Streaming multimédia applications et protocoles

Mårten Wikforss <mwikfors@ens-lyon.fr>
Benoît Masson <bmasson@ens-lyon.fr>
Vivian Brégier <vbregier@ens-lyon.fr>

May 16, 2002

Contents

1 Introduction **3**

1.1 What is Multimedia Streaming? 3

1.1.1 3 Types of Streaming 3

1.1.2 The nature of the traffic 4

2 Streaming Media Files **5**

2.1 Top Layer First 5

2.2 TCP or UDP 7

3 Packet Loss, Delay and Jitter **9**

3.1 The Limitations of a Best-Effort Service 9

3.2 Recovery from Limitations in Best-Effort Service 10

3.2.1 Recovery from Packet Loss 10

3.2.2 Recovery from end-to-end delay 11

3.2.3 Recovery from delay jitter 11

4 Protocoles **13**

4.1 Protocoles 13

4.1.1 RTP (Real-time Transport Protocol) 13

4.1.2 RTCP (Real-Time Control Protocol) 17

4.1.3 RTSP (Real-Time Streaming Protocol) 23

5 Réservation de ressource **26**

5.1 Description 26

5.1.1 Mécanismes 27
5.1.2 récepteurs hétérogènes 27
5.1.3 Admission d'appel 29
5.2 Exemples 30
5.3 Interface 32

6 Conclusion **33**

Chapter 1

Introduction

In the last few years network applications for transmitting audio and video have increased in popularity. These applications also referred to as continuous media applications have totally different service requirements than ordinary network applications. Streaming is a technique specially tailored for transmitting media files. We are in this report going to explain how Streaming applications works in general and describe the lower layer protocols of which these applications functionality is based on.

1.1 What is Multimedia Streaming?

Multimedia Streaming is a pretty new technique that allows a user to download a media file on the same time as he is running the file on his computer. All media are streamable for example audio, video, images, text, 3D data. But true streaming refers to only time based media particularly audio and video. We are in this report going to focus on true streaming.

1.1.1 3 Types of Streaming

Streaming of stored media: The media is pre-recorded and stored on a server. This is the most common application. Examples could be a prerecorded video lesson in a network course or stored audio for playing as background music to a web page. Depending on what type of server the media file is stored on the functionality of fast-forwarding, rewinding, pause/resume could be offered.

Streaming of live media Works like ordinary live broadcasts except that the broadcast media is the Internet in this case. Examples could be live radio or live video/audio from important sport events. Does not support fast-forwarding of natural causes, but pause and rewind functionality could be supported. This type of streaming is more delay sensitive than streaming of stored media.

Real-Time Interactive Audio and Video Examples could be IP-telephony or video conferencing. This is also a type of streaming since the media in these applications is streamed. These applications are extremely delay sensitive. Imagine a telephone conversation with a long delay. Since another group have chosen the subject IP-telephony are we not going to explain how these types of applications work in our report. But since they are based on the same lower level protocols as ordinary streaming applications they will not be invisible in the report.

1.1.2 The nature of the traffic

Streaming is by nature much more sensitive for network delays than ordinary data-oriented applications like text/image Web, FTP, e-mail and DNS. Packets that arrive too late become totally useless. One could say that delay is streaming's greatest enemy. On the other hand is it not as sensitive to packet loss as ordinary applications. It does not matter so much if you lose some packets on the way as it would do for a file transfer. Streaming applications certainly differs from ordinary applications in these two senses.

Chapter 2

Streaming Media Files

The media files are compressed audio/video files and can be stored on ordinary Web servers or special Streaming servers. Upon request the server sends the compressed file towards the receiver through a socket. This can be either a TCP- or a UDP socket. Before the file is put onto the socket it is segmented and the segmented parts are encapsulated with special headers. The Real-Time Protocol (RTP) which we are going to dedicate a whole chapter defines a standard for these headers. The receiver is then able to render the file when he starts receiving the RTP packets, just second after the request. To render the compressed media file the receiver needs a media player. The most common today are RealPlayer, Windows Media Player and Quicktime. These also provides the functionality to pause/resume and fast forward the media. For this to run smoothly there is a protocol called Real-Time Streaming Protocol (RTSP) which we are going to explain later in this report.

2.1 Top Layer First

So what happens when you request a media file? Let's say you do this by clicking on a link to a media file in your web-browser. First of all the link does not link directly to the media file but instead to meta file. The meta file is downloaded to your browser application which examines it. The meta file includes information about the URL to the media file and type encoding of the media file. The browser then launches the media player that supports the given type of encoding and passes the body of the meta file to the media player. The media player then sets up a connection to the server at the given address in the meta file and sends an request for the media file. The connection type depends on the type of the server.

HTTP server If the file is stored on an ordinary web server the media player will send http request and a TCP connection will be established between the server and the receiver. This solution is not recommended by companies who sell streaming servers. They claim that the http protocol is not rich enough to allow for satisfactory user interaction with the server.

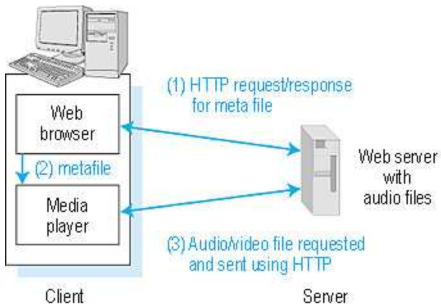

Figure 2.1: Media Stored on web server.

Streaming server If the file is stored on a streaming server then the media could be sent over UDP connection. The streaming server could be a separate machine or run on the same end system as the web server. This streaming server architecture gives many options for delivering the media to the media player.

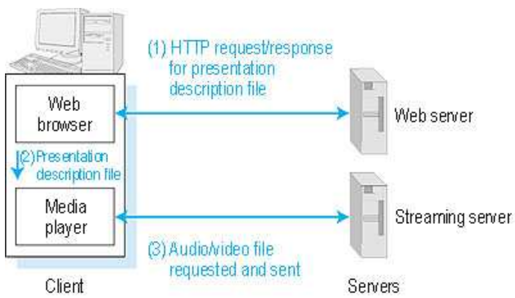

Figure 2.2: Media stored on streaming server.

1. The media is sent over UDP at a constant rate to the receiver. The rate is equal to the drain rate of the receiver that is the the encoded rate of the media. If for example the media is compressed at a rate of 13 Kbps the send

rate will also be 13 Kbps. The media is the decompressed and played as soon as it arrives at the receiver.

2. The first solution is extremely sensitive to network-introduced-jitter, to resolve this problem the client keeps a buffer. When he starts the UDP download the media which arrives first the gets buffered in a client buffer. The media player then starts to drain the client buffers prefetched media at a certain drain rate which should be equal to fill rate from the network see fig. 2.3. When we have network-jitter and packets are lost the fill rate is lower then the drain rate but the buffer protects us from these small changes.

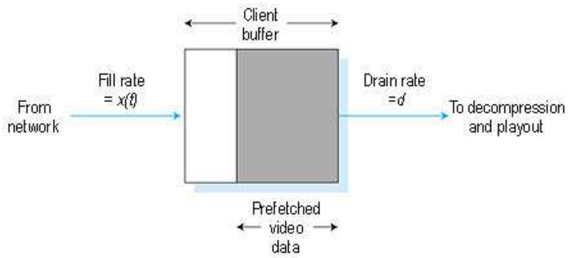

Figure 2.3: Application buffer

3. This solution is the same as the second with the exception that we use a TCP connection instead of an UDP.

2.2 TCP or UDP

TCP A TCP connection works in connected mode. This means that the client and the server communicate and exchange messages. When a packet gets lost the receiver notifies the sender and the packet gets retransmitted. By working in this way the sender knows that all packets has been received in the right order by the receiver. But it is useless to retransmit a packet which has already missed its play-out time which is very likely to happen if the packet gets lost. The sender and the receiver also exchange information about congestion in the network and throttle the transmission rate according to the congestion in the network. This is very harmful to streaming applications who are dependent on a constant transmission rate.

UDP A UDP connection is a much better solution for streaming media traffic. There is no communication between the client and the server. The sender can pump out data to the network at any rate he pleases. He does not care if they reach the receiver or not so no unnecessary retransmission has to be done. Data might get lost but this is not so important.

Chapter 3

Packet Loss, Delay and Jitter

3.1 The Limitations of a Best-Effort Service

Depending on which type of connection the RTP packets are sent over the RTP packets will be encapsulated in either TCP or UDP packets. The transport layer passes these packets down to the network layer where they get encapsulated in IP datagrams. The network layer then makes its best effort to deliver these datagrams to the receiver. But unfortunately is this not always enough, a lot can happen on the way.

Packet Loss As the IP datagrams wanders through the network it has to pass through many router queues in order to reach outbound links. If all queues are full in a router the IP datagram gets discarded by the router and is therefore lost. Packet loss is not so grave as one might think. Loss rates between 1 and 20 percent can be tolerated, depending on how the media is encoded.

End-to-End Delay Is caused by long queues in the routers. Delays by less than 150 msec are not perceived by a human listener. Delays between 150-400 msec are acceptable. A delay longer than 400 msec would seriously hinder interactivity in a voice conversation.

Delay Jitter Is caused by random queuing delays in routers. This means that the time from when a packet is generated at the sender to it is received by the receiver can vary. This is called jitter.

3.2 Recovery from Limitations in Best-Effort Service

The Streaming applications needs to recover from these limitations in the network layer of the Internet protocol stack.

3.2.1 Recovery from Packet Loss

There are a lot of ways to recover from packet loss without having to retransmit packets. To do this most streaming applications use some sort of loss anticipation scheme, the two most common are fast error correction (FEC) and interleaving.

FEC The basic idea of FEC is to add redundant information to the media. The media player is then able to use this redundant information to recover form packet loss and reconstruct missing information. One method for this is to add a redundant chunk after every n chunks. The redundant chunk contain the result after exclusive OR-ing all the original chunks. If a chunk gets lost then we could use the redundant chunk to reconstruct missing chunk. Another method used by many IP-telephone applications is to provide two media streams. One with a higher quality/bit-rate and one redundant with a lower quality/bit-rate. As soon as data from the high-bit-rate stream is missing the low-bit-rate stream will be used.

Interleaving Interleaving is a technique that spread out the information. This means that if a packet gets lost we lose a small amount of information in large part of the media stream instead of getting a big hole in the middle. Fig. 3.1 shows how this works. A negative side effect of interleaving is that it increase the delay. The media application needs to receive all packets before it can start play-out.

Receiver based repair of damaged media streams There are also ways for the streaming application to repair damaged media and recover from packet loss itself without using any anticipating scheme. Since media, in particular audio from a voice conversation, exhibits large amounts of short-term self similarity the streaming application can use data from previous packets to reconstruct lost packets. The simplest form is called packet repetition. With packet repetition the streaming application simply replaces

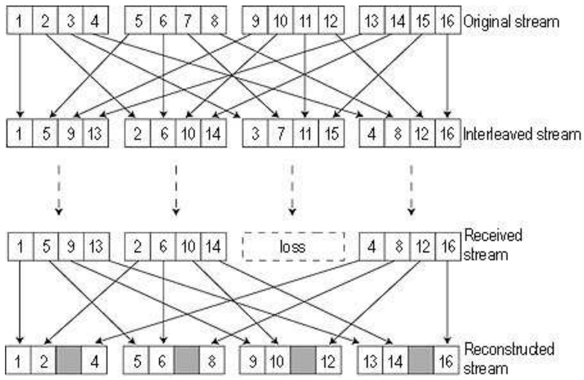

Figure 3.1: Example of interleaving

a lost packet by the packet received immediately before the loss. Another way is to interpolate. When interpolating the streaming application uses data from packets received before and after the missing packet to reconstruct it.

3.2.2 Recovery from end-to-end delay

This is unfortunately impossible.

3.2.3 Recovery from delay jitter

To deal with delay jitter the streaming application uses information in the RTP packet headers. All RTP packets are numbered and timestamped. The application can remove the jitter by delaying the play-out. There are two strategies for delaying play-out, fixed play-out delay and adaptive play-out delay.

Fixed play-out delay The application tries to play-out each chunk exactly q msecs after its generated by adding q to the timestamped value in the RTP header. With a high value of$q$ most packets gets received in time before play-out. But a high value of q also means a long delay. See fig. 3.2

Adaptive play-out delay The application tries to find an ideal play-out delay by using predefined algorithms. If you are for example using a IP-telephone application the play-out delay can change during a conversation.

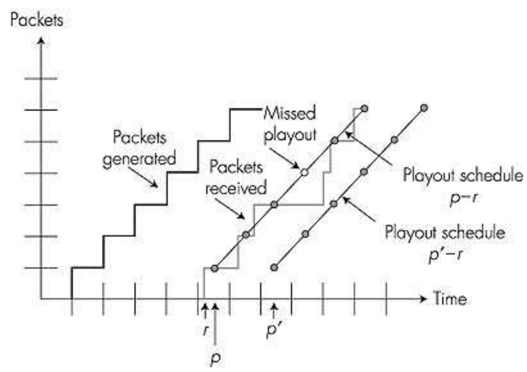

Figure 3.2: Packet loss for different values of q.

Chapter 4

Protocoles

4.1 Protocoles

4.1.1 RTP (Real-time Transport Protocol)

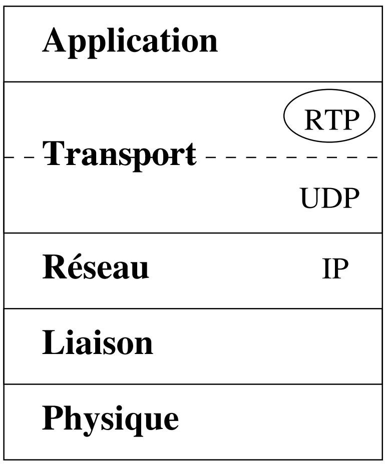

Figure 4.1: Position de RTP dans la structure en couches

C'est le principal protocole utilisé pour la transmission de données en temps réel. Il est basé sur IP, souvent situé juste au-dessus de UDP (fig. 4.1), et apporte plusieurs services tels que le réordonnancement des données dans le temps, la détection de pertes, l'identification du contenu. Il était prévu au début pour faire du multicasting (envoyer les données à plusieurs machines), mais peut très bien être utilisé entre uniquement 2 machines (par exemple pour le téléphone par Internet).

Il y en a eu plusieurs versions : en août 1991 naît le RTP version 0, appelé tout d'abord *vat*. La version 1 arrive en décembre 1992, puis la version finale, version 2, en novembre 1995. Le standard sera décrit dans la norme RFC 1889. Depuis les seules évolutions auront été des extensions à ce protocole (par exemple RTSP, décrit en 4.1.3).

Fonctionnement

RTP envoie les paquets par UDP, mais celui-ci ne conservant pas l'ordre (il est non-fiable), il va falloir les réordonner. Pour cela 2 systèmes sont utilisés : d'abord on aura un champ *temps* qui permettra au récepteur de savoir à quel instant se trouve le paquet (l'image si c'est une vidéo) et donc de pouvoir reconstruire le flux en maintenant l'écoulement du temps tel qu'il était au départ. Mais ceci ne suffit pas, en effet certaines trames peuvent nécessiter plusieurs paquets, on rajoute donc un champ *numéro de séquence* pour savoir dans quel ordre se trouvent les paquets (cela sera aussi utilisé pour la détection de paquets perdus).

Un autre champ permettra au récepteur de déterminer le format des données, ainsi que le type de compression utilisé. Celui-ci sera naturellement le même pour tout le paquet, mais peut varier en cours de transmission, notamment en cas de congestion il faudra utiliser un format plus compressé. Le format peut être soit un de ceux définis dans la norme RFC 1890, soit un format propre à l'utilisateur, auquel cas il faudra le définir. Il y a aussi un champ pour que le récepteur puisse savoir qui émet, ceci étant utile lorsqu'il y a plusieurs transmissions simultanées, on a besoin de savoir qui émet.

Enfin, RTP étant juste au-dessus de UDP, il se sert de ses services tels que le multiplexage et la détection d'erreur (sur la figure 4.2, on peut voir la structure d'un paquet RTP au niveau de la couche de liaision, avec les en-têtes ajoutés par UDP et IP).

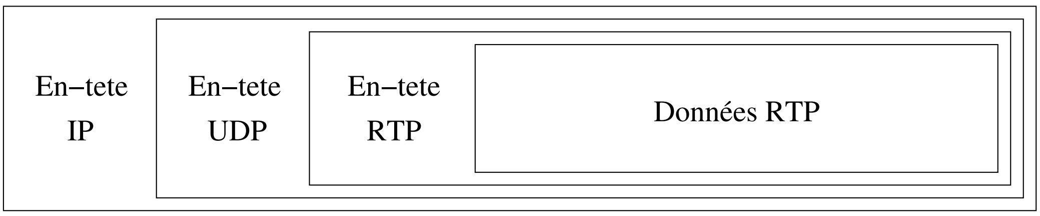

Figure 4.2: Paquet RTP transporté par UDP/IP

Cependant, bien que ceci soit l'utilisation classique, on peut aussi envoyer des paquets RTP sans passer par l'intermédiaire de UDP/IP, ce protocole étant indépendant de la couche transport. On pourra par exemple les faire passer sous TCP, IPX ou encore d'autres protocoles. Par contre, RTP étant souvent implanté dans l'application elle-même (en fait pour un développeur RTP se situe plutôt au niveau application), c'est elle qui devra gérer la récupération des paquets perdus et le contrôle de la congestion.

En pratique, comment cela se passe-t-il ? La session s'établit une fois que l'application a défini une paire d'adresses de destination, une pour l'adresse réseau et une paire de ports pour RTP et RTCP. Il faut autant de sessions qu'il y a de flux, par exemple dans une vidéo l'image et le son sont envoyés séparément. Ensuite l'application envoie des segments de données représentant une durée d, elle les découpe en paquets, les envoie au moyen de trames UDP, puis gère les problèmes de congestion au moyen de la compression utilisée. Le récepteur reçoit les paquets et les réordonne, pour finalement lire les segments à des intervalles de temps d.

En-tête d'un paquet RTP *(fig. 4.3)*

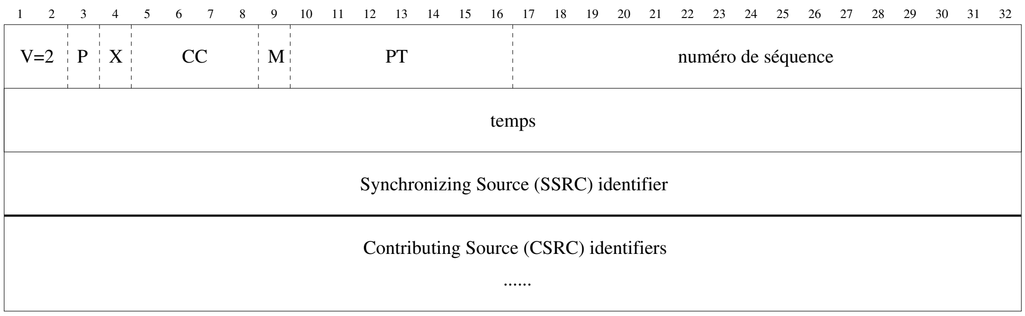

Figure 4.3: En-tête d'un paquet RTP

Les 3 premières lignes (96 bits) sont toujours présentes, la liste de CSRC ne l'est elle que lorsqu'elle a été ajoutée par un multiplexeur (voir partie 4.1.2).

- **V :** c'est la version, sur 2 bits. Aujourd'hui cela vaut 2.
- **P :** *padding*, sur un bit. S'il vaut 1, cela signifie qu'il y aura à la fin du paquet (après les CSRC et les données) des octets de remplissage, le dernier indiquant combien doivent être ignorés. Cela peut être utile lorsque les algorithmes de cryptage utilisent des blocs de taille fixe, ou pour transmettre plusieurs paquets RTP d'un coup en utilisant un protocole de plus bas niveau.

- **X :** si ce bit vaut 1, cela signifie que l'en-tête fixe est suivi d'exactement une extension (après la liste de CSRC, peu utilisé et peu recommandé).
- **CC :** compteur de CSRC sur 4 bits, pour indiquer la taille de la liste de CSRC à la fin (entre 0 et 15 donc).
- **M :** marqueur sur 1 bit qui signale la présence d'informations importantes dans le contenu du paquet (par exemple les limites de la trame).
- **PT :** le *payload type*, sur 7 bits, indique le format de la trame et comment elle doit être interprétée par l'application.
- **Numéro de séquence :** ces 16 bits servent à numéroter les paquets. La valeur augmente de 1 à chaque paquet, cependant elle ne part pas d'un nombre fixé. On part d'une valeur aléatoire imprévisible, pour diminuer les risques d'attaque (après cryptage, il devient alors très difficile de reconstruire le message). Le récepteur peut alors réordonner les paquets, et éventuellement détecter les paquets perdus.
- **Temps :** sur 32 bits, il indique le moment auquel le premier octet du paquet doit être joué. Il augmente en suivant une horloge qui évolue de manière linéaire, par conséquent il n'augmente pas forcément d'un à chaque fois, notamment lorsque le paquet contient un segment de données de durée assez longue. Là aussi on part d'une valeur aléatoire, et la progression est assez aléatoire : dans le cas de grosses trames, plusieurs paquets consécutifs vont avoir la même valeur de temps, dans le cas de trames MPEG interpolées par exemple la progression ne sera même pas monotone, puisque parfois on reviendra en arrière dans la vidéo (lorsqu'on interpole par rapport aux trames suivantes).
- **SSRC :** sur 32 bits, il permet d'identifier la source de synchronisation. Lors d'une session RTP entre plusieurs sources, toutes auront un SSRC différent. Cependant comme il sera choisi de manière aléatoire, il existe une (très faible) probabilité que plusieurs aient la même valeur, il faut alors pouvoir détecter et résoudre ceci.
- **CSRC :** liste de 0 à 15 éléments sur 32 bits chacun, pour identifier les sources après passage dans un multiplexeur. En effet dans ce cas le SSRC sera celui du multiplexeur, et les CSRC restent le seul moyen de référencer les sources d'origine.

4.1.2 RTCP (Real-Time Control Protocol)

Il est visible que toutes ces possibilités ne suffisent pas à effectuer un transfert efficace. C'est pourquoi RTP est accompagné d'un protocole de contrôle, RTCP, défini dans le même standard. Celui-ci consiste à envoyer périodiquement des paquets RTCP qui donneront des informations sur la communication en cours, par exemple le niveau de congestion du lien, et aussi qui est la source (pour détecter les conflits d'adresses SSRC mentionnés plus haut). Les 2 dernières fonctionnalités sont la gestion du nombre de participants, qui déterminera la fréquence des paquets RTCP (plus faible si la communication concerne beaucoup de monde), et pour finir c'est un moyen d'envoyer de petites informations à tous les postes concernés par la communication. Seules les 3 premières fonctionnalités sont obligatoires, et uniquement dans le cas de multicasting (mais recommandées sinon), la $4^{\text{ème}}$ est elle facultative.

Principe de fonctionnement

Il existe 5 types de paquets RTCP :

- **RR** *(Receiver Report) :* Ces paquets sont utilisés par les participants qui ne sont pas en train d'envoyer des données. Ils donnent des informations sur la qualité de réception (dernier paquet reçu, nombre de paquets perdus, délai entre arrivées, et un index temporel pour calculer le temps d'un aller-retour).
- **SR** *(Sender Report) :* Ceux-ci sont au contraire envoyés par les participants actifs. Ils informent sur la qualité du transport un peu à la manière des paquets RR, mais contiennent aussi des champs d'information sur l'émetteur (synchronisation des flux, compteur de paquets, nombre d'octets envoyés).
- **SDES** *(Source Description) :* Décrivent la source (avec notamment le *CNAME*, nom canonique qui sera unique et permettra de détecter les conflits de SSRC.
- **BYE :** Comme son nom l'indique, permet de signaler que l'on ne participe plus à la communication.
- **APP :** Fonctionnalités spécifiques à l'application. Ce type est réservé à l'expérimentation pour tester de nouvelles possibilités qui pourraient être ajoutées.

Généralement les paquets RTCP sont regroupés et envoyés dans des paquets qui contiennent au moins 2 informations. La première est nécessairement un paquet RR ou SR (il peut y avoir plusieurs RR lorsqu'il y a trop de sources), l'autre un SDES. Ensuite le reste est libre, on peut éventuellement avoir d'autres paquets de tous types (cependant un paquet BYE doit être le dernier). A noter que ceci est précédé d'un préfixe aléatoire lorsque le transfert sera crypté, pour éviter là encore le piratage.

Structure des paquets SR et RR

Un paquet SR est représenté sur la figure 4.4.

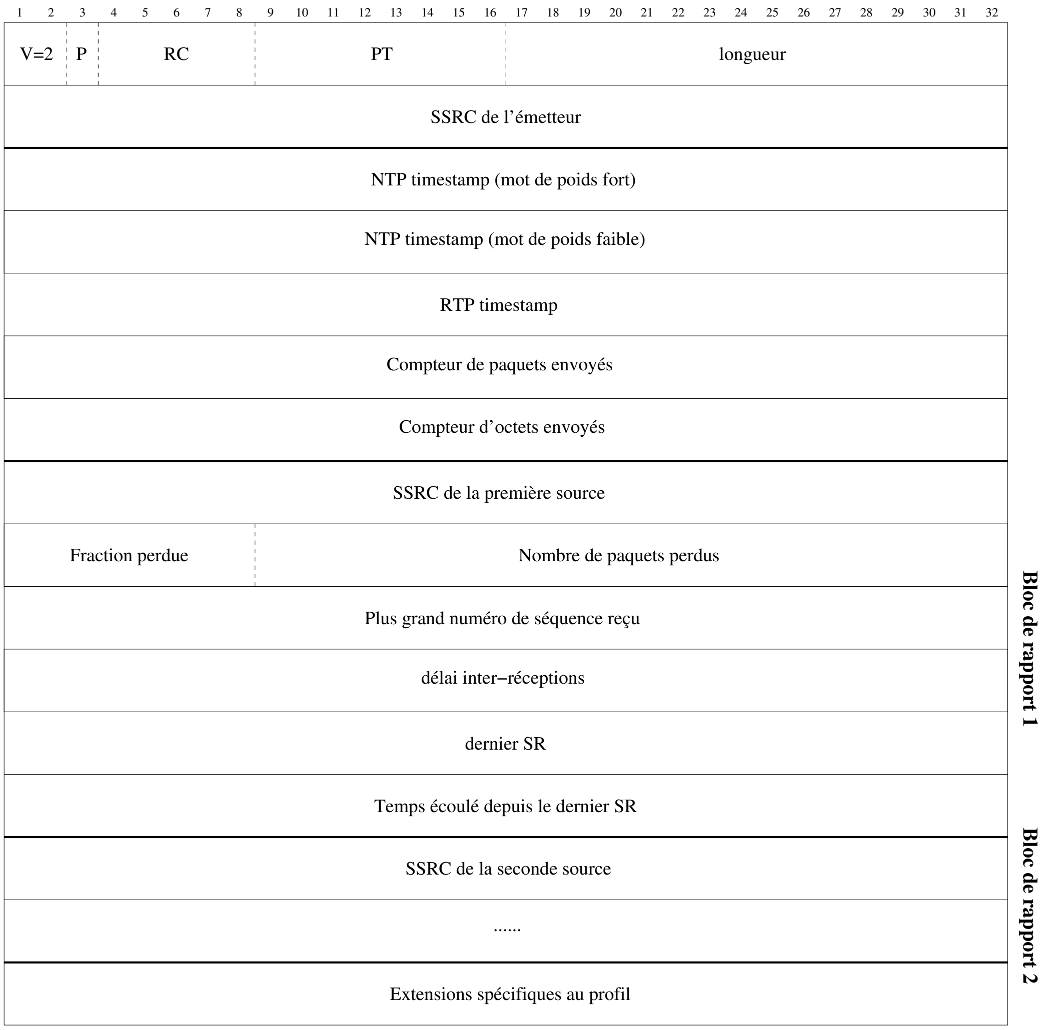

Figure 4.4: Structure d'un paquet SR

Les champs **V** et **P** sont identiques à ceux de l'en-tête de RTP. **RC** permet de donner le nombre de blocs de rapports inclus dans ce paquet (il peut y en avoir 0). **PT** *(Packet Type)* contient la valeur 200, et permet d'identifier le paquet comme étant un paquet SR de RTCP. Le champ **longueur** indique le nombre de mots contenus dans le paquet, **SSRC** l'identificateur de l'émetteur.

Ensuite la $2^{\text{ème}}$ partie du paquet contient des informations sur l'émetteur : **NTP timestamp** indique le moment où le paquet a été envoyé, pour calculer le temps d'un aller-retour, ce temps étant le même pour tout le monde. **RTP timestamp** correspond lui aussi au moment d'émission du paquet, mais sur la même échelle de temps que celle vue pour les paquets RTP. Ensuite le compteur de paquets indique le nombre de paquets envoyés depuis le début de la transmission (remis à 0 si l'émetteur change de SSRC), et le compteur d'octets indique le nombre d'octets utiles (hors en-tête et remplissage) envoyés depuis le début de la transmission.

Puis viennent les blocs de rapport qui permettent de renseigner sur la qualité de la réception, un bloc par source émettrice. On a d'abord le **SSRC** de la source en question, puis la **fraction** du nombre de paquets perdus depuis le dernier SR (réel à virgule fixe compris entre 0 et 1), et le nombre de paquets perdus depuis le début de la réception (cumulé sur tous les SR). Ensuite on indique le plus grand numéro de paquet reçu, et la variation moyenne de l'intervalle entre la réception de plusieurs paquets (en unités de *timestamp*). Les derniers champs indiquent le *NTP timestamp* du dernier paquet SR reçu depuis cette source, ainsi que le temps en $65536^{\text{ème}}$ de secondes qui s'est écoulé depuis la réception de celui-ci. Ceci permettra à la source d'estimer le temps nécessaire à un aller-retour entre les deux postes.

Il y aura autant de blocs de rapport que de sources (s'il y a trop de sources, on a vu qu'il pouvait être nécessaire de séparer en plusieurs paquets), puis un dernier champ spécifique au profil utilisé lorsque l'on doit mentionner d'autres informations assez petites pour tenir ici (cela évite l'envoi d'un autre paquet, encombrant pour rien).

Un paquet RR a exactement la même structure, sans la deuxième partie (le **PT** vaut cette fois-ci 201).

Structure d'un paquet SDES

Un paquet SDES a la structure représentée sur la figure 4.5.

La première partie contient les champs habituels, avec un **PT** de 202, **SC** *(Source Count)* indiquant le nombre de blocs, 0 étant valide mais inutile.

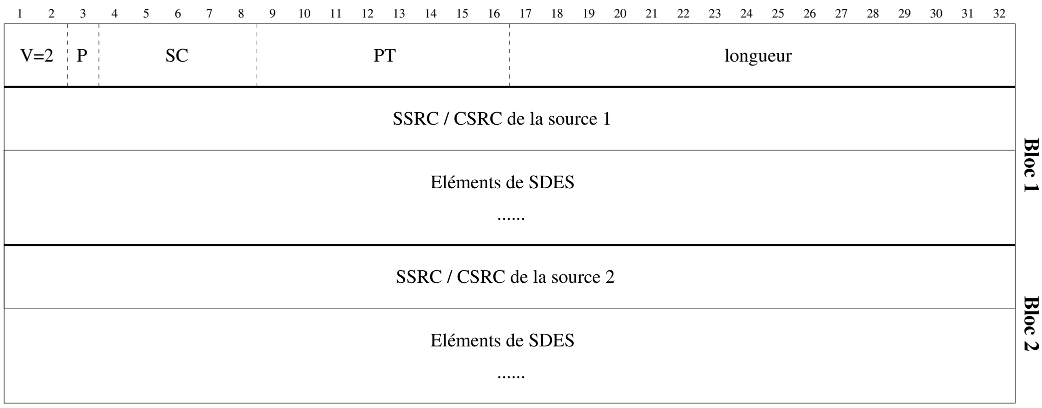

Figure 4.5: Structure d'un paquet SDES

Ensuite on trouve des blocs pour chaque source, avec les identificateurs SSRC et CSRC, suivis d'une liste d'informations sur la source : chaque élément comporte 8 bits pour indiquer le type, puis 8 bits pour indiquer la taille du texte (qui ne peut donc dépasser 255 octets), puis le texte. Un seul élément est obligatoire, c'est le *CNAME*, mais on peut aussi ajouter des données telles que le nom, une adresse e-mail, un numéro de téléphone, etc...

Structure d'un paquet BYE

Un paquet BYE est représenté sur la figure 4.6.

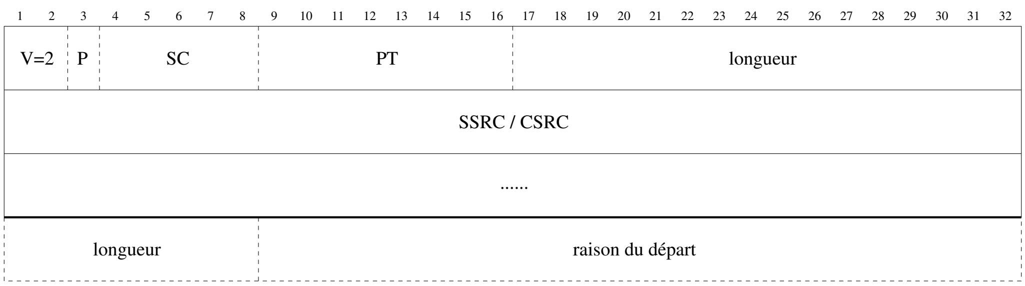

Figure 4.6: Structure d'un paquet BYE

Il n'y a qu'une partie obligatoire, comportant les champs habituels : version, *padding*, type (203 ici), et nombre de sources référencées dans ce paquet. Ensuite suit une liste (éventuellement vide, mais c'est alors inutile) de SSRC et CSRC, qui servira à un multiplexeur pour faire suivre l'information à toutes les sources concernées.

La seconde partie, facultative, permet d'indiquer la raison du départ. Un premier champ indique la longueur sur 8 bits, puis on trouve du texte pour indiquer la raison (surtout utile en cas de mauvais fonctionnement) sur 0 à 255 octets.

Structure d'un paquet APP

On a vu précédemment que ces paquets n'étaient généralement pas utilisés, il ne le sont que pour tester de nouvelles fonctionnalités.

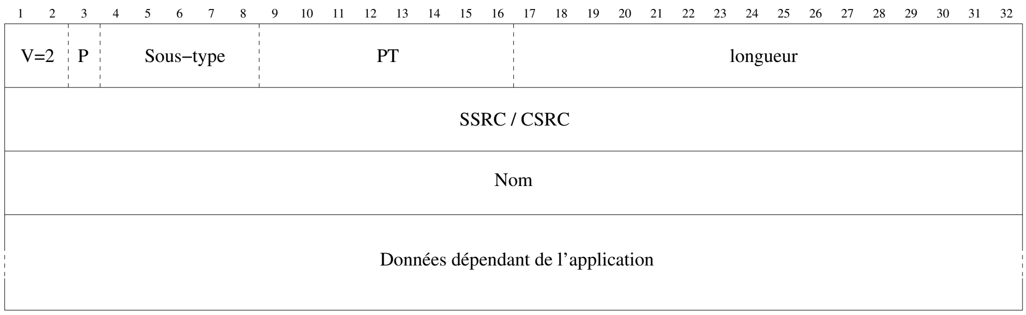

Figure 4.7: Structure d'un paquet APP

Comme indiqué sur la figure 4.7, ces paquets sont relativement simples. A l'en-tête habituel (**PT** = 204) s'ajoute un champ **sous-type**, utile lorsqu'il faut différencier des paquets ayant le même nom (champ **nom**). Lorsque l'application reçoit un paquet APP dont elle ne reconnaît pas le type, elle l'ignore. Ensuite vient un champ de longueur variable, permettant d'envoyer les données voulues.

Traducteurs et multiplexeurs

On va parfois avoir besoin de relier entre eux plusieurs réseaux qui ne peuvent être accédés directement depuis l'extérieur. Cette connection entre *nuages* se fera au moyen de 2 outils, les traducteurs et les multiplexeurs[1] . Ceux-ci reçoivent des flux de données, et après traitement les redirigent vers les destinataires initiaux. Ils peuvent effectuer plusieurs opérations sur ces paquets, par exemple changer le format de compression, crypter les données, changer le codage pour utiliser un autre protocole de transport, etc... La différence entre traducteur et multiplexeur est la suivante : un traducteur se

[1] *Mixer* en anglais, mais les traductions par mélangeur ou mixeur ne conviendraient pas vraiment. Multiplexeur est celui qui s'en approcherait le plus.

contente de convertir les flux séparément, alors qu'un multiplexeur va aussi les regrouper dans un seul paquet RTP.

Traducteur : il ne sait que faire suivre les données, éventuellement après avoir changé leur format. Il est donc indétectable par un récepteur qui n'aurait pas d'information quant au format initial des données.

Pour les paquets RTCP, cela marche de manière similaire. Les informations d'envoi (seconde partie de SR) sont maintenues telles quelles, aux conversions près, puis envoyées au destinataire prévu. De même les autres types de paquets subissent le même traitement (éventuellement conversions puis envoi), à part les paquets SDES dans lesquels des informations peu utiles peuvent être enlevées dans le cas de faible bande passante.

Multiplexeur : Il fait tout ce que fait le traducteur (conversions, filtrage, ...), avec en plus la possibilité de mélanger les flux provenant de plusieurs sources dans un seul flux. C'est là qu'interviennent les identificateurs CSRC, qui servent à référencer les sources d'origine. En effet le champ SSRC contient maintenant l'identificateur du multiplexeur, qui va donc recevoir tous les paquets avant de les faire suivre aux sources concernées. Le fait de connaître les CSRC permettra aux destinataires de savoir que ces sources existent, et au multiplexeur de leur faire suivre les données.

Les paquets RTCP sont traités différemment : les informations d'envoi dans les paquets SR reçus sont détruites, et de nouvelles sont générées mais concernant cette fois-ci le multiplexeur. En effet les informations d'envoi sont rendues inutiles par le mélange des flux, puisque l'on perd ses caractéristiques propres. Les informations de réception reçues des autres nuages sont prolongées à tous les membres du nuage du multiplexeur, mais toutes les autres sont détruites. Quant aux paquets SDES, comme pour les traducteurs certaines informations peuvent être supprimées, mais le principal est gardé et il le fait suivre. Les paquets BYE sont prolongés, et le multiplexeur en envoie un lorsqu'il veut arrêter. Pour les paquets APP, cela dépend de l'application et de l'information contenue dans le paquet.

Résumé des possibilités

- **Qualité de service :** On a vu que les paquets SR et RR renseignaient sur le niveau de congestion du réseau, ce qui permet d'optimiser son utilisation. On peut aussi détecter la zone concernée par la congestion, et donc traiter individuellement les cas pour du multicasting.

- **Identification de la source :** Les source sont identifiées non seulement par des numéros uniques, mais aussi par une description plus précise compréhensible par un être humain.
- **Synchronisation :** Comme on ne donne pas uniquement le temps auquel doit être jouée une trame, mais aussi un temps "universel", on peut ainsi synchroniser facilement plusieurs flux.
- **Contrôle du trafic :** On sait en permanence combien de sources participent à la communication, on en tient compte pour que seuls 5% des paquets envoyés lors de celle-ci soient des paquets RTCP. En fait la période d'envoi des paquets RTCP est une fonction linéaire en le nombre de participants, dont les coefficients dépendent des données transférées (taille, quantité, etc...). Cela permet d'éviter de provoquer des congestions tout en maintenant l'échange d'informations entre les participants.

Mais il ne faut pas oublier que RTP/RTCP ne représente qu'une petite partie de la transmission, il faut ensuite que d'autres couches fassent le reste. L'application devra gérer tout ce qui est congestion, pertes de paquets (RTP ne fait que les détecter) et synchronisation, le protocole de transport doit envoyer les paquets RTP en utilisant un système de trames pour pouvoir ensuite isoler les différentes parties. Par contre il est évolutif, on peut rajouter de nouveaux types de trames, ce qui permettra de le conserver dans les applications futures.

4.1.3 RTSP (Real-Time Streaming Protocol)

RTSP est un protocole de haut niveau, dans la couche application, qui permet de contrôler l'envoi des données. Par exemple le poste récepteur pourra mettre le flux en pause, le relancer, effectuer une avance rapide ou même se positionner à un endroit quelconque du flux.

Il repose sur RTP/RTCP, mais il peut décider quel moyen de transport utiliser (UDP ou autre), et sous quel format. Il a été défini en 1998 dans la norme RFC 2326.

Fonctionnement

Les 3 opérations de base sont les suivantes : le client peut demander au serveur d'établir une session par laquelle il recevra les données, une machine peut aussi en inviter une autre à s'y joindre, et enfin on peut avertir les autres participants qu'une autre présentation est disponible.

Chaque flux multimédia est identifié par une adresse RTSP, un peu comme une page web. Un fichier descriptif contenant les informations de base relatives à la présentation est accessible comme un fichier normal (http, e-mail, etc...), il permet alors de lancer une session connaissant l'adresse du flux. C'est un protocole “interactif”, grâce auquel le client et le serveur peuvent demander à changer certaines caractéristiques de la transmission. Actuellement, les services suivants sont possibles :

- **OPTIONS :** le client (ou le serveur) décrit à l'autre machine les options qu'il peut accepter.
- **DESCRIBE :** le client demande la description de la présentation identifiée par une adresse.
- **ANNOUNCE :** permet au client de demander au serveur l'envoi de la description, au serveur de mettre à jour la description de la session en temps réel.
- **SETUP :** le client demande au serveur d'allouer les ressources nécessaires à la transmission et de lancer une session RTSP.
- **PLAY :** le client commande le début du transfert établi par **SETUP**.
- **PAUSE :** le client stoppe temporairement le flux sans libérer les ressources.
- **TEARDOWN :** Le client demande à stopper le transfert et libérer les ressources.
- **GET_PARAMETER :** permet de récupérer la valeur d'un paramètre d'une présentation à partir de son adresse.
- **SET_PARAMETER :** donne une valeur à un paramètre d'une présentation à partir de son adresse.
- **REDIRECT :** le serveur prévient le client qu'il doit se connecter à un autre serveur. Il fournit en même temps l'adresse alternative à laquelle le client devra se connecter.

- **RECORD :** le client démarre l'enregistrement d'une portion de la présentation.

Ces messages sont envoyés sur un canal indépendant du canal de données, avec n'importe quel protocole de transport. En fait RTSP fonctionne comme une "télécommande" de flux, qui permet de commander à distance l'envoi des paquets RTP, à partir d'une adresse caractérisant le flux. Ce protocole est relativement standardisé, comme RTP/RTCP, et facilite donc les échanges entre machines différentes ne fonctionnant pas forcément sous les mêmes systèmes.

Chapter 5

Réservation de ressource

Comme on l'a vu, les flux multimédia sont très dépendants des performances du réseau sur lequel ils sont transmis. Une congestion réseau, même de quelques instants, aura une répercussion : dès qu'elle dure plus longtemps que la taille du tampon, la lecture de la vidéo regardée ou de l'audio écouté est stoppée. Or il est impossible de savoir, même si le flux est transmis par une liaison de bien plus haut débit que nécessaire, si personne ne va le saturer.

Pourtant les flux multimédia ont un débit déterminé et constant. Il serait donc pratique de pouvoir réserver une partie de la bande passante des liens entre la source et le récepteur, de manière à être sur que personne l'extérieur ne puisse interrompre cette transmission.

De plus, il est fréquent, lors de flux multimédia diffusés en direct, que plusieurs récepteurs regardent la même vidéo en même temps. Il est de plus quasiment certain que les deux flux ont une partie de leur chemin commune. Il serait intéressant que l'émetteur n'envoie qu'un seul flux qui se sépare lorsque les deux chemins divergent, économisant ainsi de la bande passante sur la partie commune des chemins.

Ce sont ces deux points que se charge de faire le protocole RSVP (Ressource ReSerVation Protocol).

5.1 Description

RSVP a été développé par PARC (Xerox Corp's Palo Alto Research Center), MIT (Massachusetts Institute of Technology) et ISI (Information Sciences

Institute of University of California)

RSVP sert à mettre en place des réservations de ressources réseau. Quand une application dans un hôte (comme le récepteur du flux de données) requière une qualité de service spécifique (Qos, Quality Of Service) pour son flux de données, elle utilise RSVP pour transmettre la requête aux routeurs le long du chemin du flux de données. RSVP est responsable de la négociation des paramètres de connexion avec ces routeurs. Si la réservation est établie, RSVP est aussi responsable de maintenir l'état des routeurs et de l'hôte pour fournir le service requis.

5.1.1 Mécanismes

Le protocole RSVP permet aux applications de réserver de la bande passante pour leurs flux de données. Il est utilisé par l'hôte pour demander une certaine quantité définie de bande passante du réseau. RSVP est aussi utilisé par les routeurs pour forwarder la requête de réservation de bande passante.

Comme le montre la figure 5.1, les deux principales caractéristiques de RSVP sont :

- il permet la réservation de bande passante pour les arbres multicast
- il est orienté récepteur

Les données viennent du haut et descendent vers les récepteurs en bas, alors que les messages de réservation eux viennent des récepteurs. Quand un routeur forwarde un message de réservation vers l'émetteur en amont du flux, il peut le fusionner avec un autre message de réservation arrivant de l'aval du flux.

5.1.2 récepteurs hétérogènes

Les récepteurs ne reçoivent pas forcément tous a la même vitesse, certains bénéficiant d'une mauvaise connexion ne pouvant pas recevoir a plus de 28,8 Kbps, alors que d'autres pourront recevoir a 128 Kbps, 10 Mbps ou même plus...

Ceci pose une question intéressante : Si un émetteur fait du multicast sur un groupe de récepteurs hétérogènes, doit-il envoyer la vidéo en mauvaise qualité à 28,8 Kbps, en qualité moyenne à 128 Kbps ou en bonne qualité à 10 Mbps ? Si la vidéo est encodée à 10 Mbps, seul l'utilisateur possédant

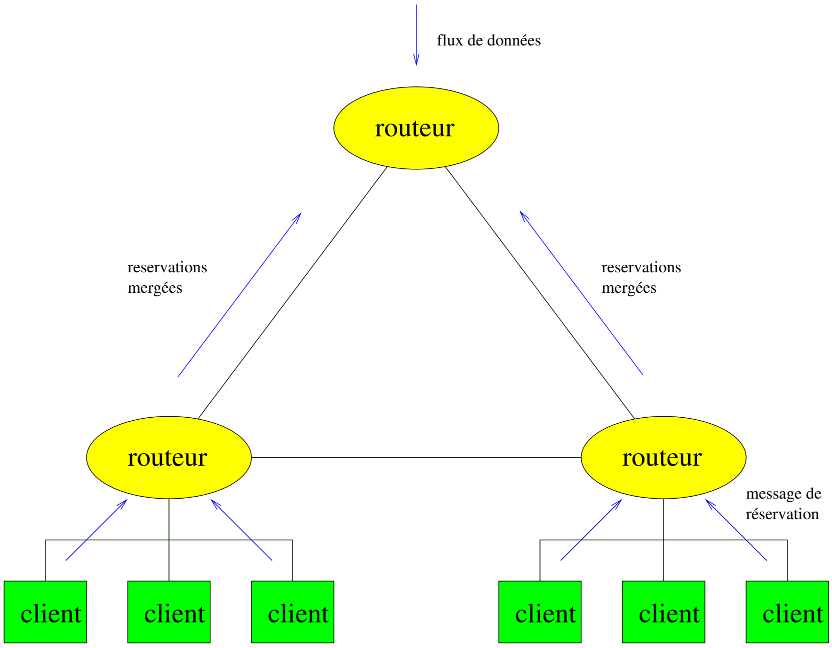

Figure 5.1: RSVP : multicast et orienté récepteur

une connexion 10 Mbps pourra la regarder, mais si la vidéo est encodée à 28,8 kbps, il devra regarder une vidéo de mauvaise qualité alors qu'il sait qu'il peut recevoir bien mieux...

Pour résoudre ce problème, il est souvent suggéré d'encoder les données audio et vidéo en couches. Par exemple, une vidéo peut être encodée en 2 couches, une couche de base et une couche améliorée. La couche de base peut par exemple avoir une cadence de 20 Kbps alors que la couche améliorée serait à 100 Kbps. Ainsi, l'utilisateur ayant une connexion de 28,8 Kbps peut recevoir la couche de base et celui ayant une connexion de 128 Kbps peut recevoir les 2 couches pour construire une image de bonne qualité.

On peut noter que l'émetteur n'a pas besoin de connaitre la vitesse de réception de tous les récepteurs, mais seulement la vitesse maximale du groupe de récepteurs. L'émetteur encode l'audio et la vidéo dans des couches multiples et envoie toutes les couches jusqu'à la vitesse maximale dans l'arbre de multicast. Les récepteurs choisissent les couches appropriées à leur vitesse de réception. Pour ne pas gâcher de bande passante dans les liens du réseau, les récepteurs communiquent au réseau les vitesses qu'ils peuvent supporter.

5.1.3 Admission d'appel

La quantité totale de bande passante réservée sur un lien ne peut pas excéder la bande passante totale du lien. Ainsi, quand un routeur reçoit une demande de réservation de bande passante, il doit d'abord déterminer si le lien en question peut supporter la réservation en question. Ce test d'admission est effectué à chaque fois qu'un routeur reçoit un message de réservation. Si ce test échoue, le routeur rejette la réservation et renvoie un message d'erreur au récepteur.

RSVP ne définit pas le test d'admission, mais suppose que le routeur effectue un tel test à chaque demande de réservation de bande passante.

Ce test d'admission pose bien des problèmes, en particulier a cause de la capacité de RSVP de fusionner des flux. En effet, une requête rejetée peut en fait provenir d'un certain nombre de requêtes qui ont été fusionnées, l'erreur de réservation doit alors être envoyée à tous les récepteurs responsables. Mais fusionner des requêtes hétérogènes crée une difficulté potentielle : le problème de "killer reservation", dans lequel une requête peut interdire le service à une autre. Il y a en fait deux problèmes de "killer-reservation :

1. le premier arrive quand il y a deja une réservation Q_0 en place. Si un autre récepteur fait une réservation plus importante $Q_1 > Q_0$, le résultat de Q_0 fusionnée avec Q_1 est susceptible d'être rejeté par le test d'admission par un noeud en amont. Ceci ne doit pas rejeter Q_0.

   La solution à ce problème est simple : quand un test d'admission échoue pour une requête d'admission, les réservations existantes restent en place.

2. le second est le contraire : le récepteur qui fait la réservation Q_1 est persistant, malgré le fait que son test d'admission échoue pour Q_1 sur un noeud (messages de réservation envoyés en continu). Cela ne doit pas empêcher un autre récepteur d'établir une réservation plus petite Q_0 qui serait acceptée si elle n'était pas fusionnée à Q_1.

   Pour résoudre ce problème, un message d'erreur `ResvErr` établit un état "blocked state" dans chaque noeud par lequel il passe. Les états bloqués dans un noeud modifient la procédure de fusion pour ne pas tenir compte du flux coupable (ici Q_1), ce qui permet au flux plus petit Q_0 d'être accepté.

5.2 Exemples

Supposons qu'il y a une source qui transmet la vidéo d'un événement sportif majeur sur Internet. Une adresse multicast a été assignée à cette session, et la source étiquette tous ses paquets sortants avec cette adresse multicast. On suppose aussi qu'un protocole de routage multicast a établi un arbre multicast de l'émetteur à quatre récepteurs comme indiqué figure 5.2.

Chaque récepteur envoie un message de réservation en amont vers l'émetteur dans d'arbres multicast. Ce message de réservation spécifie la vitesse a laquelle le récepteur voudrait recevoir les données de la source. Quand le message de réservation atteint un routeur, le routeur ajuste son ordonnanceur de paquets pour satisfaire la réservation. Puis il envoie une réservation en amont.

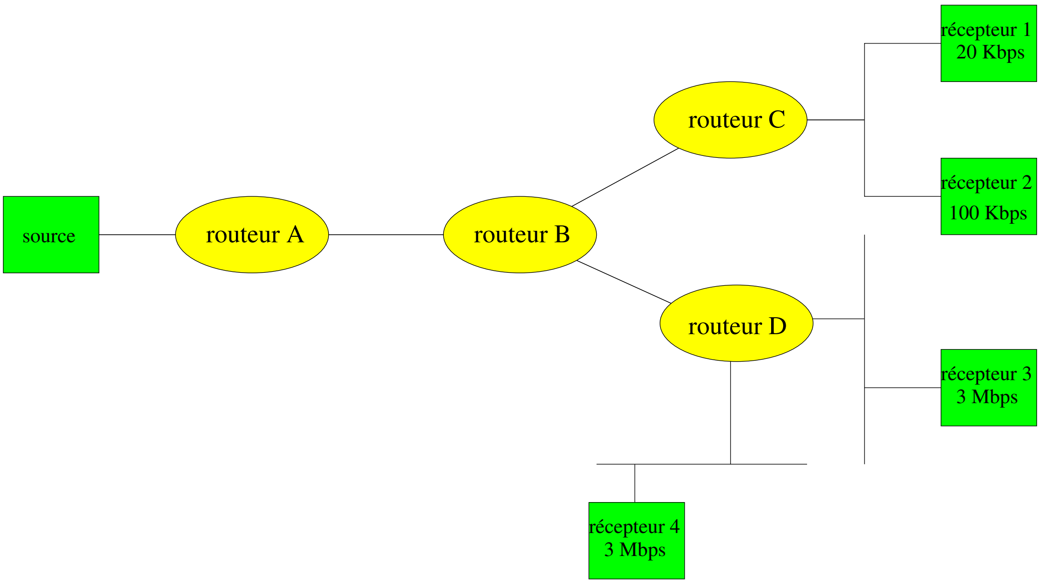

Figure 5.2: exemple de multicast RSVP

La quantité de bande passante réservée en amont par le routeur dépend des bandes passantes réservées en aval. Dans l'exemple figure 5.2, les récepteurs R1, R2, R3 et R4 réservent respectivement 20 Kbps, 100 Kbps, 3 Mbps et 3 Mbps. Ainsi, le flux aval maximal du routeur D est de 3 Mbps, donc il envoie un message de réservation au routeur B demandant a ce que celui-ci réserve 3 Mbps sur le lien qui les relie. On note que seuls 3 Mbps sont réservés et non $3+3 = 6$ Mbps ; en effet R3 et R4 regardent le même événement sportif, donc leurs réservations peuvent être fusionnées. De même, le routeur C demande au routeur B de réserver 100 Kbps sur le lien qui relie les routeurs B et C;

l'encodage multicouche garantit que le flux de 20 Kbps du récepteur R1 est inclus dans le flux de 100 Kbps. Une fois que le routeur B reçoit les messages de réservation des routeurs en aval, il envoie une nouvelle réservation à son routeur amont, le routeur A. Ce message réserve 3 Mbps de bande passante au lien entre le routeur A et le routeur B, ce qui est une fois de plus le maximum des réservations aval.

On voit sur cet exemple que RSVP est orienté récepteur, c'est-à-dire que c'est le récepteur d'un flux de données qui initie et maintient la réservation des ressources nécessaires pour ce flux. On note que chaque routeur reçoit un message de réservation de chacun de ses liens aval dans l'arbre multicast et envoie un seul message de réservation sur son lien amont.

Autre exemple, supposons que quatre personnes participent a une visioconférance, comme le montre la figure 5.3. Chaque personne a trois fenêtres ouvertes sur son bureau, pour voir les trois autres personnes. Le protocole de routage multicast a établi l'arbre multicast entre les quatre hôtes selon la figure 5.3. De plus chacun veut voir les vidéos à 3 Mbps.

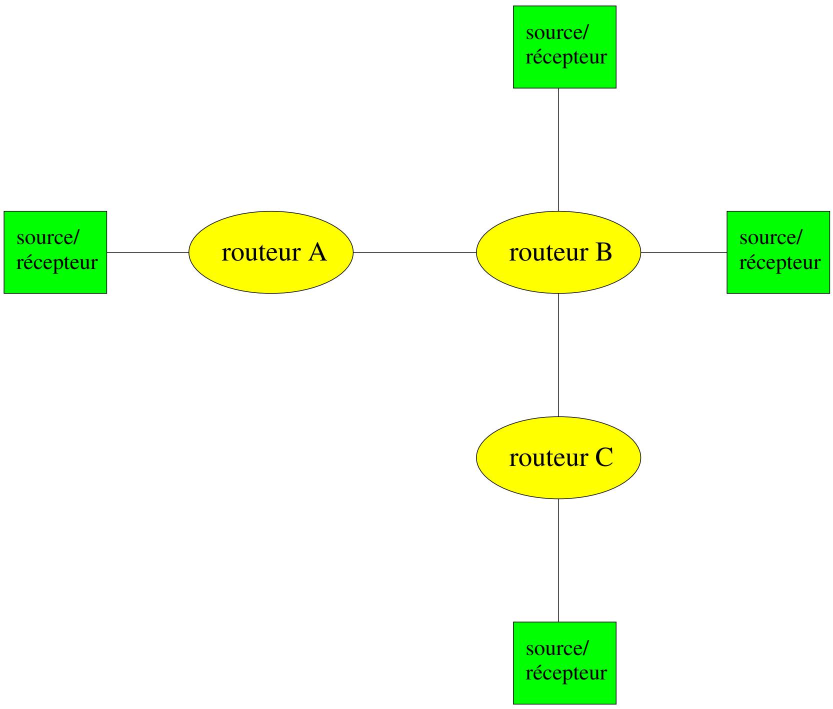

Figure 5.3: exemple de visioconférance RSVP

Sur chaque lien de cet arbre multicast, RSVP réserve $3 * 3 = 9$ Mbps dans une direction et 3 Mbps dans l'autre. On note que RSVP ne fusionne pas les réservations dans cet exemple étant donné que chaque personne désire recevoir trois flux distincts.

Si on considère maintenant une conférence uniquement audio entre les mêmes personnes, avec b bps requis pour un flux audio. Comme dans une conférence audio il est rare que plus de deux personnes parlent en même temps, il n'est pas nécessaire de réserver $3b$ bps pour chaque récepteur; $2b$ bps devraient suffire. Ainsi, dans cette dernière application on peut conserver la bande passante en fusionnant les réservations.

5.3 Interface

RSVP communique avec les applications sur l'hôte et avec des composants divers sur les routeurs. Dans une implémentation développée par ISI et Sun Microsystems, RAPI (Rsvp Application Programming Interface), les appels suivants sont implémentés :

- `rapi_session()` crée et initie une session et retourne un handler de session
- `rapi_sender()` appelée par l'émetteur pour spécifier les paramètres du flux de données. Le démon RSVP tournant sur l'émetteur utilisera les paramètres du flux pour créer les messages PATH pour ce flux.
- `rapi_reserve()` crée, modifie ou supprime une réservation
- `rapi_release()` demande au démon RSVP de relacher la réservation.

Chapter 6

Conclusion

Un autre but important du streaming, non abordé dans ce rapport, est commercial. En effet le streaming empêche l'utilisateur de sauver les vidéos qu'il regarde, cela évite le piratage. Cela confirme le fait que ce moyen de diffuser des informations est amené à se développer dans le futur.

De plus, le principe de réservation de bande passante pour garantir une qualité de service, qui n'est quasiment pas utilisé pour le moment, risque de beaucoup se développer dans l'avenir.

Il y a 3 principaux logiciels qui se sont développés, sur des standards différents : QuickTime, Windows Media Player et RealPlayer. Mais plusieurs autres applications peuvent être développées sur le même modèle, concernant la vidéoconférence, la téléphonie, etc...

Bibliography

[1] James F. Kurose, Keith W. Ross: *Computer Networking: A Top-Down-Approach Featuring the Internet*

[2] Jian Lu *Media Streaming and Broadcasting in the Internet*

[3] `http://www.cis.ohio-state.edu/~cliu/ipmultimedia/`

[4] `http://www.rfc-editor.org/`

[5] `ftp://ftp.isi.edu/in-notes/rfc1889.txt`

[6] `ftp://ftp.isi.edu/in-notes/rfc2205.txt`