient's storage, either because the mailbox lacks available space, or because messages of this size are rejected by policy. In the case that the recipient's mailbox lacks enough available space, the server responds with following:

```
452 Insufficient system resources
```

In the case that the server rejects messages of this size regardless of its storage resources, it responds with the following:

```
552 Message size exceeds maximum message size
```

In either case, a large amount of bandwidth was consumed in the process of trying to send the data, which was eventually rejected. To avoid this

situation, and deal with other inadequacies, Extended Simple Mail Transport Protocol (ESMTP) was defined in [ESMTP].

Under ESMTP, the client first queries the server to determine if it supports a specific extended command. It does this by sending EHLO rather than the usual HELO after receiving the server's greeting. If the server responds that the command is unknown, then the client must send a HELO and restrict itself to the original set of SMTP commands. However, if the server supports ESMTP, it will respond to EHLO with a list of extended commands that it supports.

Fig. A.4 shows how Bob's MTA would reject Alice's video message using ESMTP. The server responds to the EHLO command with a multi-line response, where a dash is used immediately after the response code to indicate the presence of one or more additional lines. This multi-line response contains the list of extended commands that the ESMTP server supports. In our example, the server announces support for a single additional command, called SIZE.

```
S: 220 bbb.com mail server ready
C: EHLO mail.aaa.com
S: 250-bbb.com
S: 250 SIZE
C: MAIL FROM: <alice@aaa.com> SIZE=15000000
S: 552 Message size exceeds maximum message size
```

Figure A.4: ESMTP server rejects message as too large

With knowledge that the server supports the SIZE command, the client appends onto the MAIL FROM command an attribute declaring the approximate message size. Now the server can reject the message as too large, saving the client from sending data that would have eventually been rejected. This graceful rejection procedure conserves those bandwidth and processor resources that would have been consumed attempting to send an undeliverable message.

Appendix B

SMIL

Synchronized Multimedia Integration Language (SMIL) is an XML-based language used to specify how separate media elements are to be rendered with respect to a common timeline. These separate media elements include audio, video, images, text and animation. The language appears much like HTML, and is relatively easy for content authors to create when developing a multimedia presentation. SMIL is supported by the popular media players, such as those provided by RealNetworks and Microsoft.

Similar to the way a Web page is constructed from separate files referenced by a base HTML document, a multimedia presentation is constructed from the separate media elements referenced in a single SMIL document. SMIL allows the specification of where in the window visual artifacts are displayed, and during which time intervals. Likewise, SMIL allows the specification of when audio is playout relative to a presentation timeline.

All XML documents must contain a single top-level element, referred to as the root element. In SMIL, the root element is demarcated by a `<smil>` start tag, and a `</smil>` end tag. Within the root element there can be two other elements: a `head` element and a `body` element. The `head` element contains information related to the visual layout of the media, while the `body` element contains information related to the timing of events. The `head` element is optional.

Within the `head` element there is a `layout` element. In this element, the size and background color of the rendering window is specified. Regions within the window are defined with the `region` element. A region element must include an `id` attribute, so that elements appearing in the body can reference them. Each region defines a rectangular area within the player window.

Inside the `body` element, we define the times the various presentation components are rendered. This is done with two elements: the `seq` element,

and the `par` element. `seq` is an abbreviation for "sequence," and the elements within a `seq` element are played in sequence, that is, one after another. `par` is an abbreviation for "parallel," and the elements within a `par` element are played in parallel, that is, simultaneously. `seq` and `par` elements can be nested within each other to any level.

Fig. B.1 is a sample SMIL document that illustrates some aspects of the language. Notice that all elements are contained within the top=level `smil` element. Within the head element is a single `layout` element, which gives information about the size and position of the region that is identified by the name (id) "image." Inside the `body` element, there is a single `par` element, which contains two media elements. The first media element is an image, which is displayed in the "image" region. The second media element is an audio clip, which starts to play at the same time the image is displayed.

```
<smil>
<head>
   <layout>
      <region id="image"
              left="0" top="0" height="425" width="450"/>
   </layout>
</head>
<body>
   <par>
      <img src="photo.jpg" region="image"/>
      <audio src="photo_narration.ra"/>
   </par>
</body>
</smil>
```

Figure B.1: SMIL document

The World Wide Web Consortium (W3C) created and maintains the SMIL specification. A complete description of the specification can be found at their Web site.

Bibliography

[ARON] B. Arons., SpeechSkimmer: A system for interactively skimming recorded speech, ACM Transactions on Computer-Human Interaction, March 1997.

[BREW] E.A. Brewer, P. Gauthier, D. McEvoy. The Long-term Viability of Large-scale Caching. 3rd International Caching Workshop, Manchester, UK, 1998.

[CAND] K.S. Candan, B. Prabhakaran and V.S. Subrahmanian. Retrieval Schedules Based on Resource Availibility and Flexible Presentation Specifications. Multimedia Systems, July, 1998.

[CARR95] S. Carrier and N. Georganas. Practical Multimedia Electronic Mail on X.400. IEEE Multimedia, winter, 1995.

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

[DERN] D. Dern. Postage due on junk e-mail—Spam costs Internet millions every month. InternetWeek, 4 May 1998. http://www.techweb.com/se/directlink.cgi?INW19980504S0003

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

[FENG99] W. Feng. Video-on-Demand Services: Efficient Transportation and Decompression of Variable Bit Rate Video. Ph.D. Dissertation, April 1996.

[FLUC] F. Fluckiger. Understanding Networked Multimedia: Applications and Technology. Prentice-Hall, 1995.

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

[HTML] D. Raggett, A. Le Hors, I. Jacobs, eds., HTML 4.0 Specification, W3C Recommendation, 24 April 1998.

[HTTP] R. Fielding, et al., "Hypertext Transfer Protocol – HTTP/1.1," Network Working Group, RFC 2068, January 1997.

[IMAP] M. Crispin. Internet Message Access Protocol: Version 4rev1. Internet Engineering Task Force, RFC 2060, dec 1996.

[JMF] Sun Microsystems, Inc. Java Media Framework API, 15 Jul 1999.

[JPEG] Independent JPEG Group. JPEG software library, availble at http://www.jcu.edu.au/docs/jpeg.

[JPEG2] A. Skodras, C. Christopoulos, T. Ebrahimi. JPEG2000: The upcoming still image compression standard. Proceedings of the 11th Portuguese Conference on Pattern Recognition (RECPAD 2000), Porto, Portugal, May 2000.

[JPEGC] The JPEG Committee, formally known as *ISO/IEC JTC1 SC29 Working Group 1*, http://www.iso.org.

[KAUF] C. Kaufman, R. Perlman, and M. Speciner. Network Security: Private Communication in a Public World. Prentice Hall, 1995.

[LALI] D. LaLiberte, HyperNews, freeware software, available at http://www.hypernews.org/HyperNews/get/hypernews.html.

[MCMA96] J. McManus and K. Ross. Video on Demand over ATM: Constant-Rate Transmission and Transport. IEEE JSAC, Vol. 14, 1996.

[MCMA98] J. McManus and K. Ross. A Dynamic Programming Methodology for Managing Layered Encoded VBR Sources in Packet-Switched Networks. Telecommunications Systems, Vol. 9, 1998.

[MCPH98] E. McPhillips. The structure and trends of the ISP market. Hewlett-Packard Laboratories Technical Report HPL-IRI-1999-002, Jun 1998.

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

[MIME] 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.

[NNTP] B. Kantor, P. Lapsley, Network News Transfer Protocol, IETF, RFC 977, Feb 1986.

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

[PADH] J. Padhye, J. Kurose. An Empirical Study of Client Interactions with a Continuous-Media Courseware Server. Proceedings of NOSSDAV '98, Cambridge, UK, jul 1998.

[PAX97a] V. Paxson, S. Floyd. Why We Don't Know How To Simulate The Internet. Proceedings of the 1997 Winter Simulation Conference, Dec 1997.

[PAX97b] V. Paxson. End-to-End Internet Packet Dynamics. IEEE Transactions on Networking 7(3), pp. 277-292.

[RAMA] R. Ramanujan, J. Newhouse, M. Kaddoura, A. Ahamad, E. Chartier, K. Thurber. Adaptive streaming of MPEG video over IP networks. In Proceedings of the 22nd Annual Conference on Local Computer Networks, Nov 1997, Minneapolis, MN.

[REAL] RealNetworks, Inc. SureStream: Delivering superior quality and reliability. http://www.real.com/devzone/library/whitepapers/surestrm.html.

[REIS97] M. Reisslein and K. Ross. Join-the-Shortest-Queue Prefetching. ICNP, Atlanta, 1997.

[REIS98] M. Reisslein and K. Ross. High Performance Prefetching Protocols for VBR Prerecorded Video. IEEE Network Magazine, Nov/Dec, 1998.

[RESN97] S. Resnick. Heavy Tail Modeling and Teletraffic Data. The Annuals of Statistics, 25, 1805-1869.

[RFC1830] G. Vaudreuil. SMTP Service Extensions for Transmission of Large and Binary MIME Messages. Network Working Group, RFC 1830, Aug 1995.

[RFC2017] N. Freed, K. Moore and A. Cargille. Definition of the URL MIME External-Body Access-Type. Network Working Group, RFC 2017, Oct 1996.

[RFC2212] S. Shenker, C. Partridge, R. Guerin. Specification of Guaranteed Quality of Service. RFC 2212, IETF, Sep 1997.

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss. An Architecture for Differentiated Services. RFC 2475, IETF, Dec 1998.

[RFC2635] S. Hambridge and A. Lunde. Don't Spew: A Set of Guidelines for Mass Unsolicited Mailings and Postings (spam), Internet Engineering Task Force, RFC2635, Jun 1999.

[RFC822] D. Crocker. Standard for the Format of ARPA Internet Text Messages. Internet Engineering Task Force, RFC 822, aug 1982.

[RTSP] H. Schulzrinne, A. Rao and R. Lanphier. Real Time Streaming Protocol (RTSP). Internet Engineering Task Force, RFC 2326, apr 1998.

[SALE96] J. Salehi, Z. Zhang, J. Kurose and D. Towsley. Supporting Stored Video: Reduce Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. Proceedings of ACM SIGMETRICS, 1996.

[SALE98] J. Salehi, Z. Zhang, J. Kurose and D. Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. IEEE/ACM Transactions on Networking, August 1998.

[SALS] J. Salsman, "Form-based Device Input and Upload in HTML," Internet-draft submitted to the W3C HTML activity for forms, 20 March 1999. This is a work in progress.

[SAPA] D. Saparilla and K. Ross. Streaming Stored Continuous Media over Fair-Share Bandwidth. Eurecom technical report, Feb, 2000.

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

[SCHZ] H. Schulzrinne. A Comprehensive Multimedia Control Architecture for the Internet. Proceedings, International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), St. Louis, Missouri, may 1997.

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

[SMTP] J. Postel. Simple Mail Transfer Protocol. Internet Engineering Task Force, USC/Information Sciences Institute, RFC 821, aug 1982.

[SURE] RealNetworks, Inc. SureStream: Delivering superior quality and reliability. http://www.real.com/devzone/library/whitepapers/surestream.html.

[TURN99a] David A. Turner and Keith W. Ross. Asynchronous Audio Conferencing on the Web. In Advances in Intelligent Computing and Multimedia Systems, Baden-Baden, Germany, Aug 1999.

[TURN99b] D. Turner and K. Ross. Optimal Streaming of Synchronized Multimedia Presentations. In Proceedings of ACM Multimedia 99, Orlando, FL, Oct 1999.

[TURN00a] David A. Turner and Keith W. Ross. Continuous Media E-mail on the Internet: Infrastructure Inadequacies and a Sender-Side Solution. IEEE Network, Jul/Aug 2000.

[TURN00b] David A. Turner and Keith W. Ross. Optimal Streaming of Layer-Encoded Multimedia Presentations. Proceedings of the IEEE International Conference on Multimedia and Expo, New York, New York, Jul/Aug 2000.

[TURN01a] David A. Turner and Keith W. Ross. A Comprehensive Architecture for Continuous Media E-Mail on the Internet. IEEE MultiMedia, jun/jul 2001.

[TURN01b] David A. Turner and Keith W. Ross. Adaptive Streaming of Layer-Encoded Multimedia Presentations. Invited submission.

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

[WILL96] W. Willinger, V. Paxon and M. Taggu. Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, pages 27-53, Birkhauser, Boston, 1998.

[WILL98] W. Willinger and V. Paxon. Where Mathematics Meets the Internet. Notices of the American Mathematical Society, 45(8), pp. 961-970, sep 1998.

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

[ZHAO98] W. Zhao, M. Willebeek-LeMair and P. Tiwari. Malleable Multimedia Presentations: Adaptive Streaming Tradeoffs for Best-Quality Fast-Response Systems. Proc. of the 10th Tyrrhenian International Workshop on Digital Communications, Sep 1998.

[ZHAO99] W. Zhao, M. Willebeek-LeMair and P. Tiwari. Efficient Adaptive Media Scaling and Streaming of Layered Multimedia in Heterogeneous Environments. Proc. of the IEEE International Conference on Multimedia Computing and Systems, Jun 1999.