IFSIC DEA INFORMATIQUE
UNIVERSITE DE RENNES I

# RAPPORT SUR LES TECHNIQUES DE **TRANSPORT** DE FLUX **MULTIMEDIA** SUR UNE ARCHITECTURE A **DIFFERENTIATION DE SERVICES**

CHAUMONT Marc

Sous la direction de J-M. BONNIN, L. TOUTAIN, O. MEDINA

5 juin 2000

Je tiens à remercier O. MEDINA qui m'a suivi et dirigé pendant tout ce stage. Je remercie aussi J-M BONIN dont les remarques sur mon rapport m'ont été fort utiles. Je remercie enfin l'équipe ARMOR et ses stagiaires pour la bonne humeur qu'ils m'ont apporté.

**Note pour le lecteur :**
Le rapport est fait sous Word, je m'en excuse et je m'en défend à l'avance. J'ai fait mon DEA, et par la même mon stage, en même temps que mon service militaire. Ainsi, la rédaction de celui-ci à été effectué sur ma machine sous un environnement Windows.

**1 INTRODUCTION ..... 4**

**2 MPEG1 ET MPEG2 ..... 6**

2.1 LE COMITÉ MPEG ..... 6
2.2 MPEG1 ET MPEG2 AUDIO ..... 7
2.3 MPEG1 ET MPEG2 VIDÉO ..... 11
2.4 MULTIPLEXAGE MPEG1 MPEG2 ..... 16

**3 APPLICATIONS ADAPTATIVES (*STREAMING*) ..... 17**

3.1 LE CONTRÔLE DE CONGESTION ..... 17
3.2 L'ADAPTATION DE QUALITÉ ..... 19
3.3 LE CONTRÔLE D'ERREUR ..... 21
3.4 LE CAS DU MULTICAST ..... 22

**4 RTP/RTCP ..... 23**

4.1 PRÉSENTATION DE RTP/RTCP ..... 23
4.2 RÉFLEXION SUR RTCP ..... 24
4.3 LE TRANSPORT DE MPEG ..... 25

**5 L'ARCHITECTURE À DIFFÉRENTIATION DE SERVICES ..... 29**

5.1 INTRODUCTION ..... 29
5.2 LES SERVICES PROPOSÉS PAR L'ARCHITECTURE À DIFFÉRENTIATION DE SERVICES ..... 30
5.3 LES MÉCANISMES DE FONCTIONNEMENT ..... 31
5.4 L'INTÉRÊT DE LA CLASSE AF POUR LES FLOTS MULTIMÉDIA ..... 36

**6 RÉALISATION ..... 37**

6.1 HIÉRARCHISATION DU FLUX MPEG AUDIO ..... 37
6.2 LA COMMANDE DE LECTURE ..... 39
6.3 RÉCUPÉRATION DES DONNÉES PAR LE CLIENT ..... 41
6.4 OUTILS NÉCESSAIRES À L'IMPLÉMENTATION ..... 42
6.5 ETAT D'AVANCEMENT ..... 43

**7 CONCLUSION ..... 44**

**8 ANNEXES ..... 45**

8.1 MODÈLE PSYCHOACOUSTIQUE ..... 45
8.2 FORMULE DE CALCUL DE LA TAILLE D'UNE FRAME AUDIO MPEG1 OU MPEG2 ..... 46
8.3 EXEMPLE DE COMPRESSION SPATIALE D'UN BLOC ..... 46
8.4 NOTION D'ESTAMPILLE DANS RTP/RTCP ..... 49
8.5 CALCUL DU TAUX DE PERTES VIA LES INFORMATIONS DE RTCP ..... 50
8.6 CALCUL DU RTT VIA LES INFORMATIONS DE RTCP ..... 51
8.7 LE TOCKEN BUCKET ..... 52
8.8 L'ALGORITHME RED ..... 53

**9 BIBLIOGRAPHIE ..... 54**

# 1 Introduction

Actuellement, on assiste à un essor du multimédia. La couverture mondiale d'Internet ainsi que les coûts d'accès y sont bien sur pour beaucoup. De plus, l'attente des utilisateurs est forte. Les applications concernées étant multiples : vidéoconférence, enseignement à distance, formation en entreprise, communication, vidéothèque, surveillance vidéo à la demande, *broadcast*, bornes interactives… Les secteurs d'activité sont d'ailleurs vastes : industries, banques, assurances, éducation, télécommunications, santé, média, tourisme, recherche, défense…

Deux grands axes émergent de ce marché. Tout d'abord celui des serveurs capables de diffuser des vidéos vers un grand nombre de postes clients. D'autre part celui de la vidéoconférence. Pour ce qui concerne les serveurs, il existe beaucoup de solutions propriétaires. Sun vient récemment d'annoncer *StorEdge*, la toute première plate forme média de bout en bout basée sur les standards ouverts [2]. Des kits de développement sont aussi proposés par des sociétés comme *RealNetworks*[1] [3]. En ce qui concerne la vidéoconférence, le nombre de solutions existant est similaire. On peut citer par exemple le logiciel de *Microsoft NetMeeting*.

Même s'il existe beaucoup de solutions, certains problèmes restent non résolus. En effet, les applications multimédia très gourmandes en ressources, nécessitent une réflexion sur les problèmes de stockage et de transport. Des choix doivent être effectués en fonction de critères tels que la qualité, le nombre d'utilisateurs… Il faut par exemple choisir entre l'un des standards actuels de compression H263 (28 à 64Kb/s), MPEG1 (1,5 Mb/s), MPEG2 (plus de 4Mb/s). Ce choix n'est pas anodin, 1heure de vidéo à 28,8 Kbit/s représente 12 Mo alors que 1heure à 6Mb/s représente 2,9 Go [1].

C'est donc dans l'optique d'une solution au transport de flux multimédia que le document suivant s'inscrit. Le challenge est d'ailleurs osé puisque les couches Réseaux (couche OSI 3), Transport (couche OSI 4) et Application (couche OSI 5, 6 et 7) sont concernés. Pour ce qui concerne les flux multimédia, MPEG1 et MPEG2 présentaient de nombreux avantages et c'est ces deux standards qui ont été retenus. La première partie de ce rapport présente donc MPEG1 et MPEG2. Ensuite dans l'idée des couches Internet le plan reprend celles présentées auparavant. La deuxième partie s'intéresse ainsi aux applications *streaming*. Puis, le protocole RTP/RTCP est abordé et enfin l'architecture à différentiation de service. La cinquième et dernière partie présente une solution au transport de flux MPEG Audio sur une architecture à différentiation de services.

L'ensemble des différents chapitres abordés ici, ont pour but de présenter la manière dont doit être transporté un flux multimédia. Ainsi, ce rapport ressemble beaucoup à un tutorial. Cependant, l'idée sous jacente est de comprendre l'ensemble des techniques utiles au transport du multimédia pour pouvoir proposer au final un service utilisant *DiffServ*. Ainsi, la partie présentant MPEG permet de comprendre le mécanisme de représentation des données et donc d'extraire les unités élémentaires. La partie abordant les Applications adaptatives est plus présentée par curiosité intellectuelle. Ceci dit, cette partie permet de comprendre les mécanismes mis en place par les couches supérieures à *DiffServ* et aussi de comprendre l'utilisation possible de RTP/RTCP. Elle a de plus motivé l'utilisation d'un transport en plusieurs couches. Pour ce qui est de l'utilisation de RTP/RTCP, et son apport à une solution sur *DiffServ*, on peut dire qu'il est faible. Ceci dit, l'ensemble des *drafts* proposés pour un transport de flux multimédia sont à aborder. De plus, c'est le standard, de la couche transport, avec UDP, pour les flux multimédia. En résumé, si l'on veut proposer un service réaliste sur

[1] RealNetworks est connu pour son logiciel RealPlayer et son format audio .ra et video .rv

*DiffServ*, et effectuer des tests, alors il faut connaître et mettre en œuvre les couches supérieures.

# 2 MPEG1 et MPEG2

Dans cette partie, nous allons présenter MPEG1 et MPEG2 Audio et Vidéo. Cette présentation a pour but de comprendre les mécanismes de représentation des données et de trouver une façon de découper le flux. Dans un premier temps, nous présentons le comité MPEG, puis MPEG1 et MPEG2 Audio, la technique de compression et la représentation, ensuite MPEG1 et MPEG2 Vidéo, la technique de compression et la représentation, et enfin, MPEG système.

## *2.1 Le comité MPEG*

### Introduction

MPEG [4] est le surnom du groupe de travail JTC1/SC29/WG11 de titre *Coding of Moving Pictures and Audio*. Les trois principales organisations de standardisation de ce comité, qui réunit environ trois cents experts quatre fois par an, sont l'ITU, *International Telecomunication Union*, L'IEC, *International Electronical Commission*, et l'ISO, *International Organisation for Standardisation*.

MPEG est à l'origine de la norme MPEG1 et MPEG2 qui a rendu la télévision numérique possible. Le comité vient aussi d'achever la deuxième phase de développement de MPEG4 (décembre 1999) et développe actuellement MPEG7 qui donnera lieu à un *Comitee Draft* en octobre 2000 et un *Draft International Standard* en juillet 2001.

Ce comité travaille depuis 1987 pour lancer le multimédia. Bien sûr, les standards ont été adoptés comme le montre l'utilisation de MPEG1 pour les CD vidéo ou bien l'utilisation de MPEG2 pour la télévision haute définition. Ceci dit, la standardisation n'est pas aussi rapide qu'elle pourrait l'être. En effet, les difficultés viennent des rivalités économiques entre l'industrie du divertissement, l'industrie des télécommunications et l'industrie informatique (c'est-à-dire le contenu, le transport et l'équipement).

### Les normes MPEG

MPEG1, ISO/IEC 11172, apparu en 1988, est le premier standard du groupe. Il combine le signal audio et le signal vidéo à un taux de 1,5 Mbit/s. La principale motivation de MPEG1 était de permettre le stockage sur CD d'un film de qualité comparable aux cassettes VHS. Ce standard est actuellement le format utilisé sur Internet pour la vidéo ainsi que pour l'audio. Les principales innovations portent sur le couplage de l'audio et de la vidéo, sur une implémentation logiciel et sur un nouveau format vidéo.

MPEG2, ISO/IEC 13818, apparu en 1994 a été motivé par la diffusion à haut débit (supérieur à 10Mbit/s) de haute qualité pour la télévision numérique. Tout comme MPEG1, MPEG2 a rencontré beaucoup de succès dans la diffusion par satellite ou les DVD. C'est un standard assez modulaire à la différence de MPEG1. En effet, différents profils d'implémentation sont normalisés pour des utilisations très différentes. La partie 10 de MPEG2 est d'ailleurs encore actuellement en développement. La nouveauté porte, pour l'ensemble des parties de MPEG2, sur la prise en compte dans le codage des problèmes liés a la communication, dans un environnement avec occurrence d'erreurs entre le client et le

serveur. Une autre caractéristique intéressante est l'ajout d'informations annexes telles que la nature du film ou bien le copyright.

MPEG4 est fondamentalement différent de par sa capacité à coder les objets visuels de formes arbitraires. En effet dans MPEG2, on partitionne l'image avec des blocs rectangulaires. La caractéristique principale de MPEG4 est donc de permettre une plus grande interactivité avec les objets AVO, *Audio Visual Objects*. Il est alors possible de composer des scènes, de changer le point de vue, d'ajouter des effets vidéo et audio…

## *2.2 MPEG1 et MPEG2 Audio*

### Introduction

MP3 c'est à dire MPEG1 Audio *Layer III* ou MPEG2 Audio *Layer III* est le format audio le plus en vue actuellement. Il est recommandé par l'ITU-R pour les schémas de codage audio adaptés au faible débit (60 kbit/s par chaîne) [6]. C'est aussi le standard audio le plus utilisé. Il est adopté par les industriels comme pour *WinAmp* de *NullSoft*, *RealPlayer* de *RealNetworks*, *NetShow* de *Microsoft*. De fait, il offre un bon compromis entre les fichiers de type WAV qui sont trop volumineux et les fichiers de *streaming* de type *Real Audio* qui s'avèrent de trop basse qualité.

MP3 est apparu en 1988 avec la norme MPEG1 (ISO/IEC 11172) puis une autre version via MPEG2 (ISO/IEC 13818) en 1994. Il propose des taux d'échantillonnage allant de 8 kbit/s pour une qualité téléphonique jusqu'à 128 kbit/s pour une qualité CD [7]. Les techniques de compression reposent sur la suppression de l'information inutile à l'oreille humaine. Les fréquences en dehors de l'intervalle 2khz, 5 Khz sont moins bien perçues. On supprime les fréquences intenses masquant les fréquences proches plus faibles. C'est l'effet de masque. De plus, on met en place le réservoir d'octets pour obtenir un débit pratiquement constant. Dernière chose, on utilise le codage d'Huffman. Les techniques mises en œuvre se complètent idéalement puisque dans le cas de polyphonies, l'effet de masque est très efficace alors que dans le cas de son « purs », c'est le codage d'Huffman qui fonctionne bien (les octets redondants sont nombreux) [8]. Les *Layer I* et *II* offrent des fonctionnalités moindres par rapport à la *Layer III* ceci dit leur codage est beaucoup moins complexe.

MPEG1 Audio permet de compresser une à deux chaînes. Les fréquences d'échantillonnage sont de 32, 41.1 et 48KHz et l'on a des débits de 32 à 448Kb/s. MPEG2 Audio définit quant à lui deux formats de compression. MPEG2 BC, *Backward Compatible*, peut être lu par un décodeur MPEG1 mais est moins efficace que MPEG2 AAC, *Advanced Audio Coding*, qui lui n'est pas BC. Pour ce qui est de MPEG2 BC, il permet jusqu'à 5 chaînes plus une chaîne LFE, *Low Frequent Enhancement*. Les fréquences d'échantillonnage sont 16, 22.05 et 24 KHz et l'on a des débits allant de 8 à 256Kb/s. MPEG2 AAC permet quant à lui jusqu'à 48 chaînes avec des taux d'échantillonnage allant de 8 à 96KHz pour des débits de 8 à 160Kb/s/chaine [9].

MP3 est comme on le voit le format du moment, ceci dit, il ne tardera pas à se voir remplacé par de nouveaux formats de compression plus efficace encore comme MPEG2-AAC ou bien MPEG4 qui sont déjà présents. Ce sont d'ailleurs les dignes successeurs de MP3 puisque leur efficacité est encore plus grande. Dans la suite, nous nous intéressons plus particulièrement à MPEG1 et MPEG2 Audio pour tout les avantages cités ici.

## Représentation initiale du signal audio

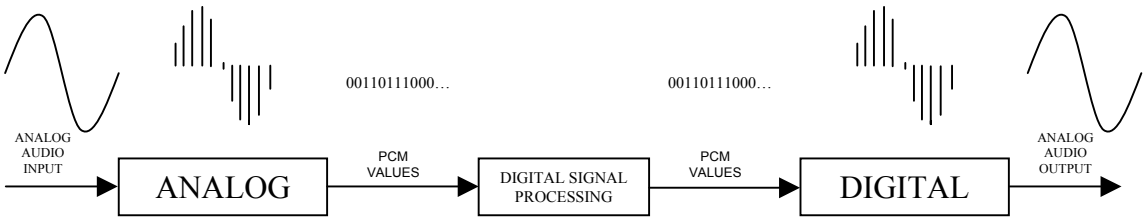



**Figure 1 Digital Audio Process**

La conversion du signal analogique au signal numérique commence par un échantillonnage à intervalle discret du signal audio et par une quantification discrète des niveaux de l'échantillon. La méthode de représentation de chaque échantillon est appelée PCM, *pulse code modulation* [Figure 1]. Les deux grands formats de représentation PCM sont RIFF WAVE (.wav) et AIFF (.aiff).

Quand le signal couvre un intervalle de fréquence de 20Khz (typiquement l'intervalle audible humain) le taux d'échantillonnage minimum selon Nyquist est de 40Khz. Typiquement les taux d'échantillonnage varie entre 8Khz et 48Khz.

En ce qui concerne le niveau de quantification, celui ci est classiquement une puissance de 2 pour utiliser un nombre fixe de bits par échantillon. L'ajout d'un bit par échantillon correspond à un gain d'amplitude d'environ 6dB. L'intervalle de bits alloué par échantillon se situe généralement entre 8 et 16.

Ainsi, la quantité de données d'un CD à 2 *channel*, échantillonné à 44,1Khz en PCM avec 16 bits par échantillon est de 1,4 Mb/s. On comprend aisément qu'il est difficile de stocker autant d'informations et de même, il est difficile de la transporter. On en vient donc naturellement à des méthodes de compression [5].

Il existe des méthodes simples de compression tel que μ-law (format .au de Sun) ou l'ADPCM avec une faible complexité, une compression faible, et une qualité audio moyenne. Ces techniques sont plutôt destinées à la compression de parole. Il existe aussi des techniques plus compliquées avec une grande complexité, une forte compression et de grande qualité audio tel que MPEG Audio.

## Schéma de compression MPEG Audio

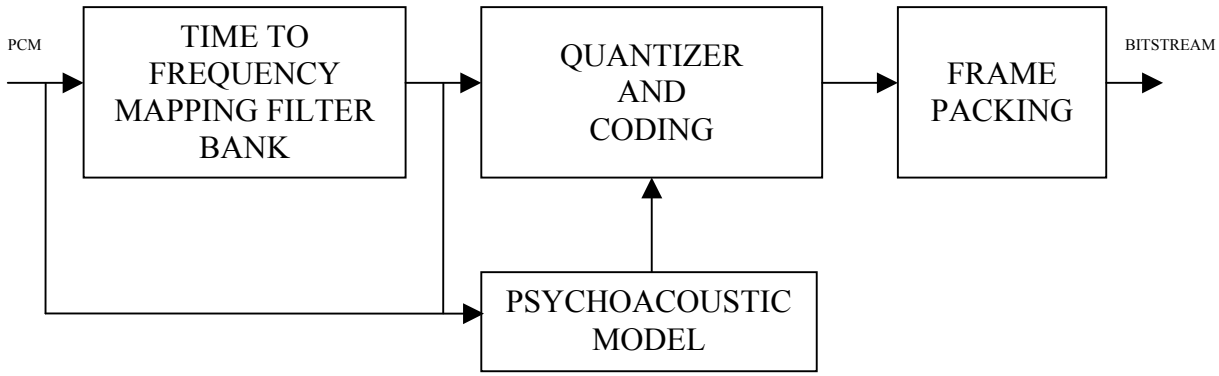



**Figure 2 Codeur MPEG Audio**

Le codage en sous bande, *Sub Band Coding* (SBC), est une méthode générale puissante de compression. A la différence de méthodes spécifiques à la source, tel que LPC qui ne fonctionne bien qu'avec une source audio de type parole, SBC peut compresser n'importe quelle source. MPEG Audio est le SBC le plus populaire.

Quatre étapes sont mises en œuvre pour la compression [Figure 2]. Tout d'abord, une fonction de transformation de l'échelle du temps en échelle des fréquences avec une décomposition de l'intervalle de fréquence en 32 sous-bandes. Ce filtre est appelé ***time-frequency mapping*** ou bien ***filter bank***. Comme on l'a vu précédemment le signal audio d'entrée est sous forme PCM. La première étape est donc le filtrage de 32 échantillons PCM grâce au filtre *time-frequency mapping*. On obtient en sortie 32 échantillons de fréquence.

La deuxième étape se réalise en même temps que la première. Un modèle psychoacoustique, ***psychoacoustic model***, analyse les 32x12 (couche I) ou les 32x12x3 (couche II et III) échantillons PCM d'entrée ainsi que les 32x12 (couche I) ou les 32x12x3 (couche II et III) échantillons de fréquence associés, issu du filtre *time-frequency mapping*. Un seuil de masquage est alors déterminé pour chaque bande et ceci en utilisant les informations psychoacoustiques [8.1 Modèle psychoacoustique]. Le calcul du ratio *signal-to-mask* pour chaque bande est alors effectué. Ce ratio correspond à la division de la valeur maximale de la bande (12 ou 12x3 valeurs par bande) par le seuil de masquage de la bande.

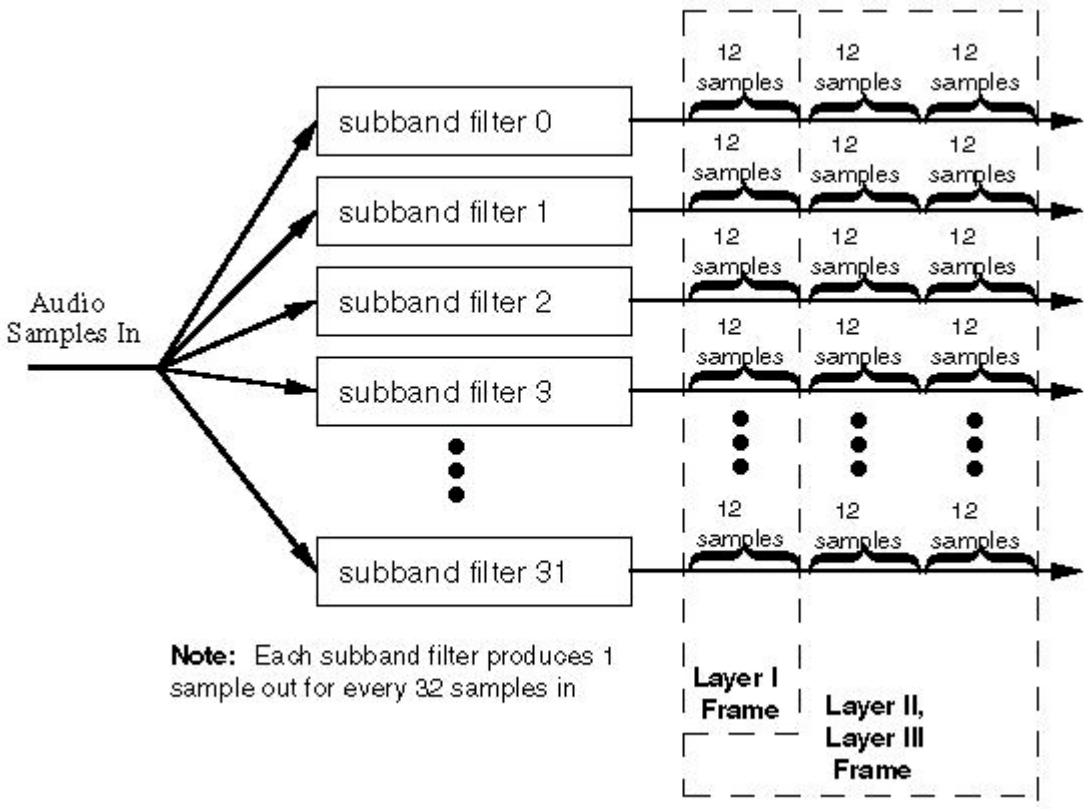



**Figure 3 Schéma de formatage**

La troisième étape est la quantification et le codage, ***quantizer/encoder***. On dispose d'un groupe de 12 échantillons par bande dans le cas de la couche I et de 3x12 échantillons par bande dans le cas de la couche II et III [Figure 3] ainsi qu'un ratio *signal-to-mask* par bande. Dans un premier temps, on détermine le nombre de bits à allouer pour représenter un élément de la bande. On appelle ce nombre le *bit allocation*. Il doit être trouvé de sorte que le bruit associé à ce nombre de bits (6bB par bits) soit inférieur au seuil de masquage de la sous-

bande. (*bit allocation* ≤ *seuil*/6). Une fois le bit allocation déterminé, il faut trouver le facteur de quantification de la sous bande, *scale factor*. La valeur maximale du groupe de la sous bande doit être représentable sur le nombre de bit alloué par le *bit allocation* (*scale factor* ≥ valeurmax/$2^{bit\ allocation}$). Une fois que le *scale factor* et le *bit allocation* sont déterminés pour le groupe alors on peut quantifier.

La dernière étape consiste à empaqueter les 32 bandes dans une *frame* audio, ***frame packer***. Un en-tête est placé en début de trame [Représentation de flux : bitstream] puis éventuellement un CRC et enfin les donnés audio. Chaque sous-bande est précédée de bits décrivant le *bit allocation* et le *scale factor* [5] [10].

## Représentation de flux : *bitstream*

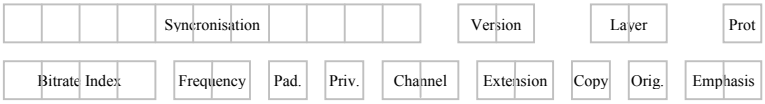
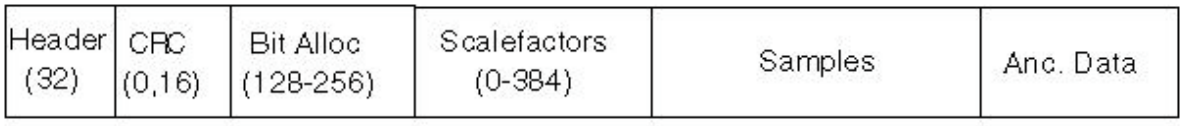
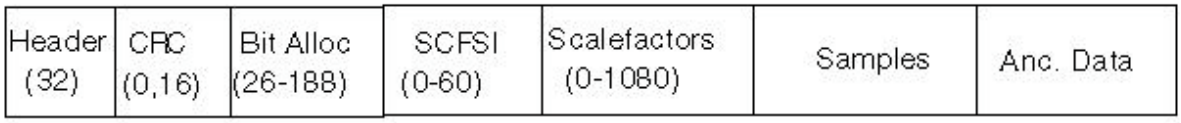



**Figure 4 En-tête MPEG Audio**

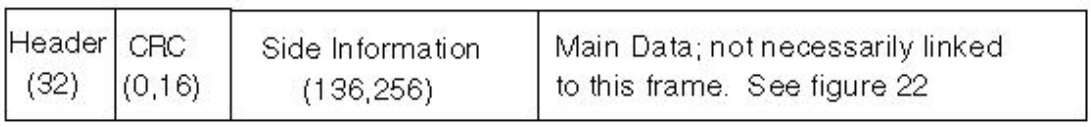
L'unité élémentaire du *bitstream* est la *frame*. Celle-ci est composée d'un en-tête suivi des informations audio. Le *bitstream* est donc une succession de *frame*. Pour ce qui concerne la partie donnée audio, la Couche I contient toujours 384 échantillons par *frame* et les Couches II et III contiennent 1152 échantillons par *frame*. Connaissant la fréquence d'échantillonnage ainsi que le débit du *bitstream* il est alors facile de déterminer la longueur de la *frame* [8.2 Formule de calcul de la taille d'une frame audio MPEG1 ou MPEG2]. Il faut remarquer cependant que le débit du *bitstream* peut varier en fonction des échantillons composant la *frame*. Ainsi, la taille de la *frame* n'est pas toujours constante. Ceci permet de faire de meilleures compressions tout en conservant une grande qualité [11].




*19a The Frame Format of a Layer I Bitstream*




*19b The Frame Format of a Layer II Bitstream*




*19c The Frame Format of a Layer III Bitstream*

**Figure 5 Format en fonction de la couche**

Pour ce qui est de l'en-tête [Figure 4], elle est composée de 32 bits puis éventuellement d'un CRC sur 16 bits et enfin pour la couche III d'informations supplémentaires [Figure 5]. Sur les 4 octets d'en-tête, on retrouve tout d'abord 11 bits de synchronisation puis des informations comme la version (MPEG1, MPEG2 ou MPEG2,5), le débit, la fréquence d'échantillonnage, le copyright, etc.… Il existe aussi une description du fichier MPEG audio via le Tag ID3v1. Ce tag fait 16 octets et est placé totalement à la fin du fichier audio. Des informations tel que le titre de la chanson, l'artiste ou bien le genre sont présentes.

Du fait des différentes techniques de compression utilisées par chaque couche, le format des données après l'en-tête et l'éventuel CRC ne sont pas identiques. La Figure 5 montre ce qu'il en est, avec l'intervalle en nombre de bits (séparé par une virgule) ou les deux valeurs possibles en nombre de bits (séparé par un tiret) [12]. Il faut remarquer aussi que le format d'empaquetage de la couche III, c'est à dire mp3, est beaucoup plus complexe que les deux autres couches car la technique du réservoir d'octet est mis en place [Figure 6]. Ainsi, la chronologie des échantillons dans le *bitstream* n'est pas respectée. Une *frame* n'est donc pas indépendante, c'est à dire quel ne peut être joué si l'on ne dispose que d'elle. Ce n'est donc pas une *Application Data Unit* (ADU).

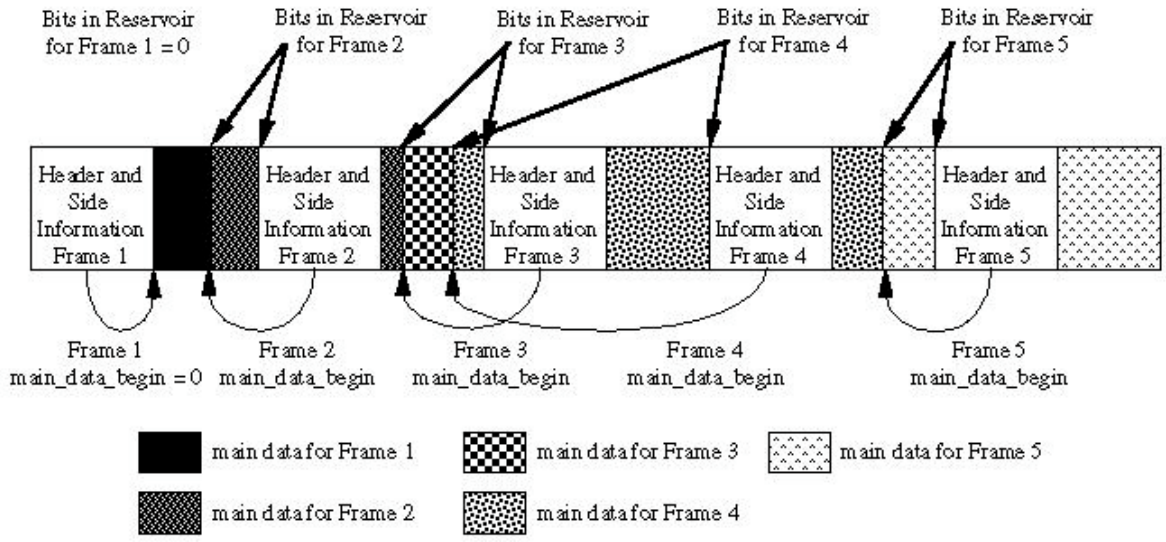



**Figure 6 *Reservoir bit* dans mp3**

## *2.3 MPEG1 et MPEG2 Vidéo*

Maintenant que nous avons présenté le codage Audio, nous abordons le codage vidéo. Nous présentons tout d'abord la technique de compression puis la représentation du *bitstream*.

### Introduction

MPEG1 et MPEG2 sont actuellement des standards inévitables de la compression vidéo. L'originalité de la méthode de compression est d'utiliser la prédiction spatio-temporelle. Ceci correspond à l'élimination des redondances d'information dans une image (compression spatiale) et à l'élimination des redondances entre deux images (compression temporelle).

## Représentation du signal Vidéo Initial

Un signal vidéo est représenté par une succession d'images individuelles dont chacune est modélisée par une matrice de pixel. En fait, le signal analogique est échantillonné à intervalle discret, c'est à dire typiquement entre 25 et 30 Hz. Chaque échantillon (image) est discrétisé sous forme de matrice de pixels avec des représentations pixel allant de 3x8 bits à 3x24 bits. La représentation de chaque pixel est définie par les trois composantes (Y, U, V) où Y est la luminance et U et V sont les composantes de chrominance. Ce système de représentation peut être transposé dans le système (R, V, B).

Le signal vidéo standard de la télévision numérique avant compression est le CCIR-601. La résolution est de 720x480 pour la luminance et 360x240 pour les signaux de chrominance. Une telle résolution peut nécessiter un pré traitement pour réduire celle ci. Ainsi, MPEG1 procède à la conversion de format CCIR-601 au format SIF. Cette conversion correspond pour la luminance à diviser le nombre de lignes initial par deux et à utiliser un filtre horizontal. On obtient alors une résolution de 360x240. Pour la chrominance on divise aussi le nombre de lignes par deux et l'on filtre horizontalement. On obtient alors une résolution de 180x120.

MPEG1 divise l'image Y au format SIF en macroblocs de 16x16 valeurs. La résolution horizontale n'étant pas un multiple de 16, on élimine 4 valeurs à gauche et 4 à droite de chaque ligne. De même, pour les images U et V au format SIF, on élimine deux valeurs à droite et deux valeurs à gauche de chaque ligne. (Pour la chrominance, les blocs peuvent être de résolution 8x8 ou 16x16 ; le raisonnement aboutit à la même action). La matrice de luminance Y est donc alors de résolution 352x240 et les matrices de luminance sont de résolution 176x120 [13].

## La compression Vidéo MPEG

La compression d'un flux vidéo en format MPEG génère trois types d'images : Intra, Prédites et Bidirectionnelles [14]. Les images Intra I sont codées sans compensation spatiale et servent de référence. Ces images n'ont pas besoin d'autres images pour être décodées. Dans le cadre d'une transmission réseau, ces images sont précieuses car elles transportent beaucoup d'informations et permettent la re-synchronisation de la séquence vidéo. Leur taux de compression est bien sûr peu élevé. Les images Prédites P sont calculées à partir de l'image I ou P antérieure. Celles-ci servent aussi de référence aux images B. Les images bidirectionnelles B, quant à elles présentent le meilleur taux de compression, mais leur existence demande un réarrangement des images pour la transmission. Ce type d'images, appelées aussi interpolées, utilise deux références pour être calculé : l'image d'avant et celle d'après.

### La compression spatiale

Comme on l'a vu, une image se présente sous la forme de trois matrices Y, U, V. Chacune de ces matrices est partitionnée en blocs de 8x8 éléments. Chacun des blocs se voit alors appliquer une DCT, *Discrete Cosine Transform*. En annexe se trouve un exemple de compression d'un bloc MPEG. La caractéristique de cette transformation est de concentrer les valeurs non nulles dans le coin supérieur gauche du bloc. Ensuite, nous appliquons un quantificateur sur notre matrice bloc. C'est ici que l'on réalise une compression avec perte d'information. En effet, la quantification réduit le domaine de valeurs de manière irréversible.

Ceci dit, la distorsion engendrée n'est pas visible. Puis, on représente la matrice bloc par la suite de valeurs issues du parcours en zigzag de celle ci. Puisque les valeurs finales sont des zéros, on ne les représente pas. Enfin, on utilise le code de type Huffman en codant les valeurs les plus souvent rencontrées par un code plus court que les autres [Figure 7]. La compression spatiale est réalisée sur les trois types d'image I, P, B. Les images P et B ont en plus une compression temporelle.

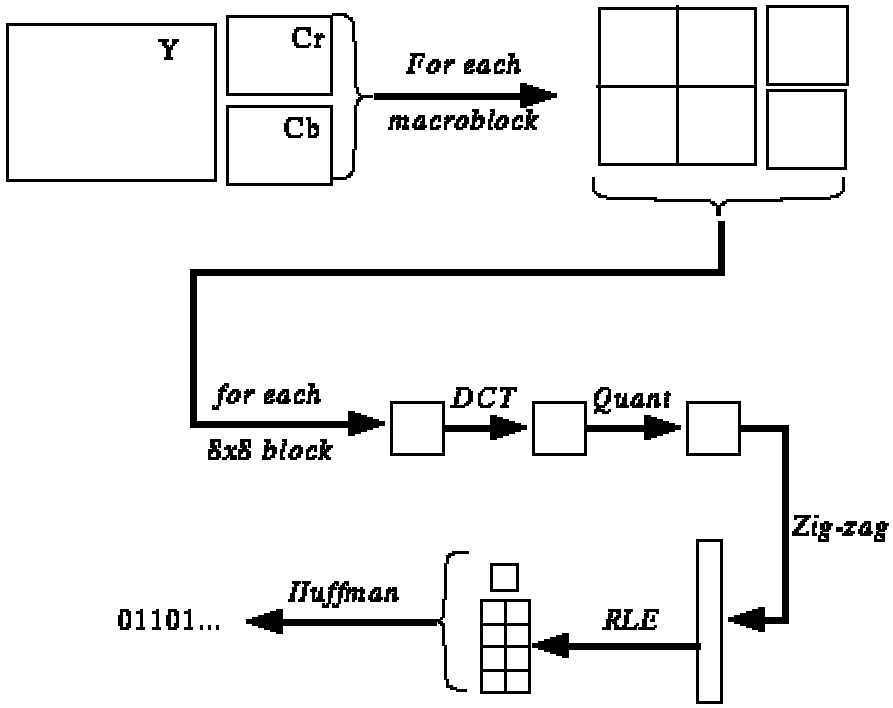



Figure 7 Compression spatiale

La compression temporelle

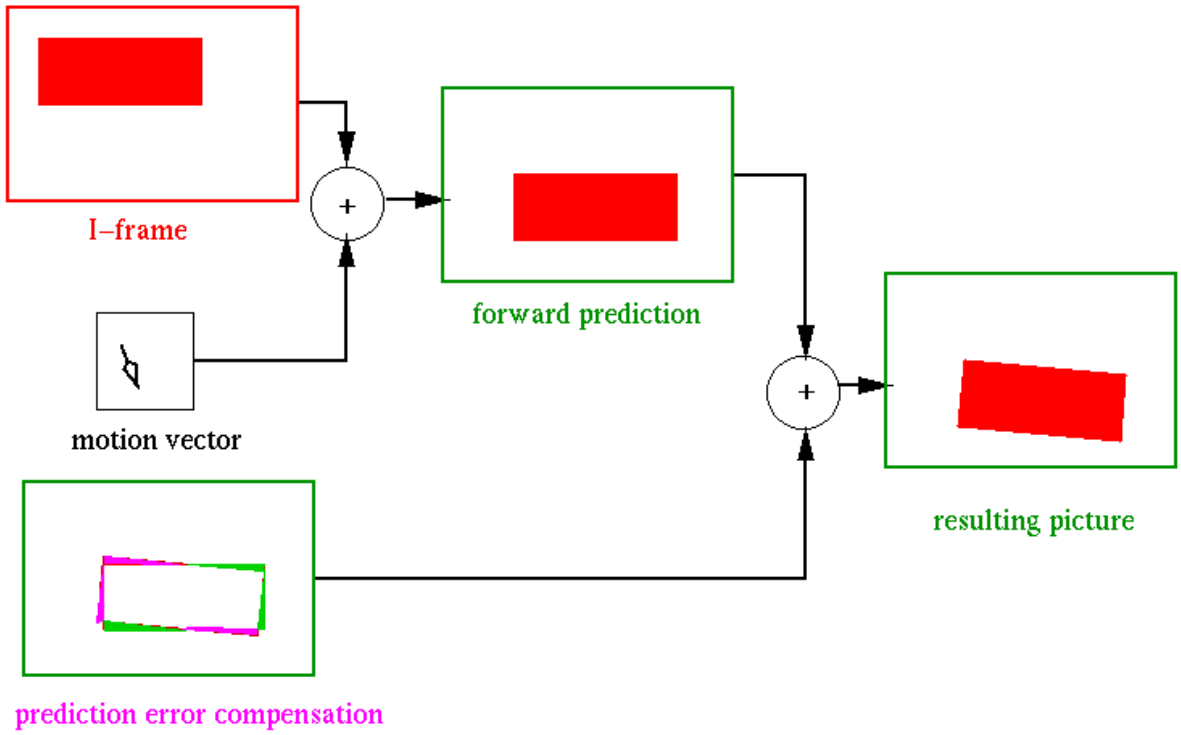


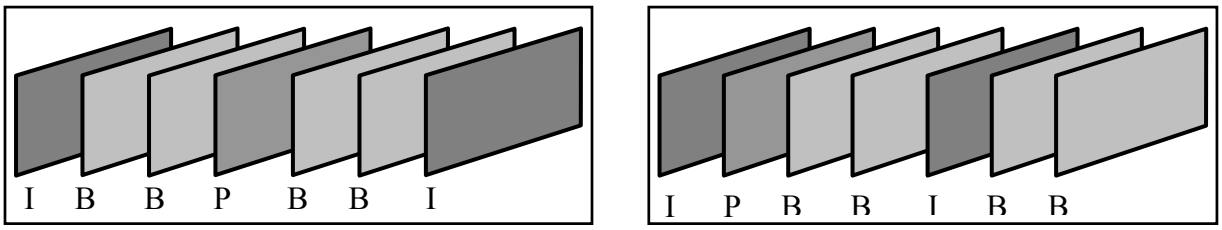

Figure 8 Compression temporelle

La compression sur la redondance temporelle utilise l'image quantifiée, la reconstruit et la compare à l'image précédente. Cette comparaison donne lieu à la détermination de vecteurs de mouvements ainsi qu'à des erreurs de prédiction (différences entre l'image au temps t et l'image au temps t-1) [Figure 8]. Les vecteurs de mouvement ainsi que les erreurs de prédiction recevront alors une compression spatiale.

Pour déterminer les vecteurs de mouvement (cas des vecteurs de déplacement en *forward*) entre deux images consécutives, il faut tout d'abord choisir un macrobloc 16x16 de l'image au temps t. On recherche alors dans une zone de l'image t-1, centrée sur la position du macrobloc de l'image t, le macrobloc. Le déplacement de ce macrobloc entre l'image t-1 et l'image t donne le vecteur de mouvement *forward* associé au macrobloc de l'image t. Cette opération est ce que l'on appelle le *bloc matching*.

Une fois le vecteur déplacement du macrobloc calculé, il ne reste plus qu'a coder l'erreur issue de la translation du macrobloc de l'image t-1 grâce au vecteur déplacement et l'image t. Le macrobloc erreur de prédiction est alors compressé spatialement [13].

## Représentation de Flux (*bitstream*)




Ordre de Présentation Ordre de transmission

**Figure 9 Group d'image (GOP)**

Le flux MPEG Vidéo est une succession de groupe d'images (GOP). De manière classique l'enchaînement des images est du type IPBBPBBIBB (ordre de transmission) [Figure 9]. Chaque image est composée de tranches, *slice*, et un *slice* est lui-même formé de macroblocs. Les macroblocs en format 4:2:0 sont composés de 6 blocs : 4 blocs de luminance Y, 1 bloc de chrominance Cb, et 1 bloc de chrominance Cr. En format 4:2:2, les macroblocs sont formés de 4 blocs de luminance, 2 blocs de chrominance Cb et 2 blocs de chrominance Cr. Enfin, en format 4:4:4, les macroblocs sont formés de 4 blocs de luminance, 4 blocs de chrominance Cb et 4 blocs de chrominance Cr [15]. Lorsque l'on a affaire à des images de type P ou B, les vecteurs de mouvement sont calculés grâce au déplacement supposé du macrobloc entre deux images consécutives [Figure 10].

Un bloc correspond à une matrice 8x8 de valeurs. Selon les cas, les valeurs du bloc peuvent être soit des informations de pixel (cas de l'image I), soit des informations d'erreur de prédiction (cas de l'image P et B). Dans tous les cas, le bloc est compressé par le biais de la DCT, quantifié et adressé en zigzag [8.3 Exemple de compression spatiale d'un bloc]. La représentation exacte du bloc est donc une suite de valeurs non nulle terminée par un marqueur de fin de bloc [16].

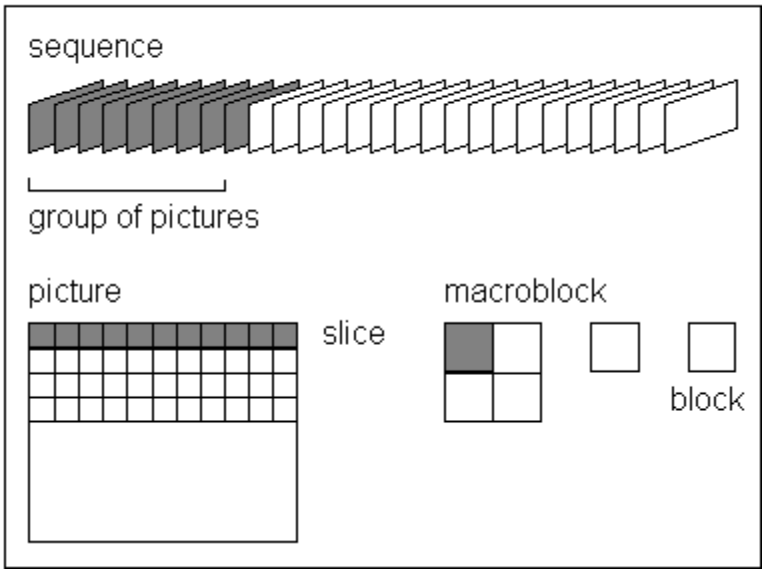
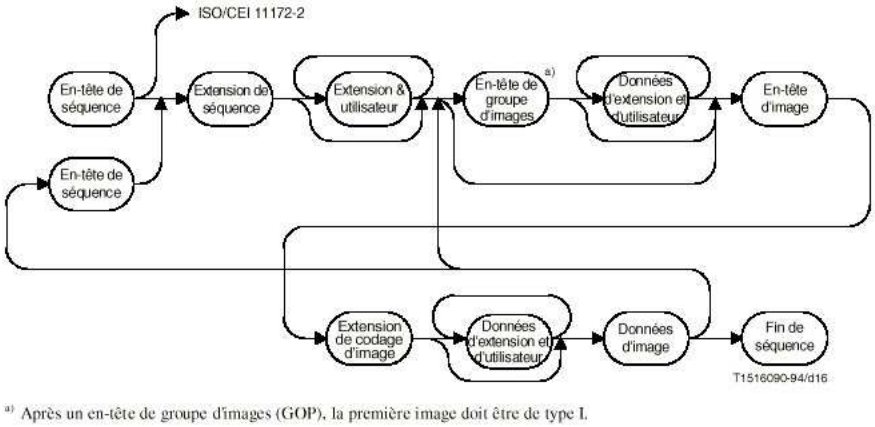



**Figure 10 Flot MPEG Vidéo**

Finalement, le *bitstream* vidéo se résume en une suite de blocs et de vecteurs de déplacement avec des en-têtes informatives. Ceci dit, l'unité de donnée élémentaire de donnée n'est pas aussi microscopique. En effet, cette ADU, *Application Data Unit*, est le slice. Dans la suite, nous ne nous occuperons plus de tous ce qui est macrobloc et bloc. La figure 11 montre la structure du bitstream MPEG2 qui est similaire à MPEG1. Le séquence header marque le début d'un groupe d'image (GOP). Celui ci permet l'accès aléatoire dans le bitstream. Puis, le *GOP header* peut être présent, il est suivi par un *picture header*. Ensuite, une série de *slice* composent l'image. Les *slice* prennent une portion d'image de 16 lignes x nombreDeBlocs. Chaque slice est codé indépendamment des autres *slice* de l'image, ce qui permet de borner les erreurs du *bitstream*. Ainsi, lors de la détection d'une erreur, on passe à l'autre *slice*. C'est en partie pour cela que le *slice* est l'ADU. Le *slice* est lui-même composé de macroblocs…




a) Après un en-tête de groupe d'images (GOP), la première image doit être de type I.

**Figure 11 Flux binaire MPEG2**

## *2.4 multiplexage MPEG1 MPEG2*

MPEG1 et MPEG2 ne multiplexe pas exactement de la même façon l'audio et la vidéo. MPEG2 est plus général car il autorise le multiplexage d'un nombre illimité de flux élémentaires. Il offre de plus, deux types de multiplexage. L'un est similaire à celui de MPEG1 et l'autre est plus adapté au transport.

### Partie système MPEG1

La compression de l'audio et de la vidéo est faite indépendamment. On obtient donc deux flux binaires comme ceux décris plus haut. Les deux flux sont alors synchronisés à l'aide d'une horloge travaillant à 90KHz. Des estampilles temporelles, *timestamp* sont ajoutées dans le signal produit du multiplexage. Ceci permet la resynchronisation lors du décodage.

### Partie système MPEG2

Les flux audio et vidéo codés sont, chacun de leur coté, empaquetés dans ce que l'on appel un PES, *Packet Elementary Stream*. Une estampille temporelle leur est donc associée. Une fois que le *packetizer* est passé, il ne reste plus qu'à multiplexer les différents PES. Deux types de *bitsteam* peuvent alors être générés. Il y a le flux de programme PS, *Program Stream*, qui est similaire au flux système MPEG1. Il est utilisé pour multiplexer ensemble des flux élémentaires qui ont une base de temps commune, et qui doivent être affichés d'une manière synchrone. Le PS utilise des paquets à longueur variable. Il y a aussi le flux TS, *Transport Stream*, qui est utilisé pour multiplexer des flux n'utilisant pas de base de temps commune. La longueur des paquets TS est fixe et fait 188 octets. Ce flux est adapté aux transports sur des canaux non fiables.

# 3 Applications adaptatives (*streaming*)

La partie présenté ici aborde les applications adaptatives. Cette partie est exposé pour comprendre le fonctionnement des applications transportant du flux multimédia. Cette compréhension à permis de proposer un service de transport de MPEG Audio via *DiffServ* [6 Réalisation] qui est totalement adapté au schéma de fonctionnement des application adaptatives.

Les caractéristiques d'un canal non fiable et plus exactement du canal Internet sont sa bande passante, les délais introduits et les pertes introduites. Les applications multimédia accaparent la bande passante au détriment d'autres applications et ne supportent pas bien les délais ni les pertes d'information. Actuellement, ce genre d'applications se déploient énormément, il est donc nécessaire de mettre en place des mécanismes qui vont faire que les flux multimédias s'adaptent plus harmonieusement au canal de transport.

Une application adaptative, *streaming*, est une application qui envoie sur le réseau un flux multimédia (audio, vidéo), dont l'équité avec TCP est respectée (le partage de la bande passante est équitable entre UDP et TCP), dont le débit s'adapte à la bande passante disponible (la diminution de la qualité du flux en terme de résolution est maîtrisée), et dont la qualité en terme de fluidité de la réception est assurée (la robustesse aux pertes est assurée). Les trois points essentiels à respecter lors de l'élaboration d'un logiciel de transports de flux multimédia sont donc le **contrôle de bout en bout de la congestion**, **l'adaptation de qualité**, le **contrôle d'erreur**.

Il est à noter qu'une architecture à différentiation de service permet d'une certaine manière de réserver de la bande passante pour une classe donnée. Ceci dit, rien n'empêche un flot d'expérimenter une congestion dans sa propre classe. Les mécanismes présentés ici sont donc de manière évidente des mécanismes qui seront présents dans les applications multimédia à venir. Il est donc essentiel dans le cadre d'une étude sur le transport de multimédia sur une architecture à différentiation de service, de prendre en compte ces mécanismes et les protocoles existant pour une meilleure adaptation aux données transportées.

## *3.1 Le contrôle de congestion*

Ici, le but est de récupérer des informations sur la congestion du réseau et du client pour déterminer un débit d'émission à respecter. Tout d'abord, il est important pour l'application d'adapter son débit en fonction de la bande passante libre sinon quoi, on risque de saturer le réseau et d'augmenter les pertes de paquets. De plus, actuellement, il y a une augmentation des flux audio et vidéo sur Internet. Il est donc nécessaire d'améliorer l'équité inter protocole entre TCP et UDP en mettant en place un mécanisme de contrôle de congestion de bout en bout [18]. Dernier point, si le mécanisme de TCP-friendly est mis en place dans les routeurs, c'est à dire que le partage de la bande passante est équitablement partagé entre TCP et UDP. Un flux UDP ne régulant pas son débit va voir automatiquement une augmentation des pertes globales en flux UDP. Ainsi, on assistera à une dégradation non plus uniquement de TCP mais aussi d'UDP. D'où la nécessité pour tous les flux UDP de contrôler leur débit.

### La fonction de décision

Le principe est de décider si l'on doit augmenter le débit actuel ou bien le diminuer. Pour faire ce choix, il est nécessaire de prendre en compte le taux de pertes qui indique la congestion éprouvée par un participant donné. Ici, bien entendu, on ne fait pas du tout de prévention sur la perte de paquets et l'on effectue une correction qu'après la perte effective de paquets. Le protocole RTCP permet d'extraire le taux de pertes ($\lambda$) d'un participant [8.5 Calcul du taux de pertes via les informations de RTCP]. On compare alors ce taux à deux seuils fixés (low_loss et high_loss) pour déterminer s'il y a CONGESTION, NONCONGESTION ou CHARGE [17]. Pour éviter les oscillations de ce taux, on utilise un filtre passe bas avant son utilisation : $\lambda \leftarrow (1 - \alpha)\, \lambda old + \alpha\, \lambda new$. La valeur de 0,3 pour $\alpha$ semble intéressante [17].

Il faut remarquer que l'on s'appuie ici sur un mécanisme de contrôle déjà existant qui est RTCP. Le protocole RAP [20] met quand a lui en œuvre ce mécanisme mais pour de l'unicast. Actuellement, les taux de pertes seuils utilisés sont low_loss = 2% et high_loss = 5% [19] [17]. Dans le cas unicast, la CONGESTION correspond à la décision de DECROITRE le taux de transmission et la NONCONGESTION correspond à la décision faire CROITRE le taux de transmission. Lorsque l'on tombe sur l'état CHARGE, on ne modifie pas le taux de transmission. Ci dessous, la fonction de décision :

$\lambda = (1 - \alpha)\, \lambda old + \alpha\, \lambda new$

**SI** $\lambda >$ high_loss **ALORS** DECROITRE
**SI** $\lambda <$ low_loss **ALORS** CROITRE

On peut déjà mettre en place l'équité avec TCP. En effet, être TCP-friendly c'est expérimenter le même nombre de perte lors de congestions. La formule suivante $\lambda = (1{,}22 \times MTU / (T \times RTT))^2$ estimée sur par model analytique de TCP [29] citée par [19, 20, 22] peut être une valeur intéressante à utiliser comme seuil high_loss. $\lambda$ représente le taux de perte moyen et T le taux de transmission, la MTU peut être trouvé par la source (RFC1191), le RTT peut être calculé en utilisant les estampilles [19] [8.6 Calcul du RTT via les informations de RTCP] et le taux d'émission est connu de la source. Ceci dit, cette approche a des limitations de part les différentes versions de TCP incluant différents algorithmes de fenêtre de congestion avec différents paramètres de bande passante. Il est donc difficile d'être TCP-friendly avec toutes les versions.

### L'algorithme d'incrément décrément

Maintenant que l'on connaît l'action à mener grâce à la fonction de décision, il faut calculer le taux de transmission que l'on va adopter. Pour ceci, il est courant d'utiliser l'algorithme AIMD d'incrément additif et de décrément multiplicatif [17, 19, 20] qui converge vers un état d'équité stable :

**SI** decision = DECROITRE **ALORS** taux = max (taux x DEC, TAUX_MAX)
**SI** decision = CROITRE **ALORS** taux = min (taux + INC, TAUX_MIN)

Bien entendu, le choix de GAIN, INC et TAUX_MAX, TAUX_MIN est relativement délicat. Pour s'adapter à TCP et particulièrement a sa manière de varier, il faut que le coefficient directeur du taux de transmission en fonction du temps soit inversement proportionnel au carré du RTT. En ayant une fréquence de renouvellement du taux de

transmission égale au RTT, on obtient pour la partie croissance du taux de transmission une pente de INC/RTT = *TaillePaquet*/RTT² si INC = *TaillePaquet*/RTT. Pour obtenir ceci, on suppose que la taille de paquet envoyé est fixe et donc *TaillePaquet* peut être estimé à la valeur de la plus petite MTU sur le chemin du paquet. Ainsi, en prenant INC = MTU/RTT on a une approximation du comportement de TCP en phase croissante. Pour ce qui est de DEC, il est fixé à ½ pour Tahoe et à 0,875 pour Reno. Ensuite, TAUX_MIN et le TAUX_MAX dépendent du flux transporté.

On peut vouloir calculer plutôt le temps qui doit s'écouler entre deux émissions de paquet : l'IPG inter-packet-gap. On a alors avec INC = MTU/C (sachant que C est fixé et est une estimation du RTT) l'algorithme ci-dessous. Remarque, la valeur de INC dans [17] est fixé à 50kb.

**SI** decision = DECROITRE **ALORS** IPG = max (IPG x DEC, MTU/TAUX_MAX)
**SI** decision = CROITRE **ALORS** IPG = min (IPGxC/(IPG+C), MTU/TAUX_MIN)

Il faut tout de même être critique sur cet algorithme. En effet, il diffère de celui utilisé par TCP. TCP fonctionne avec une fenêtre de contrôle et les ajustements de taux ne sont pas uniquement dus à la variation de la taille de la fenêtre. En conséquence notre algorithme diffère du fonctionnement d'une connections TCP et donc le partage de bande passante ne sera pas si équitable que ce que nous espérons [19]. RAP propose en plus un mécanisme d'adaptation de taux à grain fin. Il semble cependant délicat de mettre en œuvre ce genre de mécanisme tant que l'adaptation de taux à gros grain n'est pas encore totalement sûre.

### La fréquence de décision

Comme on l'a vu tout à l'heure, il faut effectuer un renouvellement du taux de transmission avec une certaine fréquence. Il est d'ailleurs suggéré de ne pas renouveler ce taux trop fréquemment [20]. Changer trop souvent mène à une oscillation et trop peu souvent mène à un comportement irresponsable. Pour RTP le problème est assez délicat car il se peut que dans certains cas, aucune information ne soit arrivée bien qu'il soit nécessaire de faire ce renouvellement. On peut dans ce cas choisir de ne pas modifier le taux de transmission ! Dans tous les cas, le renouvellement doit être envisagé tous les RTT.

L'expérience [21] montre que les messages RTCP ont une période trop importante par rapport au temps pris par un acknlowledgement de TCP. Ainsi, si l'on n'y prend garde, la décision de décroître son taux après des pertes est prise de manière bien trop lente et donne un comportement non TCP-friendly.

Pour ce qui est du taux de perte, on ne peut pas faire autrement que d'attendre le prochain paquet RTCP. Par contre le RTT peut très bien être évalué d'une autre façon [8.6 Calcul du RTT via les informations de RTCP] et permettre ainsi de trouver le taux de perte théorique de TCP et de décider s'il est temps de DECROITRE. L'expérience [21] se base sur une estimation du RTT après des pertes pour mettre à jour le taux de transmission tous les RTT sans pour autant avoir un renouvellement aussi fréquent du taux de perte.

## *3.2 l'adaptation de qualité*

Le module précédent permet de déterminer le débit d'émission en sortie de la source. Il est d'abord nécessaire de repartir la bande passante entre les différents média : audio, vidéo… De manière générale pour les deux médias présents, il est nécessaire de maintenir une

bonne qualité audio. En effet, les expériences menées sur les logiciels de conférence ont montré que les utilisateurs coupaient la vidéo pour favoriser l'audio [26], quand le service global se dégradait. De plus, le débit audio est beaucoup plus faible et donc si l'on doit faire varier le débit global, il est plus intéressant de s'en prendre à la vidéo dans un premier temps. Il est aussi nécessaire de partager la bande passante avec les flux de contrôle (RTCP) et les mécanismes de recouvrement d'erreur.

Ici, le principe est de pouvoir maintenir un débit imposé par le contrôle de congestion. Cela peut paraître étonnant mais il est possible de faire varier le débit d'une application à débit constant. Il existe plusieurs solutions pour dégrader la qualité. Il y a le codage au vol du flux ou aussi appelé codage adaptatif, qui une fois le débit de codage choisi, ne le changera pas durant un intervalle assez grand. La charge CPU nécessaire n'est pas négligeable. Il y a la possibilité de stocker le même flux mais dans une résolution différente et donc d'interchanger quand cela est nécessaire. Cela pose le problème de capacité de stockage mais aussi de permutation brusque de qualité. Enfin il y a la possibilité d'émettre plus ou moins de couche dans le cadre d'un codage du flux de manière hiérarchique (sous-bande). Cette approche à plusieurs avantages : elle est plus intéressante pour les caches dans le sens ou le débit en fonction du client est adaptable par rapport à la première solution, et le stockage est moindre que dans la deuxième solution. De plus le nivelage de l'information permet une adaptation de qualité douce et sélective[18].

Pour un flot donné, nous avons donc un débit $T_{max}$ maximum imposé par le module de contrôle de congestion. Possédant un codage en sous bande, on envoie le maximum de couches de manière à ne pas dépasser le débit $T_{max}$ dans le cas d'une gestion du débit par l'émetteur. Dans le cas d'une gestion du débit par le récepteur, on joint les groupes sous bandes de telle sorte que le débit ne dépasse pas $T_{max}$. L'algorithme [22] est donc en supposant que chaque couche a le même débit $T_i$ :

- Trouver L tel que $\sum_{i=1}^{L} T_i \leq T_{max}$
- Envoyer ou Joindre les L premiers groupes

Dans tous les cas, l'envoi de chaque couche doit être décalé (*shift parameter*) de manière à ne pas avoir une émission brusque et importante au même moment. Les expériences de [22] montrent que la qualité de réception (fluidité) est bien meilleure en utilisant cet algorithme qu'avec les applications traditionnelles. L'algorithme peut débuter immédiatement, en effet, initialement la valeur de $T_{max}$ peut être de TAUX_MIN [L'algorithme d'incrément décrément]. Cette valeur doit correspondre à l'émission de la couche de base. La taille des paquets doit être la plus grande possible sans dépasser la MTU. En effet, l'*overhead* des en-têtes RTP/UDP/IP est assez importante. On peut d'ailleurs compresser ces en-têtes comme décrit dans le *draft* [23].

La solution présentée ci-dessus peut se voir améliorer par des techniques de lissage*,* *smoothing*, de la variation du nombre de couches. En effet, un des effets de la technique précédente est de supprimer l'envoi d'une couche dès que le taux imposé n'est plus suffisant. Le récepteur ne peut plus utiliser la couche. Il est possible qu'ensuite le taux redevienne suffisant pour ré émettre cette couche. Un mécanisme de *bufferisation* des couches permet d'éviter cette brève variation de qualité à la réception. Le lissage est quelque chose d'important car les utilisateurs ne tolèrent pas les variations rapides de qualité. L'article [24] s'attaque au problème et obtient des résultats très intéressants. Le principe est de ne joindre une nouvelle couche que si le débit résultant des couches est inférieur au débit moyen et si la quantité de données bufférisées est suffisante pour survivre à K réduction du débit de transmission. La réduction à proprement parler est la division par deux du débit de

transmission. Cette réduction est le mécanisme mis en œuvre dans TCP lors de la perte de paquets.

Il faut remarquer que l'utilisation de *buffer* de réception permet en outre de contrer la gigue. Une approche classique est de débuter le jeu du média qu'une fois que les *buffer* du client sont suffisamment remplis. En effet, de cette manière les variations aléatoires du temps de transmission peuvent être absorbées.

### *3.3 le contrôle d'erreur*

Le mécanisme de recouvrement d'erreurs est nécessaire comme le montre l'expérience menée sur le *Mbone* dans [19] ou même si les mécanismes d'adaptation de débit de transmission sont mis en place, les pertes expérimentées sont assez importantes. Ceci s'explique par le nombre important d'utilisateurs. Palier à ces pertes est donc très important. Il existe beaucoup de techniques de recouvrement d'erreur. Il y a la redondance, la retransmission, l'entrelacement et enfin les mécanismes de FEC [30]. Les pertes de paquets lorsqu'elle surviennent sur Internet surviennent de manière très discrète. En effet, la congestion ne dure qu'un court instant. Le nombre de paquets perdus est souvent de un. [28, 29]. Dans ce cas, il est assez simple de contrecarrer l'effet de perte. Le mécanisme de recouvrement doit être choisi en fonction du niveau de fiabilité requis par le codage de l'application, en fonction des délais tolérables et du taux de perte supposé ou bien mesuré.

Dans le cadre d'un serveur vidéo *unicast* l'article [18] adopte la technique de la retransmission. Le mécanisme de FEC adaptative est mieux adapté dans le cas de l'utilisation de RTP/RTCP. En effet, on n'est pas nécessairement prévenu de la perte d'un paquet et de plus la retransmission est plus consommatrice en bande passante. La retransmission nécessite aussi de stocker les paquets émis tant que l'acquittement n'est pas parvenu. Ceci ajoute encore de la complexité au mécanisme. Ceci dit, l'article [18] utilise le protocole RAP qui prévient automatiquement de la perte d'un paquet et qui ne produit que très peu de surcharge pour le flux de contrôle. Les défauts de RAP sont tout de même de ne pas être un protocole *multicast* et de ne pas proposer toutes les informations d'identification et donc de sécurisation qui sont présentes dans RTP.

L'expérience [19] montre que la gigue n'est pas un indicateur préventif à la perte. Ceci dit, le RTT semble plus intéressant dans la mesure ou une rapide augmentation de celui-ci indique un début de remplissage de file d'attente. Dans ce cas on peut anticiper les pertes en décidant de renforcer le mécanisme de recouvrement d'erreur. Tout comme dans le mécanisme de contrôle de congestion [3.1 Le contrôle de congestion], nous pouvons imaginer un mécanisme de contrôle de variation de RTT qui nous indique le niveau de recouvrement d'erreur que nous devons mettre en œuvre. L'article [19] propose quant à lui de faire varier le nombre de paquets consécutif sur lequel porte le mécanisme de FEC en fonction du taux de pertes. Une idée est donc de faire varier le nombre de paquets sur lesquels on applique la FEC, en fonction de la variation du RTT. Le mécanisme de FEC peut être l'application de XOR sur les k derniers paquets. Il est tout de même nécessaire de ne pas perdre de vue que la bande passante doit être répartie entre l'émission des données, le flot de contrôle RTCP, et le flot de FEC. Les données devant bien entendu recevoir le plus de bande passante possible.

Le mécanisme de FEC tout comme le flot de contrôle sont à considérer comme des informations de signalisation. Ces informations sont donc à considérer comme prioritaire. Dans une architecture à différentiation de service, les paquets transportant celles-ci doivent être marqués prioritairement. Pour ce qui est du transport de MPEG, un choix possible est de considérer que seul les paquets transportant des images B sont à protéger par des FEC. En effet, les images I et P sont plus protégées par l'architecture. Mais dans le cas de congestions

importantes, ce choix peut s'avérer inintéressant. Un autre choix peut être de protéger les images I plus que les autres. Dans tous les cas, il faut éviter de trop augmenter l'utilisation de la bande passante.

### *3.4 Le cas du multicast*

Tous ce qui a été présenté jusqu'à présent concerne plutôt l'unicast et les adaptations des mécanismes précédents au *multicast* ne sont pas immédiates. La principale question est comment adapter le débit source et les mécanismes de FEC en fonction des différents utilisateurs. Plusieurs solutions ont été proposées et la *Receiver Driven Layered Source* (RLM) est finalement une manière de répartir certaines idées et techniques présentés pour l'*unicast*. Dans ce cas, la source émet sur l'arbre *multicast* le flux vidéo complet. Les récepteurs joignent alors un ou plusieurs groupes correspondant chacun à une des couches du flux vidéo. Le récepteur peut donc adapter sa réception en fonction de la congestion locale du réseau. Ici, ce n'est plus la source qui est responsable de la prise en compte de la congestion mais bien le récepteur. Le mécanisme de *pruning* permet alors d'alimenter en paquets les liens nécessaires pour atteindre les récepteurs actifs [19]. Certains problèmes restent à résoudre. Il est nécessaire de réguler la charge créée par les paquets *join* en partagent l'information de souscription avec les autres récepteurs : *shared learning*. Ceci dit, il est difficile d'estimer l'efficacité du *shared learning* en l'absence de connaissance sur l'arbre *multicast*. Le temps de convergence à un état stable ainsi que le choix des couches à ajouter ou a supprimer restent des problèmes ouverts [19]. Une solution proposée par [19] est de baser sur une estimation du débit de transmission TCP-friendly pour joindre les couches [3.1 Le contrôle de congestion].

Il existe aussi les vidéos *gateways* qui font du codage au vol et qui sont actuellement présentes dans le *Mbone*. Le problème principal est de trouver un lieu approprié pour mettre ce mécanisme en place. Certains schémas ont été proposés pour mettre en place des lieux dynamiques.

# 4 RTP/RTCP

Nous abordons maintenant la couche transport des données avec le protocole RTP/RTCP. Ce protocole est intéressant dans la mesure ou c'est un standard qui est utilisé par les applications actuels. De plus, un ensemble de *draft* pour le transport de flux multimédia via RTP sont en cours d'écriture et donc permettent de donner des idées sur les techniques de transport à utiliser.

Dans un premier temps, nous présenterons RTP/RTCP. Ensuite, nous discuterons de la validité des informations RTCP. Puis, nous aborderons les techniques de transport de MPEG déjà existantes. Enfin, nous aborderons les techniques de transport en couches qui sont les plus adaptées à la solution que nous proposons pour un service sur *DiffServ*.

## *4.1 Présentation de RTP/RTCP*

### RTP

RTP, *Real Time Protocol,* a été conçu par l'Internet Engineering Task Force (IETF) pour le transport de flux multimédia tel que la visioconférence, la vidéo à la demande ou bien la musique à la demande. C'est donc un protocole de transport mais il se présente plus comme une surcouche d'UDP. En effet, RTP n'a pas de notion de connections, nécessite une prise en charge de la segmentation par les couches inférieures et il n'offre pas de mécanisme de fiabilité. Il est d'ailleurs typiquement implémenté comme faisant partie de l'application et non du système d'exploitation. Quant à la notion de temps réel (*Real Time Protocol*), elle n'existe pas puisque aucun mécanisme assurant le respect d'échéance n'est mis en place. Ceci dit, RTP offre aux couches supérieures, c'est à dire à l'application, des fonctionnalités permettant de garantir la qualité de service. Celle-ci sont l'identification du type de format transporté, le séquencement ainsi que l'estampillage temporel des paquets, et aussi différentes statistiques sur la qualité de la session (taux de perte, RTT, gigue…). Ainsi, RTP comporte deux partie bien distinctes, la partie flot de données qui permet le transport de l'audio de la vidéo etc. et la partie flot de contrôle via RTCP. Les paquets de données et de contrôle utilisent deux ports consécutifs, avec le port pair pour les données et le port impair pour le flot de contrôle [17].

RTP grâce à l'ajout d'un en-tête permet finalement de fournir les informations n'étant pas présentes sur UDP et étant nécessaires au transport de flux multimédia. L'identification est effectuée grâce à un numéro unique ainsi que le contenu du paquet RTP. Un tel mécanisme permet de jeter les paquets indésirables qui pourraient perturber une session. Le chiffrement ajoute la confidentialité. Le numéro de séquence permet au récepteur de reconstruire le flux de manière chronologique et permet à celui-ci d'estimer le nombre de pertes. L'estampillage temporel permet la synchronisation entre plusieurs flot et le calcul temporel de gigue.

### RTCP

L'intérêt du flot de contrôle est d'exploiter les données des paquets RTP pour avoir des statistiques sur l'état du réseau (RTT, gigue), et des participants à la session (taux de pertes). En effet, chaque participant envoie périodiquement des rapports à tous les autres

participants. Plus précisément, chaque émetteur envoie des rapports SR, *Sender Report*, contenant la quantité de données émises et l'estampillage temporel du rapport. Chaque récepteur envoie un rapports RR, *Receiver Report*, pour chaque flot reçu, contenant la fraction de données perdue, le dernier numéro de séquence reçu, la gigue, l'estampille du dernier SR reçu pour ce flot et le temps écoulé entre le dernier SR reçut et l'émission du RR. Des informations supplémentaires telles que la description du participant peuvent être émises. Il faut bien voir qu'un participant peut à la fois être émetteur et récepteur (visioconférence).

En outre, pour des raisons de *scalabilité*, RTCP régule son débit pour n'utiliser qu'un faible pourcentage de la bande passante de session. Couramment, le canal de contrôle est limité à 5% de la bande de session c'est-à-dire à la bande passante allouée a l'ensemble des participants pour la conférence, le film… RTCP estime le nombre d'utilisateurs, et le nombre d'émetteurs, grâce aux informations des paquets RTCP. Le débit du flux de contrôle s'autorégule alors à partir de ces informations. Typiquement, la période d'un émetteur est (1) et la période d'un récepteur est (2). De plus, il est recommandé que la période d'émission des paquets RTCP soit supérieure ou égale à 5 secondes.

$$T = \frac{NombreEmetteur \times TailleMoyennePaquetRTCP}{25 \times 5\% \times BandePassanteDeSession} \quad (1)$$

$$T = \frac{NombreEmetteur \times TailleMoyennePaquetRTCP}{75 \times 5\% \times BandePassanteDeSession} \quad (2)$$

Ainsi, l'interface RTP/RTCP en plus de proposer le transport de données permet l'accès à des indicateurs de congestion tel que le taux de perte de paquets, la gigue, le RTT. Comme on l'a vu, ces informations peuvent être utilisées pour les applications *streaming* adaptatives. Le problème majeur étant comme spécifié précédemment que ces informations sont périodiques et que leur fréquence est limitée par la nécessité d'être *scalable*.

### *4.2 Réflexion sur RTCP*

En supposant que l'on a un rapport RTCP par RTT cela fait 20 octets d'en-tête IP, 8 octets d'en-tête UDP et 32 octets RTCP (sans information CNAME). Avec un RTT de 100 ms cela fait un débit 4,8kb/s [25]. Un tel débit est non négligeable cependant, il doit toujours être inférieur au 5% de bande passante alloué. Ainsi, sur une connexion modem 56kb/s, le débit utilisable par le flot de contrôle n'est plus que de 2,8kb/s. Dans ces conditions il est probable que l'émetteur n'aura pas un rapport RTCP par RTT. L'algorithme AIMD ne s'adapte alors plus très bien aux variations de la bande passante du réseau. Il est à remarquer de plus que certains délais sont directement introduits par le matériel. Dans le cas du modem, une compression des données est automatiquement effectuée ce qui prend environ 100ms [26] ! Dans l'exemple précédent le paquet mettrait donc environ 109 ms à être émis ! Ce délai est assez grand pour une information de feedback ! Sur des liens lents à faible bande passante, le temps d'inter arrivé entre deux RTCP successif peut excéder sept secondes ce qui peut mener à des réponses plus lentes des mécanismes d'adaptation [26]. Ainsi, un des problèmes de RTCP est de ne pas être suffisamment réactif sur des liens à faible débit ou lorsqu'il y a beaucoup de participants.

Un deuxième inconvénient est son aspect uniquement périodique. En effet, les informations sur le taux de pertes et la gigue sont émises à intervalle régulier. Cet intervalle comme on l'a vu précédemment peut être bien trop long pour la pertinence de l'information. Il est donc difficile de prendre en compte les pertes sporadiques de paquet. L'idée principale

est de prévoir ces pertes pour pouvoir y remédier. Il a été remarqué que l'augmentation de la gigue comme signal de future perte de paquet n'a aucun lien de cause à effet [19]. Cependant, l'augmentation du RTT semble signaler un remplissage des files d'attente traversées et donc une possible congestion. L'utilisation du RTT comme signal de congestion peut donc être envisagé sereinement si la périodicité des rapports RTCP est petite. Hors a priori, la périodicité des rapports RR n'est pas petite sur les liens à faible débit. Le protocole RAP propose une périodicité du flux de contrôle qui est similaire au RTT. Ainsi, on suit bien mieux l'état du réseau. L'article [26] propose quant à lui un feedback client – serveur uniquement lorsqu'il y a perte de paquet. On a donc ici un flux de contrôle non périodique. Les mécanismes d'identification et *multicast* sont par contre non présents dans les deux solutions précédentes.

## *4.3 Le transport de MPEG*

### Partitionnement du flux MPEG

[31] [32] proposent trois approches au transport de paquet RTP contenant du flux MPEG1/MPEG2. La première approche est plutôt axée sur une interopérabilité avec le codage MPEG. En effet, pour MPEG1, le canal de données sera utilisé pour le flot *System* qui encapsule les éléments vidéo et audio, *Elementary Stream*. Pour MPEG2, un canal sera réservé au flot *Program* ou au flot *Transport*. La deuxième approche fournit une plus grande opérabilité avec Internet. Ici, les problèmes *System* sont laissés de coté et l'on s'intéresse aux flots élémentaires (ES). Deux canaux sont alors définis. L'un correspond au flux vidéo et l'autre au flux audio. La dernière méthode propose d'utiliser un unique canal pour les flux élémentaires. Ces approches peuvent se voir augmenter de techniques de recouvrement d'erreur et de resynchronisation [31].

De manière générale, la première méthode ne prend pas suffisamment en compte le flux transporté et est donc moins performante vis-à-vis de l'occupation réseau et vis a vis des pertes d'informations. Ainsi, le mécanisme d'empaquetage est réalisé trois fois : d'abord dans *Packetized Elementary Stream* (PES), ensuite dans *Transport Stream* (TS), puis dans RTP. La surcharge d'information est donc assez importante. En ce qui concerne la perte de paquets, elle n'est pas sans conséquences car elle peut rendre désuet les informations des paquets précédent et suivant. D'un autre coté, cette méthode ne nécessite pas de traitement sur le flux MPEG (pas d'extraction de flots élémentaires). De plus, l'estimation et la réduction de la gigue peuvent être réalisés grâce aux informations temporelles tel que la *Program Clock Reference* (PCR), le *Decoding Time Stamps* (DTS) ou le *Presentation Time Stamps* (PTS). Ceci dit, le *parse* de MPEG s'avère nécessaire sur une architecture à différentiation de service pour pouvoir attribuer des priorités aux paquets. Ainsi, il suffit pour l'application serveur d'améliorer un peu ce parse pour s'adapter naturellement à la deuxième et à la troisième méthode.

Dans la deuxième et troisième méthode, les règles d'empaquetage respectent ce que l'article [28] préconise. Les informations tel que les en-têtes ou les tranches d'image doivent être totalement contenu dans un paquet. Ainsi, la perte d'un paquet n'introduit pas la perte d'information autre que celle contenu par ce paquet. On a donc une réduction des dépendances entre paquets et donc plus de robustesse aux pertes de paquets. De plus, la charge imposée au réseau est allégée par le traitement de *parse* effectué sur le flux MPEG. En effet, les informations *System* sont supprimées. En plus de ceci, on ajoute des redondances d'information tel que le *Sequence_Header*, le *GOP_Header* et le *Picture_Header* qui est

régulièrement introduit dans les paquets RTP. Cette redondance permet de recevoir et décoder à un endroit arbitraire du flux ainsi que de faire du recouvrement d'erreur en reconstruisant les en-têtes perdus. Par rapport à la première solution, le PTS est similaire à l'estampille RTP mais le DTS et les mécanismes de contrôle de *buffer* ne sont pas définis [33].

Les solutions deux et trois ont toutes deux des avantages et des inconvénients. La troisième solution provoque moins de surcharge, possède une synchronisation implicite audio vidéo, possède une taille de *buffer* globalement meilleure et le serveur expérimente moins de ports alloués. D'un autre coté, cette solution nécessite plus de traitement sur un mixer ou un translateur qui désirerait diminuer la qualité vidéo ou audio ou bien tout simplement supprimer un de ces flots. Elle pose aussi problème pour les applications type conférences où les mixers peuvent avoir à regrouper des flux audio ou vidéo issus de plusieurs sources. Dans le cadre de la VOD, la troisième solution est par contre plutôt bien adaptée.

Création de différentes couches

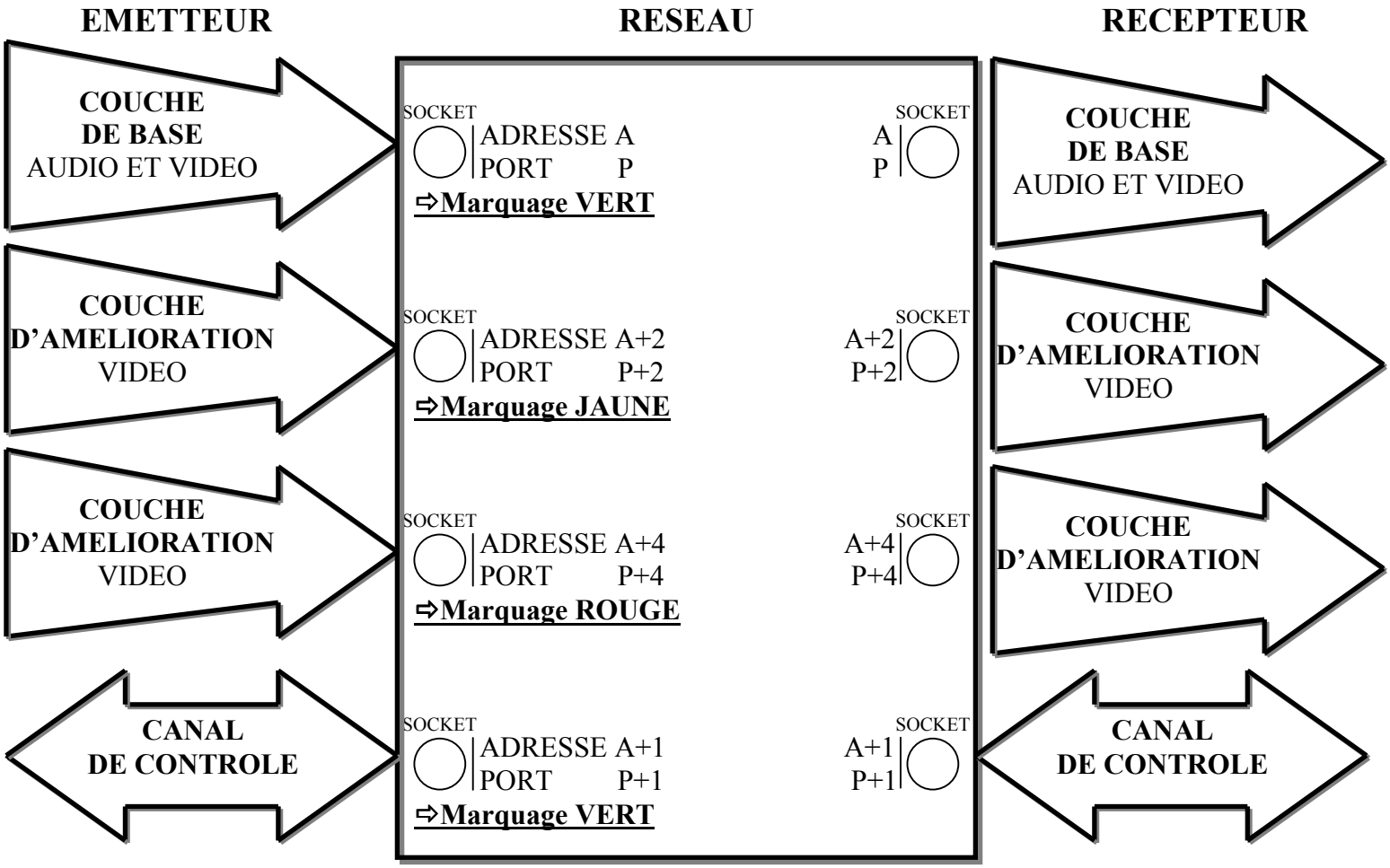



**Figure 12 Une session RTP avec trois couches de données (cas *multicast*)**

Comme on l'a vu précédemment, l'adaptation de qualité d'un flux multimédia passe par une régulation du débit de la source. Que ce soit dans l'*unicast*, dans le *multicast*, dans des applications type conférence (plusieurs sources) ou bien dans des applications type vidéo ou musique à la demande, il semble que l'utilisation du codage en multicouche soit le plus intéressant. Avec un flux multimédia de type hiérarchique, il est alors largement envisageable d'utiliser plusieurs canaux RTP pour une même session. Chacun des canaux RTP se voit alors attribuer un niveau de couche sachant que la couche zéro peut très bien contenir les éléments

vidéo couche de base et audio (troisième solution). Les couches suivantes peuvent être composées uniquement de vidéo (deuxième solution). La couche de base dans une architecture à différentiation de service est bien entendue marqué comme prioritaire (couleur Verte de la classe AF) et les couches suivantes sont marquées moins prioritaires (Jaune et Rouge). Ainsi, l'adaptation dirigée par la source ou bien par le récepteur dans le cas *unicast* ou *multicast* est parfaitement viable.

[34] proposait déjà une solution similaire. Leur solution consiste à créer pour chaque couche du flux vidéo un canal de données et un canal de contrôle. Le nombre de ports alloués est alors assez important. De plus, la nécessité de gérer les informations de contrôle de chacune des couches est assez fastidieuse (une source aura un SRC différent pour chacun des canaux de contrôle). Une solution plus sérieuse consiste à allouer un seul canal pour le contrôle. Bien sur, cela nécessite une légère modification de RTP puisqu'une session comporte alors plusieurs canaux de données. Ceci dit, c'est bien le type du *payload* qui détermine les caractéristiques de la session. On peut très bien envisager tout en restant pratiquement conforme à RTP de définir un nouveau *payload* qui induit plusieurs canaux de données. Le premier canal de données correspondrait à quelque chose de similaire a [32]. Ce canal serait présent sur un port P pair a une adresse A. L'unique canal de contrôle serait sur le port P+1 à l'adresse A+1. Ensuite les adresses paires et les ports pairs suivants seraient alloués aux autres couches. L'intérêt majeur d'avoir une adresse différente par couche et d'avoir différents canaux est de pouvoir faire de la régulation de flux en joignant ou quittant les couches [3.4 Le cas du multicast]. Les différents ports s'expliquent plutôt pour des raisons de déficiences des systèmes d'exploitation [34]. Dans le cas *unicast*, on peut très bien utiliser une adresse unique et différents ports. La figure 12 montre un exemple d'une session RTP à trois canaux de données et un canal pour RTCP dans le cas *multicast*.

Ainsi, dans le cas *unicast*, il est préférable de réguler le débit depuis l'émetteur en envoyant plus ou moins de couches. Cette solution est préférable car aucun mécanisme de type *join* n'est à mettre en place. On envoie donc l'ensemble des données à une même adresse destination et l'on différentie les couches par différents ports.

Dans le cas *multicast*, le débit est régulé par les récepteurs et celui ci joint ou quitte les couches. Cette solution est préférable car chacun des récepteurs peut ainsi gérer son débit comme il le souhaite et de plus on s'appuie sur les techniques *multicast* existantes. Dans ce cas, on a donc une adresse unique et un port unique par couche.

En ce qui concerne les mécanismes de recouvrement d'erreur, sur une architecture à différentiation de service avec la classe AF, le mécanisme de recouvrement d'erreur n'est pas forcément nécessaire. En effet, la classe AF permet la protection des informations les plus utiles et le codage hiérarchique minimise l'effet de la perte des informations les moins utiles. Ainsi, l'ajout d'un mécanisme comme les FEC introduit une surcharge réseau supplémentaire qui n'est pas forcément intéressante. Ceci dit, la question reste ouverte. Dans le cas d'un ajout de mécanisme tel que la FEC, une autre question est : quelle couche protéger ? A priori, la couche de base est la plus importante mais la protection est déjà d'une certaine manière assurée par son marquage de couleur verte. Cependant, un scénario extrême voyant tous les paquets marqués rouge et jaune supprimés, ainsi que les paquets verts ayant régressé en paquets jaunes ou rouges démontrerait probablement l'utilité des FEC sur les paquets jaunes. On peut aussi envisager de mettre en place les FEC sur plus d'une couche tout en gardant à l'esprit la surcharge réseau introduite. Comme [19] le propose, le mécanisme de recouvrement d'erreur peut évoluer en fonction d'un paramètre tel que le taux de perte, le RTT (ou les deux !). Dans tous les cas, si mécanisme de recouvrement d'erreur il doit y avoir, celui-ci doit être inclus dans la librairie RTP/RTCP.

### Taille des paquets RTP

En ce qui concerne la taille du contenu d'un paquet RTP il convient de prendre en compte la taille RTP/UDP/IP qui doit être inférieur a la plus petite MTU du chemin emprunté pour éviter la fragmentation. Pour exemple, un réseau Ethernet possède une MTU de 1500 et un réseau X25 à une MTU maximale de 576 [35]. L'en-tête RTP fait sans ajout 12 octets, l'en-tête UDP est fixe et fait 8 octets, l'en-tête IP sans option fait 20 octets. On a donc pour un paquet RTP, 40 octets d'en-tête à prendre en compte en plus de la taille du contenu du paquet RTP. Chaque *Elementary Stream* doit être complètement contenu dans un paquet. Ainsi, la taille minimale acceptable pour la MTU est de 261 octets (cela correspond au transport de quant_matrix_extension) [31] ce qui fait un *datagram* de 301 octets. Il est d'ailleurs possible de mettre plusieurs Unités Elémentaires dans un même paquet RTP, la seule contrainte étant d'avoir un *datagram* de taille inférieure à la plus petite des MTU sur le chemin emprunté.

Ceci dit, il peut arriver que l'on dépasse largement cette MTU. En effet, un slice d'une image I peut faire que le paquet RTP/UDP/IP soit plus grand que la plus petite MTU sur le chemin suivi (1945 octets pour le plus grand slice dans l'expérience de [36]). Dans ce cas, il est nécessaire de fragmenter soit même l'unité élémentaire. Il peut aussi être envisagé si c'est possible de réduire la qualité de l'image avec au lieu de 44 macroblocs par slice passer à 11 macroblocs ou bien 4 macroblocs par slices. Dans le cas de 11 macroblocs par slice, la perte estimée en efficacité de compression est de 3% et la perte en qualité d'image est estimée à 0,2 dB. Dans le cas de 4 macroblocs par slice, la perte en compression est estimée à 12% et la perte en qualité d'image est estimée à 1dB (cf. expérience de [36]).

# 5 L'architecture à différentiation de services

Dans le cadre de l'amélioration des services Internet, l'IETF[2] propose une architecture à différentiation de services [37]. Cette architecture appelée *DiffServ* se situe au niveau de la couche Réseau de l'OSI, c'est à dire qu'elle fonctionne au même niveau que IP. Ainsi, c'est surtout les mécanismes de routage qui sont améliorés. L'architecture à différentiation de services permet de hiérarchiser les flux, c'est à dire les paquets, entre eux. Ainsi, certains flux sont favorisés par rapport à d'autre. Par exemple, un flux marqué peut expérimenter un délai de transmission plus court ou bien moins de pertes de paquets qu'un flux non marqué.

C'est donc l'aspect routage avec marquage des paquets qui induit la notion de service proposé par le fournisseur d'accès. En effet, le client se verra dans l'obligation de signer un contrat, pour obtenir la possibilité de voir router ses paquets marqués, c'est à dire pour obtenir l'accès à un service. La notion de qualité de service est ainsi pour une grande part un des apports de *DiffServ* à Internet et en particulier au *backbone*[3].

Dans un premier temps nous présenterons l'intérêt de l'architecture à différentiation de services. Ensuite, nous exposeront les différents services proposables par un fournisseur d'accès. Puis, nous expliqueront les principes de fonctionnement de l'architecture. Enfin, nous discuterons de l'intérêt de la classe AF pour les flots multimédia. Ce dernier point justifie le service de transport MPEG Audio sur DiffServ proposé dans [6 Réalisation].

## *5.1 Introduction*

L'un des problèmes majeur des réseaux IP, et en particulier Internet est une absence totale de qualité de service, *Quality Of Service*, *QoS*. Initialement, le fait d'avoir une répartition de la bande passante identique entre tout les flux, c'est-à-dire ne disposer que d'un service : le service *Best Effort*, n'était pas tellement problématique. En effet, la majorité des flots étaient de type TCP et donc régulaient leur débit en fonction de l'occupation réseau. Il n'y avait donc pas de risques d'effondrement du réseau, c'est-à-dire d'une saturation entraînant la suppression de tous les paquets arrivant.

Actuellement, on assiste à l'émergence des flux multimédia via le protocole UDP. Les applications impliqués ont des contraintes d'échéances temporelles. Il devient nécessaire pour le bon fonctionnement de ces applications d'introduire des mécanismes de type réservation de bande passante. D'autre part, UDP a tendance à accaparer toute la bande passante. Ceci s'explique par le fait que TCP dispose d'un mécanisme de régulation de son débit et pas UDP. Ainsi, avec l'introduction de classes de services on peut protéger les flots entre eux.

La première approche de l'IETF a été de proposer la mise en place de mécanisme de type réservation de bande passante. Le principe était très similaire à ce que fait X25 ou bien ATM. Le problème est qu'à grande échelle, les routeurs ne peuvent plus assurer à la fois leur rôle premier, qui est le routage, et le mécanisme d'allocation de circuit. Chaque circuit alloué nécessite de la place mémoire et c'est un facteur non *scalable*[4]. De plus, la stratégie de contrôle de la quantité de ressource réservée aux utilisateurs est complexe. Cette technique de réservation est appelée *IntServ* et est abandonnée dans le contexte de réseaux de cœur depuis 1996-1997. Actuellement, on se tourne plutôt vers un mécanisme plus souple et plus simple qui ne nécessite pas la modification de l'existant : *DiffServ*.

---

[2] *Internet Engineering Task Force*, www.ietf.org

[3] *Bakcbone* : épine dorsale de l'Internet; lien à très haut débit voyant passer l'ensemble du trafic Internet.

[4] Quelque chose est *scalable* signifie qu'il passe le facteur d'échelle.

La différentiation de service aborde le problème de qualité de service via l'accord de privilèges à certains flux de données. On a donc une qualité de service qui n'est pas aussi stricte que *IntServ* qui lui assure une réservation de bande passante. Ces privilèges sont réglementés et les flux agressifs sont pénalisés. La réglementation des flux est mise en place, grâce à un profil de trafic. Tant que l'utilisateur reste dans le profil, ses paquets recevront une certaine qualité de service. En résumé, l'architecture *DiffServ* propose une qualité de service moindre que *IntServ* mais qui est beaucoup plus simple à mettre en place.

## *5.2 Les services proposés par l'architecture à différentiation de services*

### Présentation de ces trois services

Actuellement, on peut distinguer trois services dans l'architecture à différentiation de service. Il y a le service *Default forwarding* qui utilise un trafic de type *Best Effort*. Celui-ci correspond en fait au comportement actuel d'Internet, dans lequel, les routeurs desservent l'ensemble des flux approximativement de la même manière. Il n'y a pratiquement aucuns mécanismes privilégiant plus un flux que d'autre. Ainsi, la classe *Class Selector* [39] a été créé pour avoir un comportement similaire au comportement des paquets actuels. Cette compatibilité s'exprime de la façon suivante : un paquet marqué par une valeur spécifique du *Class Selector* doit recevoir une durée de relais inférieure à un paquet marqué par une valeur inférieure du *Class Selector*. Dans l'architecture à différentiation de services, les flux marqués *Best Effort* sont traités comme les moins important sur les routeurs. Ainsi, les flux *Best Effort* se partagent ce qui leur reste de bande passante après que les deux autres services se soient servis !

Le deuxième service est le *Better Than Best Effort* ou bien *Olympique* qui se base sur la classe *Assured Forwarding* qui est plus adapté au flux multimédia. Ce service à l'avantage de voir ses paquets desservis prioritairement dans les files d'attentes par rapport aux paquets *Best Effort*. Ainsi, les paquets marqués *Assured Forwarding* expérimentent moins de temps d'attente et moins de pertes que *Best Effort*. Un autre avantage de ce service est de permettre la nivelation des informations transportées. En effet, à l'intérieur d'une classe AF, il existe trois marqueurs que l'on nomme par les couleurs rouge, jaune, vert. Le paquet rouge a la plus grande probabilité de suppression alors que le vert a la moins grande. Lors d'une congestion dans la classe AF, c'est les paquets rouges qui seront d'abord supprimé, puis viendront les jaunes et enfin les paquets verts.

Le dernier service est le *Premium* qui se base sur l'*Expedited Forwarding*. Ce service est celui qui expérimente la plus grande qualité. En effet, ses paquets sont immédiatement desservis dans les files d'attentes. Ainsi, il y a peu de délais de transmission, pratiquement pas de gigue car l'attente dans les files d'attente est faible voire nulle. Ceci dit, ce service est limité à un débit d'au maximum de 10% de la bande passante. Cette limitation est nécessaire pour éviter d'accaparer le réseau. Un mécanisme de réservations et de signalisations doit donc être mis en place. Ainsi, le service *Premium* s'apparente à une ligne physique indépendante, on dit aussi qu'il est assimilable à une ligne virtuelle louée.

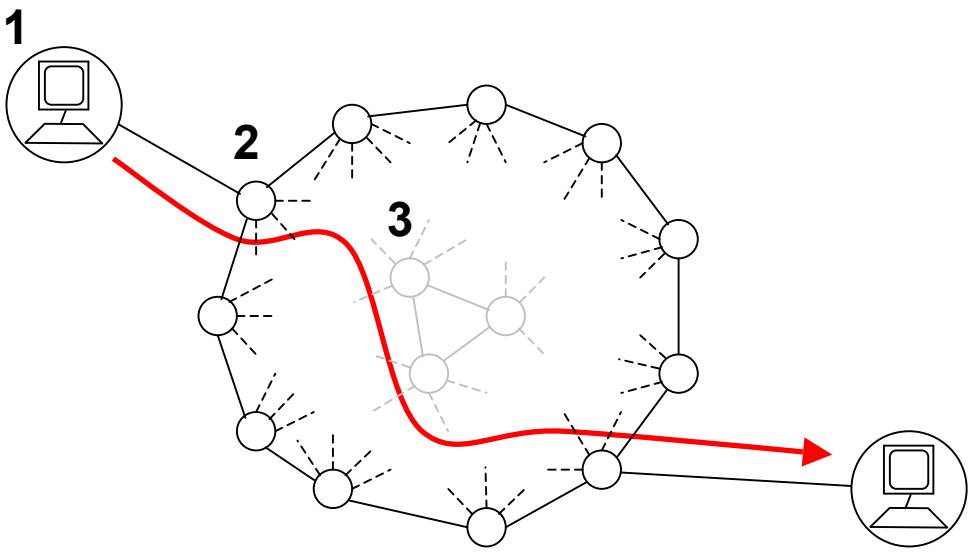
Le tableau ci-dessus [Figure 13] donne quelques exemples d'utilisation des différents services qu'un fournisseur d'accès peut proposer [38]. Il faut noter qu'à priori, le service *Premium* sera facturé beaucoup plus chère que le service *Better Than Best Effort* ou bien *Default Forwarding*. Ainsi, on constate que l'utilisation du service *Premium* vaut la peine pour des applications ayant des échéances temporelles fortes tel que la visioconférence ou la téléphonie. Les applications à contraintes temporelles moindres, et pouvant tolérer quelques

pertes sont plus adaptées au service *Better Than Best Effort*. Enfin le reste des applications peuvent continuer à expérimenter un service de type *Best Effort*.

| Classe de service | Catégorie | Type d'application |
|---|---|---|
| Expedited Forwarding | 1 classe | Téléphonie, visioconf… |
| Assured Forwarding | 4 classes<br>3 niveaux de suppression par classe | Vidéo, audio, transactions financières… |
| Default Forwarding | 1 classe<br>(Best Effort) | Messagerie, Web, ftp… |

**Figure 13 Service et type d'applications associées**

### *5.3 Les mécanismes de fonctionnement*




1 Client : Effectue un marquage de son paquet c'est-à-dire positionne l'octet DS qui détermine sa classe (*Best Effort*, *Assured forwarding* ou *Expedited Forwarding*).
2 Routeur d'entrée : Vérifie que les paquets provenant de 1 sont conformes (marquage autorisé, débit autorisé…) et si nécessaire change le marquage du paquet ou le détruit.
3 Routeur internes : Mettent en œuvre la différentiation de services grâce au marquage des paquets effectué par 1 ou par 2. Ainsi, certain flux reçoivent plus d'avantages que d'autre.

**Figure 15 Fonctionnement général de *DiffServ***

De manière technique, l'architecture *DiffServ* comporte trois modules distincts : l'émetteur, le routeur d'entrée du domaine et les routeurs internes au domaine [42]. L'émetteur injecte son trafic dans le réseau et sa seule obligation est de marquer son flot de données. Le routeur d'entrée du domaine est chargé quant à lui de vérifier la conformité de chaque flux. Cette conformité s'exprime en terme de débit, de type de marquage… Le routeur d'entrée doit donc observer le trafic et prendre des décisions de régulation, de marquage, de suppression des paquets. Enfin, les routeurs internes sont chargés d'ordonnancer séparément les flots selon leur classe et de mettre en œuvre des mécanismes de suppression en cas de congestion. La figure 15 représente le fonctionnement général de l'architecture à différentiation de services *DiffServ*. Ci dessous sont présentés les trois points essentiels de *DiffServ* : le marquage par l'émetteur, la vérification du profil des paquets par le routeur d'entrée et la différentiation de service dans les routeurs internes.

## Le marquage par l'émetteur

```
0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|        DSCP             |  CU   |
+---+---+---+---+---+---+---+---|
```

**Figure 16 Champs DS**

L'émetteur peut décider de marquer ses paquets IP s'il a souscrit un contrat avec son fournisseur d'accès pour un service donné [Les services proposés par l'architecture à différentiation de services]. Dans ce cas, pour permettre la mise en œuvre de la différentiation de services on définit dans l'en-tête du paquet IP le champ DS [39] [Figure 16]. Ce champ permet de différentier les paquets entre eux par l'action de marquage, c'est-à-dire par le positionnement des bits de ce champ. A l'intérieur du réseau, les routeurs relaient le paquet en fonction de la valeur de ce champ DS.

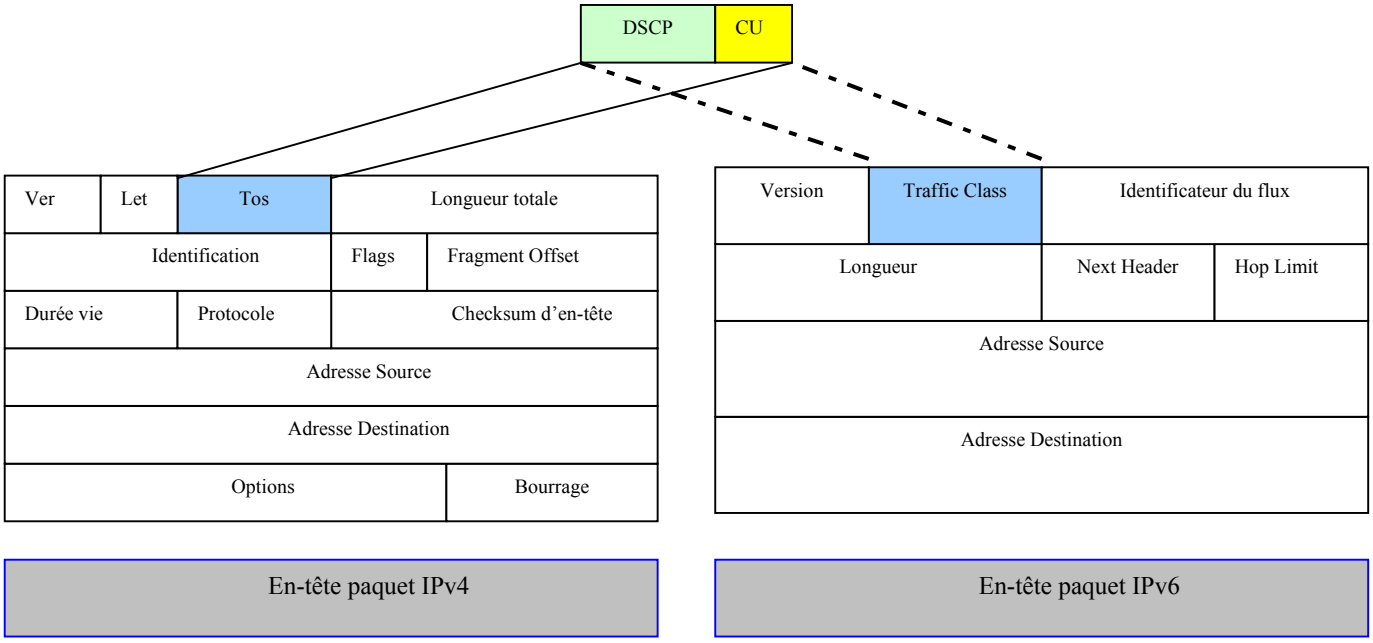



**Figure 17 Champs DS dans l'en-tête des paquets IPv4 et IPv6**

Le champ DS [Figure 16] est défini dans l'architecture *DiffServ* en remplacement de l'ancien octet TOS d'IPV4 et l'octet de *Traffic Class* d'IPV6 [Figure 17]. Les six premiers bits de poids faibles sont utilisés pour ce que l'on appelle le codepoint DS (DSCP). Les deux derniers bits de poids forts ne sont pas actuellement utilisés *Currently Unused*. Ceci dit, on doit conserver un minimum de compatibilité ascendante avec l'utilisation de l'IP TOS. Ceci est réalisé à travers la classe *Class Selector*.

## La vérification du profil utilisateur par le routeur frontière

La figure 18 donne le fonctionnement général d'un routeur d'entrée au domaine DS. Le *classifier* envoie les paquets sur les *Traffic Conditioning Block* [43] du client correspondant. Le TCB vérifie si le trafic est conforme et envoie sur la sortie du routeur.

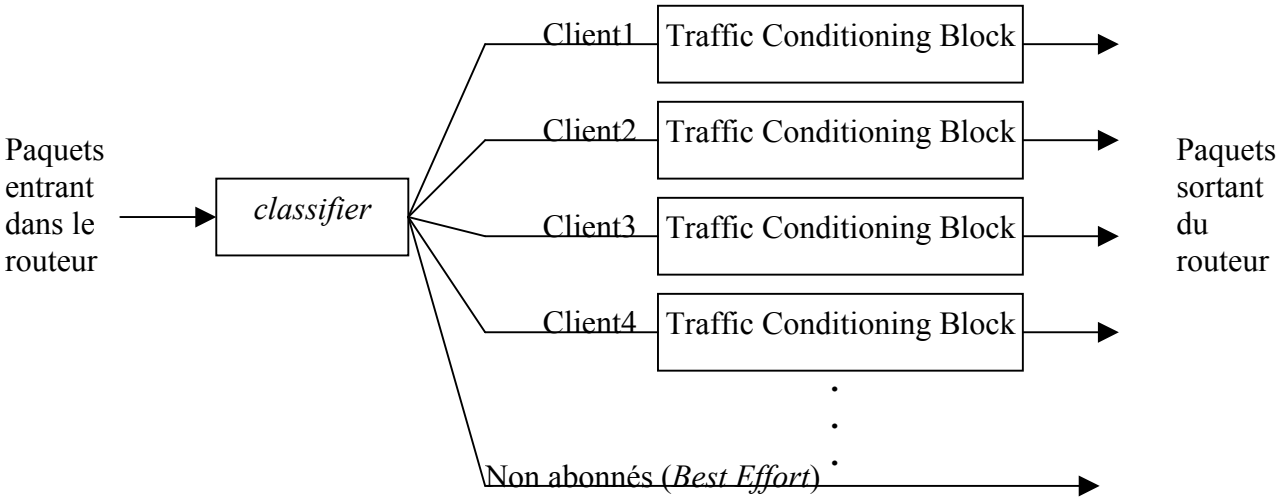



**Figure 18 Fonctionnement d'un routeur frontière au domaine DS**

Ainsi, la première chose que doit vérifier le routeur est la provenance du paquet pour identifier l'émetteur c'est-à-dire le client. Cette identification est effectuée par un *classifier*. Le rôle du *classifier* est donc de classer le trafic. Il en existe plusieurs sortes mais chacun utilise en fait une information de l'en-tête du paquet. La classification est donc effectuée en fonction de l'adresse IP ou bien MAC, en fonction du codepoint DS etc. Le principe du *classifier* est de dispatcher les paquets en entrée sur différentes sorties en fonction des filtres de classification.

La figure 19 montre le schéma d'un *Behaviour Agregate classifier*, celui ci traite les paquets entrant en fonction du champ DS de l'en-tête IP. Les paquets de codepoint DS de même valeur que la valeur du filtre sont dirigés vers la sortie A. Les autres paquets sont dirigés vers la sortie B.

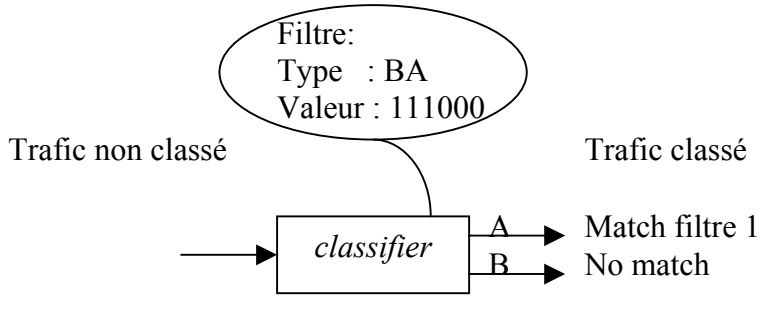



**Figure 19 Classifier BA**

Une fois que le paquet entré dans le routeur a été classifié par le *classifier*, c'est-à-dire une fois que l'on connaît la provenance du paquet, on vérifie la conformité de son trafic. Cette conformité s'exprime en type de marquage, mais aussi en débit. Le profil du trafic est vérifié grâce à un *Traffic Conditioning Block* [Figure 18]. Une fois l'opération de vérification de trafic et dans le cas échéant de mise en conformité terminée, on envoie les paquets sur la sortie du routeur.

Le TCB est propre à chaque client et défini le profil du client. Finalement, le contrat passé entre le client et le fournisseur d'accès est implémenté à l'intérieur du TCB. Un TCB peut contenir un *classifier*, un *marker*, un *shaper*, un *meter* et un *droper*. La figure 20 montre ce que peut donner un TCB. Ceci dit, un TCB est propre à un utilisateur. Dans notre cas, les paquets marqués EF respectant le débit sont envoyés sur la sortie du routeur, ceux ne respectant pas le débit EF sont supprimés. Pour les paquets marqués AF, les paquets hors profil sont envoyés sur un marqueur qui les re-marque en paquets *Best Effort*. Enfin, les paquets *Best Effort* ne reçoivent aucun traitement.

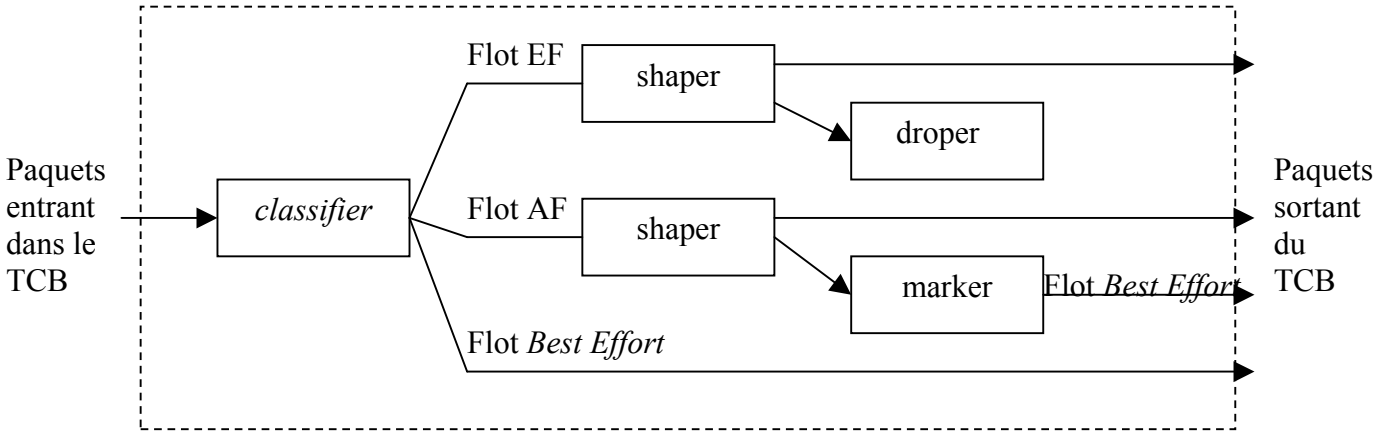



**Figure 20 Exemple de Traffic Conditioning Block**

Le *marker* [6] positionne le champ DS à un codepoint particulier. Le *marker* peut agir sur des paquets non marqués aussi bien que sur des paquets précédemment marqués. Quand le paquet voit son codepoint changé, le paquet est dit re-marqué. Le positionnement du codepoint par le marker déterminera ensuite les traitements que le paquet subira dans les routeurs internes du réseau.

Le *shaper* [6] est utilisé pour donner au trafic un débit donné. Ce débit est bien entendu spécifié entre le client et le fournisseur d'accès. De plus, il est utilisé pour lisser le trafic. C'est-à-dire que le débit instantané ne doit pas varier brusquement. Pour réaliser cette deuxième condition, on utilise couramment le *token bucket* [8.7 Le Tocken Bucket]. Le *token bucket* permet un débit constant et stocke dans une file d'attente les paquets arrivant en rafale.

Le *meter* [6] contrôle les temps d'arrivée des paquets d'un flot et détermine le niveau de validité de chaque paquet par rapport à un profil préétabli. Les fournisseurs d'accès à différentiation de service peuvent choisir d'offrir des services aux clients basés sur des profils temporels auquel le client a souscrit. Il peut aussi être utilisé pour des fonctionnalités d'administration comme les applications de facturation. Le vérificateur comporte une entrée et plusieurs sorties. A chaque sortie est associé un niveau de validité pour un profil donné.

Le *dropper*[6] a pour rôle d'éliminer les paquets. Un *droper* est un point terminal d'un chemin de données. Il peut être intéressant de relayer le paquet à un moniteur pour l'analyser avant de le détruire. On peut par exemple analyser la quantité de paquets éliminés et informer le client de cette quantité pour qu'il ajuste son débit ou son contrat si nécessaire.

### La différentiation de service dans les routeurs internes

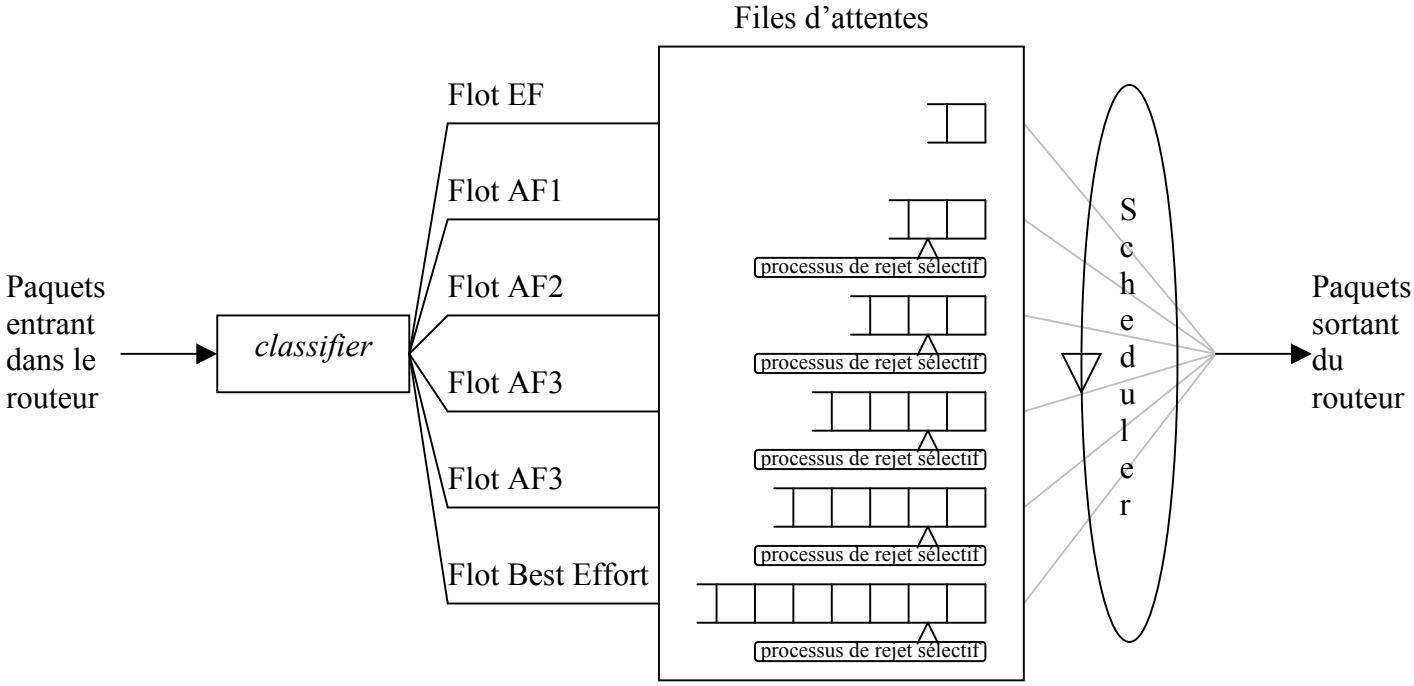



**Figure 21 Fonctionnement d'un routeur interne du domaine DS**

La figure 21 montre le fonctionnement général d'un routeur interne. Les paquets entrants sont envoyés dans l'une des files d'attentes associées à leur classe. Le *classifier* regarde donc le champ DS des paquets pour effectuer sa classification.

Ensuite, le *scheduler* dessert l'ensemble des files d'attentes, c'est-à-dire qu'il extrait les paquets de chacune d'elle pour les envoyer sur la sortie du routeur. Ici, plusieurs politique d'ordonnancement peuvent être mise en place. Ceci dit, on est obligé de respecter le schéma de fonctionnement des classes EF, AF et *Class Selector*.

Ainsi, le *scheduler* dessert en priorité la file réservée à EF. Le débit de retrait dans cette file doit être configuré pour ne pas dépasser 10% du débit du lien de sortie. De plus, la file EF doit être très petite (de l'ordre de quelques paquets) pour éviter à tout flux EF d'expérimenter des temps d'attentes dans les routeurs.

Les autres files d'attentes doivent être desservies par le *scheduler* avec des politiques tel que le *Weighted Fair Queuing* qui donne un débit minimal à la file d'attente, ou bien le *Waiting-Time Priority* [44] qui impose un délais croissant en temps d'attente quand on parcours les classes en partant de la classe AF1 pour se diriger vers la classe *Class Selector*.

De plus un mécanisme de rejet sélectif est mis en place sur chaque file d'attente. Celui-ci est basé sur l'algorithme de RED, *Random Early Detection*. Ainsi, pour les files d'attentes de AF, 3 RED fonctionnent en même temps, on appel cet algorithme RIO. Pour la file d'attente de la classe *Class Selector*, un seul RED est nécessaire. Les algorithmes RED et RIO sont présentés ci dessous.

## *5.4 L'intérêt de la classe AF pour les flots multimédia*

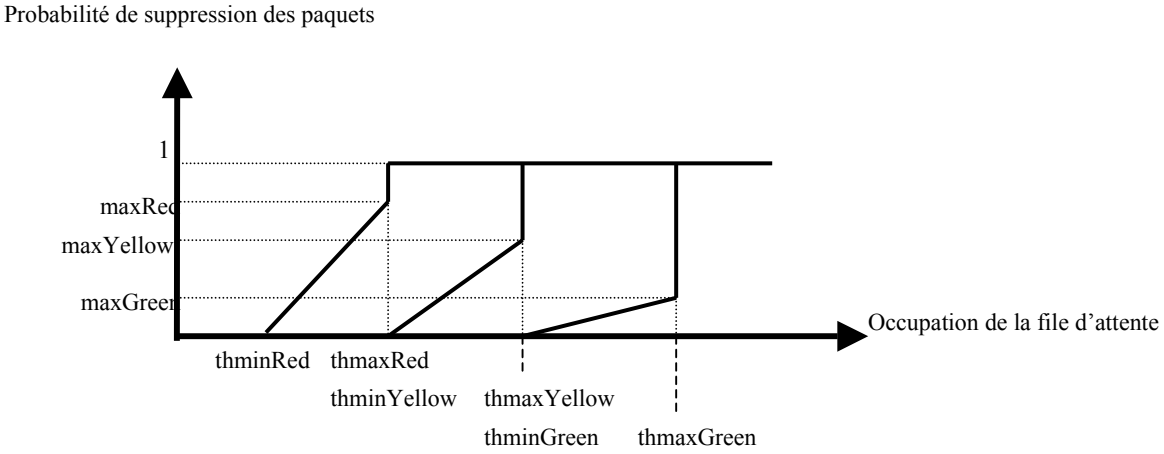



**Figure 22 Algorithme RIO**

Le mécanisme qui permet à AF de fonctionner se trouve à l'intérieur des routeurs internes du domaine DS. C'est le processus de rejet sélectif RIO qui permet d'introduire les couleurs rouge, jaune et vert. L'algorithme de RIO est tout simplement une généralisation de RED [L'Algorithme RED] puisque l'on utilise plusieurs RED pour former RIO. Sur la figure 22, est présenté le mécanisme de rejet des paquets selon l'algorithme RIO pour la classe AF. On constate que les paquets marqués Rouge seront les premiers supprimés en cas de congestion puis viendra les paquets jaunes et ensuite les paquets verts.

La classe AF, comme on l'a dit précédemment, est bien adaptée au flux multimédia à contraintes temporelles moyenne, et pouvant tolérer quelques pertes. Tout d'abord car le service rendu est de meilleur qualité que le service *Default Forwarding*. Cette qualité s'exprime en terme de délais de transmission mais aussi en terme de taux de pertes. Ensuite, car les trois niveaux de probabilité de suppression Rouge, Jaune et Vert peuvent être habilement utilisés pour amoindrir l'effet des pertes de paquets. En effet, comme on l'a vu avec le codage MPEG certains paquets peuvent contenir une information plus vitale que d'autre [Création de différentes couches]. Ainsi, la protection des informations les plus importantes est réalisable via la classe AF.

L'article [28] conforte d'ailleurs la validité de ce choix. En effet, l'utilisation classique d'UDP avec un flot MPEG, lors d'un exemple de congestion sur New York, pour 20% de perte de paquets entraîne 90% de perte de *frame*. Ceci est dût à l'entrelacement des images du flux vidéo. On constate donc que la protection de certaines informations du flux MPEG est nécessaire sans quoi on risque une forte perte de qualité lors des phases de congestions ! Cette remarque est la base de la réflexion de ce rapport. Dans la partie suivante, on manipule le flux transporté pour s'adapter à la classe AF et ainsi amoindrir l'effet de pertes. On propose en résumé un service de transport de flux MPEG Audio adapté à la classe AF.

# 6 Réalisation

Au vu de tous les principes exposés dans les chapitres précèdent, on va s'attacher maintenant à la réalisation d'une application audio à la demande pour une architecture à différentiation de service.

Comme on l'a montré précédemment, la classe AF est bien adaptée aux flots hiérarchiques car elle protège plus certaines informations que d'autres. De plus, les applications adaptatives, qui sont les applications de demain, nécessitent aussi une hiérarchisation de l'information. Ainsi, dans cette partie, nous présentons une technique pour permettre le transport de flux MPEG Audio sur différentes couches.

Dans un premier temps, nous présenterons la technique de hiérarchisation du flux. Ensuite, nous aborderons le mécanisme de fonctionnement du serveur et du client. Puis, nous présenterons brièvement les outils nécessaires à l'implémentation.

La hiérarchisation du flux MPEG Audio passe par un traitement des *frame* du *bitstream*. En effet, on veux pouvoir créer trois couches qui seront chacune envoyées au client. Une des couches sera la couche la plus importante (couche de base) et sera donc marquée en vert. Ensuite les couches d'amélioration seront marquées en jaune et en rouge. L'ensemble des couches doit permettre de reconstituer le *bitstream* original. Bien entendu, si une couche d'amélioration est manquante, le *bitstream* peut être reconstitué mais il est alors de moindre qualité. L'ensemble des couches est transporté via une session RTP/RTCP. L'utilisation de ce protocole permettra, plus tard, de mettre en œuvre les mécanismes d'adaptation du débit ainsi que les mécanismes de FEC. Dans tout les cas, l'utilisation de ce protocole permet de tester de manière plus réaliste le serveur que nous mettons en œuvre.

## *6.1 Hiérarchisation du flux MPEG Audio.*

Comme on l'a vu sur la figure 3, la frame de la *Layer I* a 12 échantillons par bande et celles des *Layer II* et *III* ont 3x12 échantillons par bande. Sachant, qu'une *frame* contient 32 bandes de fréquences. L'idée est de créer à partir d'une *frame* trois sous-*frame* qui lorsqu'elles sont cumulées permettent de réobtenir la *frame* initiale.

Pour ce faire, une solution serait de retrouver les échantillons PCM de la *frame* et de ré-échantillonner à une fréquence plus faible. Par exemple, une *frame* échantillonné à 44100 Hz pourrait être sous échantillonnée à 22050 Hz. Au final cela nous permet d'obtenir 2 *frame* échantillonnées à 22050Hz qui lorsqu'elles sont cumulées permettent de réobtenir la *frame* initiale.

Cependant, la solution précédente nécessite une décompression puis une re-compression avant émission et après réception. L'idée est donc d'imiter la première solution mais sur la *frame* codé en MPEG. La *frame* associé à la couche de base comportera 6 x 32 échantillons, les deux autres *frame* comporteront chacune 3 x 32 échantillons. Ainsi, la *frame* de la couche de base échantillonne la *frame* initial avec un rapport de moitié. Les *frame* des couches d'amélioration échantillonne au quart la frame initiale.

Le choix des rapports est dicté à la fois par *DiffServ* qui pour la classe AF marque la couche de base avec la couleur verte [Création de différentes couches] et donc doit voir cette couche transporter plus d'information. De plus, la couche de base doit dans le pire des cas (perte des paquets rouges et jaunes ou régulation drastique du débit) suffire à elle-même, il faut donc suffisamment d'information dans cette couche. Ensuite, les couches supplémentaires doivent permettre de diminuer le débit si on supprime leur émission [3.2 l'adaptation de qualité] et elles doivent en même temps apporter suffisamment d'information.

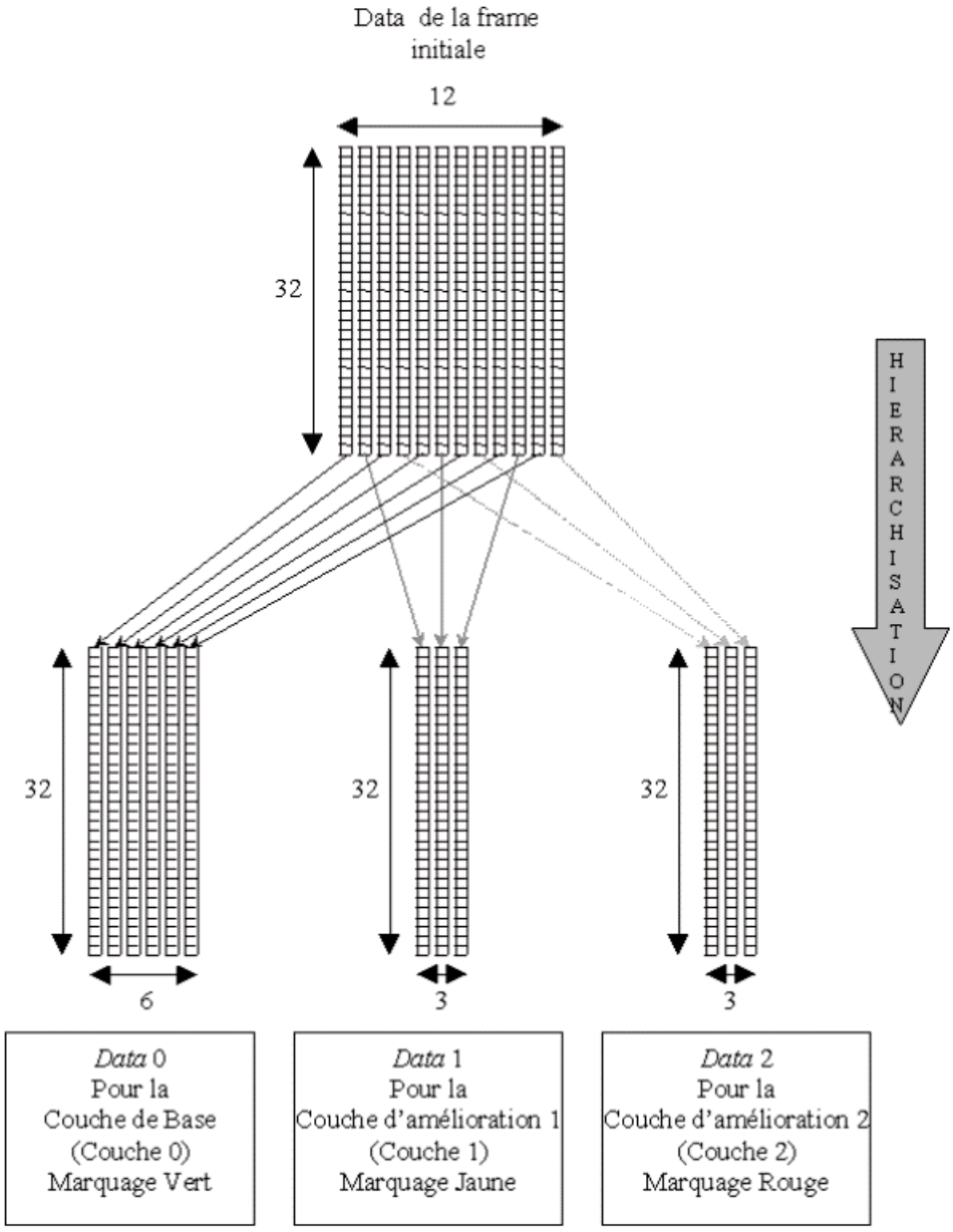



**Figure 23 Hiérarchisation du flux MPEG Audio**

La figure 23 montre comment le découpage des données de la *frame* peut être effectué. Bien sûr, avant émission d'un paquet d'une couche, on regroupe les données de *frame* consécutives de cette couche pour remplir un paquet RTP. Il faut bien voir que cette technique implique que si les paquets des couches d'amélioration sont perdus, ont se retrouve avec une trame reconstruite uniquement avec les informations de la couche de base. Ainsi, puisque l'on à seulement 6x32 échantillons pour une trame initiale, on se retrouve lors de l'écoute avec une oscillation de 32 échantillons normaux puis 32 échantillons vide de sens. Pour donner une idée de l'effet que cela peut donner, 32 échantillons à une fréquence de 44100 Hz représentent 0,726 ms. Ainsi, on à moins de un millième de secondes non valides.

La reconstruction du flux grâce au différentes couches est effectuée en fonction du nombre de couches reçues par le récepteur. Bien entendu, les couches manquantes doivent être reconstruites pour obtenir une *frame* MPEG Audio. La solution la plus simple est de copier à la place des 32 valeurs de fréquences manquantes les 32 valeurs de fréquences que l'on dispose. Cela nécessite bien entendu de disposer de la couche de base qui est la seule à

posséder les en-têtes MPEG Audio et qui possède un groupe de 32 valeurs de fréquences sur deux. L'effet produit par un tel filtre de reconstruction sera probablement de l'écho.

Pour MPEG1 et MPEG2 Audio *Layer I* et *Layer II*, le schéma de hiérarchisation présenté ne nécessite pas plus de travail sur le bitstream mais par contre, pour mp3, il est nécessaire de construire l'ADU avant d'effectuer la hiérarchisation. En effet, les *frames* audio Layer III telle qu'elles existent ont des dépendances entre elles. Ceci est dût à la technique du réservoir de bits [Figure 6]. Ainsi, il faut réorganiser les donnés avant hiérarchisation. Le *draft* [45] explique comment créer les ADU, il est d'ailleurs actuellement un des sujets de débat de la *mailing list avt*.

Le *parsage* du fichier MPEG Audio donne donc lieu à une analyse *frame* par *frame* [8.2 Formule de calcul de la taille d'une frame audio MPEG1 ou MPEG2]. Un exemple de codage du *parser* est donné en annexe. Lorsque l'on analyse une trame, on génère le codage hiérarchique décrit au-dessus. Trois ADU de la *frame* analysée sont donc obtenus. Chaque ADU est alors placé dans la file d'attente propre à sa couche. Dans le paragraphe suivant nous décrivons le processus de lecture qui correspond à l'émission du flux MPEG Audio vers un client.

### *6.2 La commande de lecture*

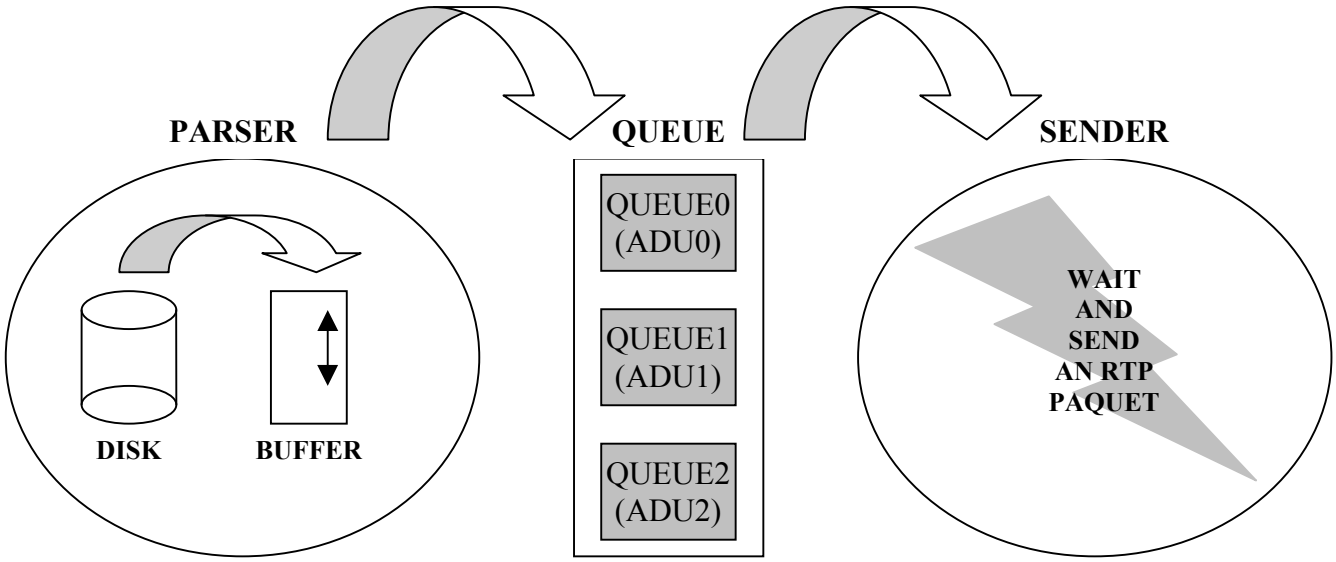



**Figure 24 Fonctionnement de l'émission du flux marqué AF**

Le serveur après avoir reçu la commande de lecture de la part du client lance l'exécution de deux *thread* [Figure 24]. Le premier *thread* est le *parser*. Il a pour but d'extraire les *Applications Data Unit* du *bitstream*. Pour cela, une recopie du fichier en mémoire est effectuée puis un *parse* du flux. Une fois que l'ADU de chaque couche est déterminé, l'objet est placé dans la file d'attente de sa couche (*Queue*).

Le second *thread* est le *sender*. Il retire des files d'attentes les ADU et les envois au débit propre à chaque couche. Il faut remarquer que l'attente avant émission peut être envisagée de plusieurs façons. Selon que l'on est en avance normale ou lente, l'attente n'est pas la même.

Le mécanisme de synchronisation est classique. Il s'agit d'un producteur consommateur régulé par sémaphore. La zone d'échange étant bien entendue la file d'attente. Le mécanisme *multithread* a montré qu'il était plus performant [36] qu'un mécanisme *monothread*.

## Le Parser

Le principe du parser est simple, il trouve une *frame* dans le *bitstream*, crée un ADU hiérarchique, et le place dans la bonne file d'attente. L'intérêt de la synchronisation par sémaphore est que le *parser* se voit stoppé quand il n'y a plus de place dans les files d'attentes. De plus, le *parser* peut commencer à produire quelque paquet avant que le Sender ne puisse en consommer. Pour ceci, il suffit de choisir judicieusement la valeur initiale du sémaphore. Elle doit permettre de produire plus de *MTU/tailleADU* objets ADU avant que le Sender ne consomme. Bien entendu, il faut absolument que le *parser* ait un débit de production supérieur à celui du Sender. Autre constatation : la taille de l'ADU ainsi que son numéro de séquence sont des informations importantes qu'il faut conserver dans l'objet ADU.

## Le Sender

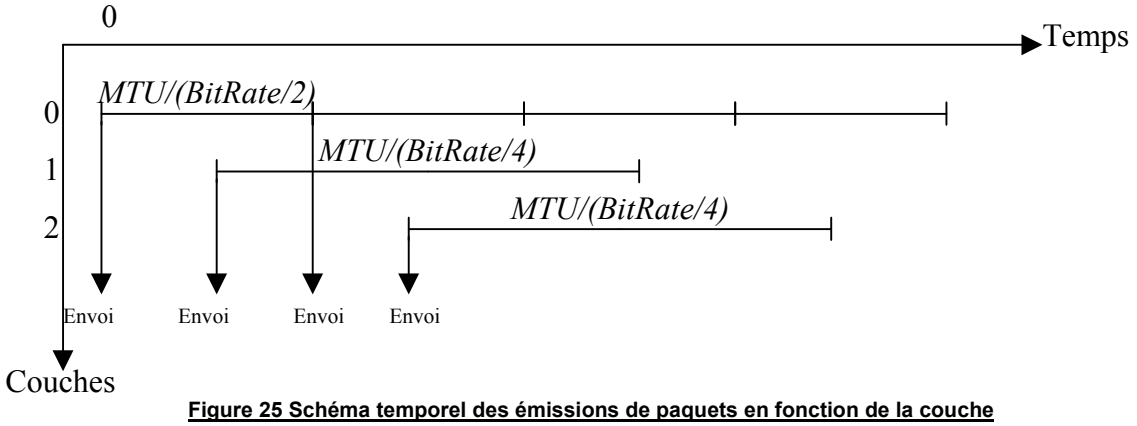



**Figure 25 Schéma temporel des émissions de paquets en fonction de la couche**

Pour ce qui est du débit d'émission de chaque couche, il faut faire en sorte que le débit utile en sortie du serveur soit le même que le débit réel du flux audio. C'est à dire que la vitesse d'émission doit être la même que la vitesse de jeu ou d'écoute du client. Ainsi, le *BitRate* du flux audio doit être respecté en émission. La couche de base avec la hiérarchisation du paragraphe 6.1 doit avoir un débit utile légèrement supérieure à *BitRate/2*. Les deux autres couches ont un débit légèrement inférieur à *BitRate/4*. Pour utiliser un décalage à l'émission des paquets (*shift parameter*) on envoie donc un paquet toutes les *MTU/BitRate* secondes, si l'on considère que la taille du paquet émis est de MTU octets. Ainsi, on envoie d'abord un paquet de la couche 0 puis un paquet de la couche 1 ensuite un paquet de la couche 0 et enfin un paquet de la couche 2. Puis on recommence le cycle avec un intervalle de *MTU/BitRate*. La figure 25 montre le schéma temporel des émissions.

On peut remarquer qu'il est alors facile d'intégrer le mécanisme de régulation de débit puisqu'il suffit, si une couche ne doit pas être envoyée, de supprimer les ADU qui auraient dues être envoyés et bien sûr, ne rien envoyer du tout.

## Format d'échange des données entre le serveur et le client

Comme on l'a vu, le *Sender* envoie des paquets contenant un ou plusieurs ADU consécutifs de la même couche. Il faut donc un format d'échange des données commun entre le Serveur et le Client. Ce format doit pouvoir permettre au client de reconstruire les ADU lorsqu'il reçoit le paquet. Ainsi, le contenu du paquet RTP, *payload*, sera d'abord composé de

deux octets indiquant le numéro de séquence du premier ADU. Ensuite, un octet indique le nombre d'ADU contenu dans le paquet RTP. Puis, s'il y a plus d'un ADU dans le paquet, on a *nbADU – 1* pointeurs indiquant l'endroit de fin des *nbADU – 1* premier ADU. Ces pointeurs occupent un espace de 2 octets. Ensuite, les données ADU sont misent les unes a la suite des autres.

Le choix du *Sender* quant au nombre d'ADU contenus dans son paquet est dicté par la taille globale du paquet. En effet, la taille du paquet IP doit être inférieure au MTU. Ainsi, le nombre d'ADU *nbADU* doit être choisi de telle sorte que :

$$43 + (nbADU - 1) \times 2 + TailleChaqueADU \leq MTU$$

Cette formule, se comprend facilement en regardant la figure 26

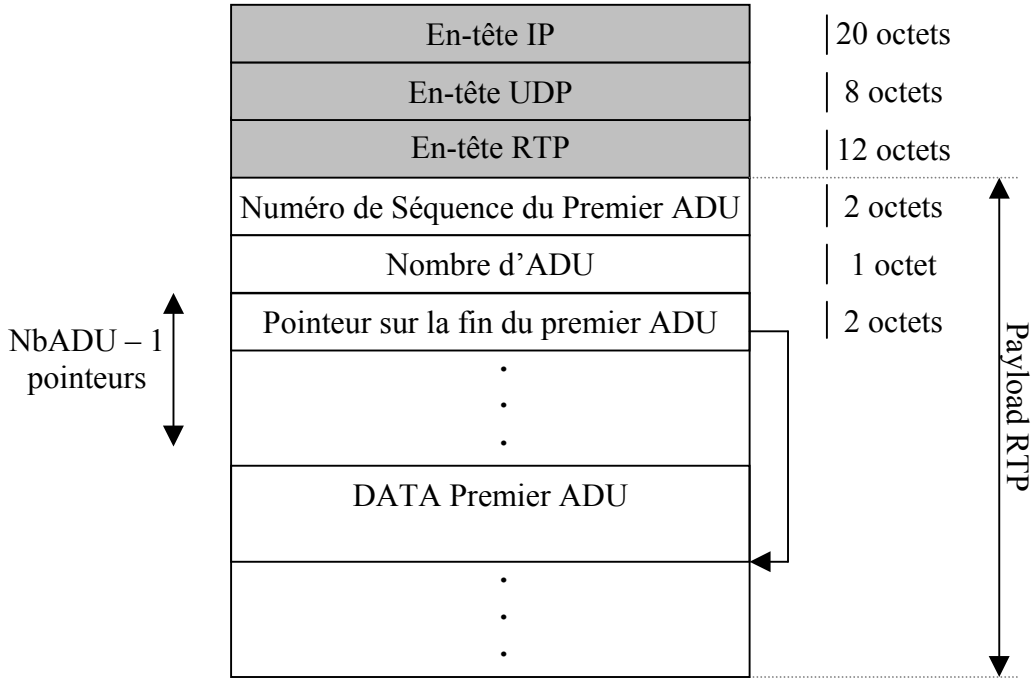



**Figure 26 Format d'échange de données**

Ainsi, on envoi différentes couches au client via RTP et en utilisant 3 canaux de données [Création de différentes couches] avec un marquage vert pour la couche de base 0, jaune pour la couche d'amélioration 1, et rouge pour la couche d'amélioration 2. Le paragraphe suivant indique comment le récepteur récupère les données envoyées par le serveur.

## *6.3 Récupération des données par le client*

De la même manière que le pour la commande lecture, le client en réception lance deux *thread* [Figure 27]. Le premier est le *Receiver* qui consulte régulièrement les trois ports de la session. Il extrait les paquets quand ils arrivent et recrée les ADU pour les placer dans la file d'attente *Queue*.

La file d'attente de chaque couche est une file ordonnée. C'est-à-dire que l'ADU ayant le plus petit numéro de séquence se trouve en tête de file. Lorsque le *Distributor* retire un

ADU de la file d'attente, il prend le premier de la liste. Le principe des files d'attentes ordonnancé, a trois avantages. Le premier avantage est que cette manière de faire permet au *Distributor* de savoir si des paquets de certaines couches sont manquantes. En effet, lorsque le *Distributor* retire des ADU, normalement, il retire un ADU par file d'attente (couches). Si dans une des files d'attentes, l'ADU correspondant au numéro de séquence courant n'est pas présent, alors cette ADU est considérée comme perdue. Le deuxième avantage est que le mécanisme de bufferisation permet d'annuler l'effet de gigue. Enfin, le dernier avantage est que l'on peut réordonner les ADU. En effet, il se peut que lors de la traversé des paquets dans le réseau, il y ait des inversions d'ordre.

Le deuxième *thread* est le *Distributor*. Son rôle est d'alimenter le *player* du client au débit du *bitstream*. Pour ce faire, il extrait de la file d'attente de chaque couche les ADU et reconstruit les *frame* initiales. Chaque *frame* est donnée au *player* à intervalles réguliers.

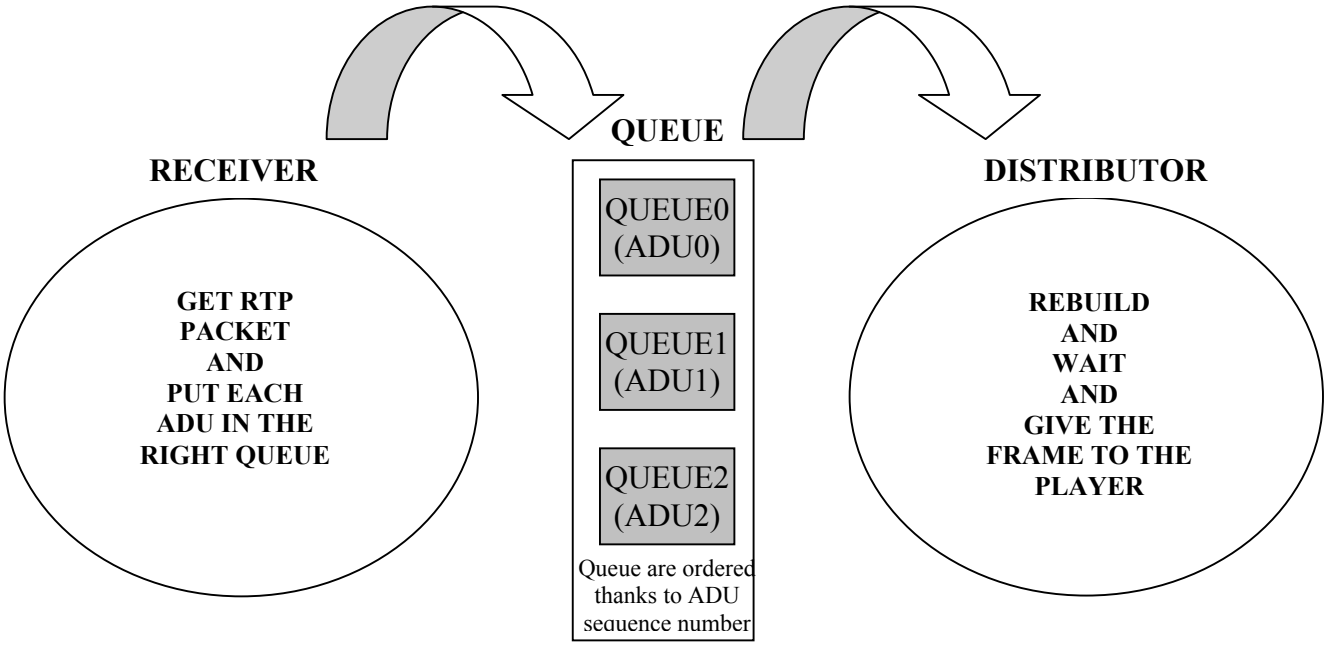



**Figure 27 Fonctionnement à la réception du flux**

## *6.4 outils nécessaires à l'implémentation*

Dans cette partie, nous présentons brièvement, la librairie RTP et les ajouts que celle ci doit subir, et le *player* MPEG Audio.

La librairie RTP/RTCP que nous avons retenu est la librairie de Jori Liesenborgs. Celle ci en est à la version 2.3 et date du 20 avril 2000. On la trouve sur le site *www.cs.columbia.edu/~jdrosen/rtp-api.html* et elle suit les spécifications du RFC 1889. Jori Liesenborg est un étudiant belge actuellement en thèse et travaille sur RTP/RTCP. Plusieurs personnes utilisent et font évoluer cette librairie, on peut citer l'équipe TEMICS de l'IRISA. La notoriété qu'on apporte à cette librairie est du à sa conformité, sa clarté et sa portabilité. La première fonctionnalité à ajouter est la possibilité de marquer le flux via des méthodes de l'interface de la librairie. La seconde fonctionnalité à ajouter est la possibilité d'envoyer des paquets IPv4 ou bien IPv6.

Le logiciel de jeu de MPEG Audio que nous avons retenu est mpg123. On l'a retenu car il autorise la lecture de toutes les couches MPEG1 et MPEG2 Audio. De plus, il est portable et est reconnu pour ces bonnes performances. On le trouve sur le site *http://www.mpg123.de/*.

## *6.5 Etat d'avancement*

Pour le moment, l'implémentation du serveur de musique à la demande n'est pas encore implémenté. Donc aucun test n'est pour l'instant réalisé. Je travaille actuellement en co-tutelle avec un étudiant pour la hiérarchisation du flux MPEG. J'ai déjà implémenté en C++ du code qui parcours le bitstream MPEG. J'ai aussi fait une première évaluation de la librairie RTP/RTCP avec un petit client serveur. Bref, l'implémentation devrait être terminé d'ici fin juillet.

Dans tout les cas, il reste à inclure le mécanisme de marquage dans la librairie RTP. Il faut permettre l'utilisation d'Ipv6 dans la librairie (l'architecture à différentiation de service fonctionne sur Ipv6). Enfin, il faut mettre en place un mécanisme de traçage des trames perdus, (*monitoring*) pour les tests.

Dernière chose, l'introduction des mécanismes de FEC, de mécanisme de régulation de débit (application adaptatives) et l'ajout du protocole RTSP[5] peuvent être envisagés dans le futur.

[5] RTSP : Real Time Streaming Protocol.

# 7 Conclusion

Le multimédia est actuellement en plein essor et le transport de flots multimédia à travers des réseaux tel Internet est un sujet vaste qui aborde deux grands domaines que sont l'image et les réseaux. Deux idées essentielles émanent.

Tout d'abord, il est nécessaire de scinder le flux de manière à respecter l'ADU, *Application Data Unit*. Cette ADU doit être taillée pour respecter les contraintes de taille des paquets IP et de la MTU. Si celle-ci n'existe pas, alors il faut la créer. Tout ceci bien entendu pour restreindre l'influence de la perte de données sur le réseau.

Une autre idée émergente associée à la différentiation de service ainsi qu'aux applications adaptatives, est la nécessité de niveler le flux multimédia. Avec l'introduction de niveaux d'importance, les flux multimédias vont pouvoir prendre leur envol. Pour l'instant, on s'oriente grandement vers des marquages de paquets AF et EF. Pour AF, il est donc nécessaire de fournir un flux nivelé pour avoir un marquage qui protège les paquets transportant le plus d'information.

Ainsi, plus particulièrement pour le flux MPEG1 et MPEG2 Audio, l'ADU est la *frame* pour les couches I et II. Pour ce qui est de la couche III, l'ADU doit être construite. Cette ADU nécessite d'être hiérarchiser pour pouvoir introduire la notion de couches et ensuite profiter pleinement de la classe AF de *DiffServ*. Le rapport présente donc une technique de hiérarchisation de l'ADU. Les trois couches créées vont être marquées en vert pour la couche de base 0, en jaune pour la couche d'amélioration 1, et en rouge pour la couche d'amélioration 2. Les trois couches sont alors émises au client sur trois canaux RTP différents. Ce mécanisme en place, il est très facile d'y ajouter un fonctionnement de type application adaptative.

Maintenant que ces constatations ont été faites, plusieurs grands axes de recherche sont possibles. Le premier s'oriente plutôt vers une étude de transports de nouveaux flux tel que MPEG4 ou bien la voix. Comment *parser* le flux ? Comment marquer le flux ? Beaucoup de personnes travaillent actuellement sur ce genre de problème via l'écriture de *draft* RTP. On peut citer l'équipe TEMICS de l'IRISA.

Le deuxième axe concerne plus l'aspect réseau et la différentiation de service. Beaucoup reste à faire pour trouver des solutions à l'implémentation de services fonctionnels. Cette partie reste plus délicate dans le sens ou l'expérimentation à grande échelle est plus difficile à mener. Ceci dit, beaucoup de questions restent ouvertes. L'équipe ARMOR de l'IRISA travaille sur cette partie.

# 8 Annexes

## *8.1 Modèle psychoacoustique*

Le modèle psychoacoustique est basé sur le système d'audition humaine. Les trois points remarquables sont l'intervalle de fréquence audible, le phénomène de masquage de fréquence proche et le masquage temporel.

Pour ce qui est de l'intervalle audible, il est situé principalement entre 2 et 5 KHz. [Figure 28]. En dehors de ces fréquences le son doit être plus élevé pour être perçu.

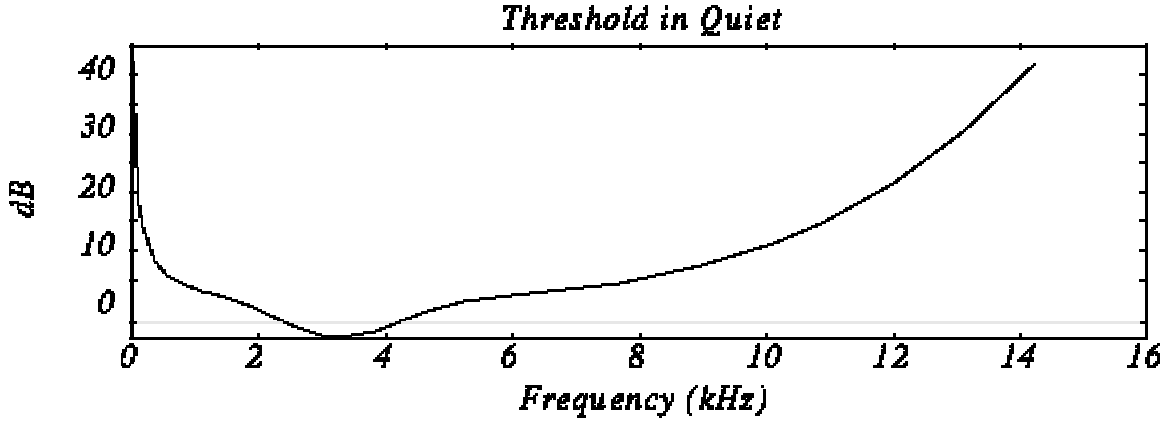



**Figure 28 Caractérisation des fréquences audible**

Le phénomène de masquage est des plus simples, un son de forte intensité rend imperceptible les sons d'intensité faible, qui ont une fréquence dans la zone de masquage. La figure 29 montre le phénomène.

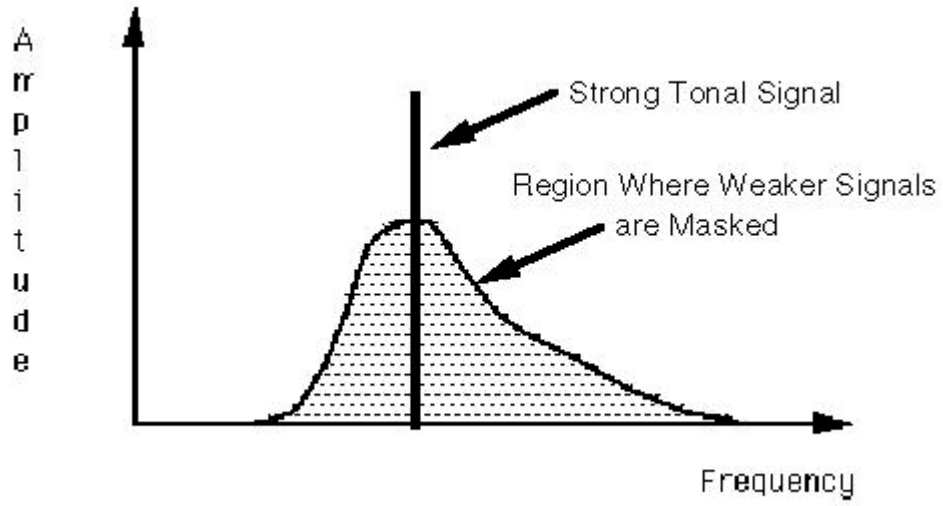



**Figure 29 phénomène de masquage**

Le phénomène de masquage temporel est tel qu'un son de forte intensité rend imperceptible les sons d'intensités faibles, qui sont joués dans la zone de masquage temporel. La figure 30 montre le phénomène.

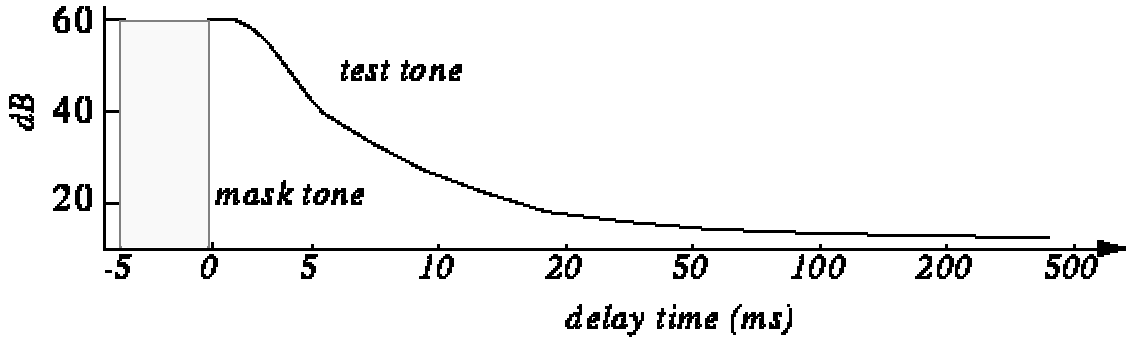



Figure 30 Masquage temporel

Finalement, n'importe quel son définit une enveloppe de masquage fréquentielle et temporelle qui peut être largement utilisée pour la compression de données. La couche I exploite uniquement l'intervalle audible et le masquage de fréquence. Les couches II et III quant à elle utilisent en plus le masquage temporel.

### *8.2 Formule de calcul de la taille d'une frame audio MPEG1 ou MPEG2*

Pour ce qui concerne la couche I, la taille de la *frame* en octets est :

*FrameLengthInBytes* = (12 x *BitRate* / *SampleRate* + *Padding*) x 4

Pour ce qui concerne la couche II et III, la taille de la *frame* en octets est :

*FrameLengthInBytes* = 144 x *BitRate* / *SampleRate* + *Padding*

Le *BitRate*, le *SampleRate* et le *Padding* sont déterminés grâce à l'en-tête de *frame*. Le *padding* étant un booléen indiquant s'il y a des bits de bourrage. Le bourrage s'effectue par ajout de *slot*. Un *slot* est de 4 octets pour la *Layer I* et 1 octet pour la *Layer II* et *III*.

### *8.3 Exemple de compression spatiale d'un bloc*

La compression spatiale prend un bloc 8x8 [Figure 33] et lui applique la transformation DCT [figure 31] *Discrete Cosinus Transformation* [Figure 34]. Cette opération transforme l'échelle de représentation qui est la luminosité Y ou bien la chrominance U et V, en fonction de la position dans le bloc, en une échelle de fréquence d'apparition d'un pixel dans un bloc. On concentre de cette façon l'énergie du bloc au voisinage du point de coordonnée (0,0).

$$F(u,v) = \frac{C_u}{2}\frac{C_v}{2}\sum_{y=0}^{7}\sum_{x=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

with:

$$C_u = \begin{cases} \frac{1}{\sqrt{2}} & \text{if u} = 0, \\ 1 & \text{if u} > 0 \end{cases}; C_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{if v} = 0, \\ 1 & \text{if v} > 0 \end{cases}$$

Figure 31 Formule de la DCT

$$f(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7} F(u,v)\,\frac{C_u}{2}\,\frac{C_v}{2}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

**Figure 32 Formule de la DCT inverse**

Une fois que la matrice DCT est obtenue [Figure 34], il faut quantifier celle-ci [Figure 35]. On restreint de cette manière le domaine de valeurs contenues dans la matrice. La valeur d'un élément quantifié est égale à l'arrondi obtenu par division de la valeur par le coefficient de quantification. Il est évident qu'il y a perte d'information puisque la déquantification (opération inverse) ne permet pas d'obtenir la valeur initiale. Ceci dit, la distorsion introduite par la quantification n'est pas visible a l'œil.

Une fois que la matrice est quantifiée [Figure 36], on adresse celle ci en zigzag [Figure 37]. Cela signifie que l'on parcourt celle-ci en partant du coin supérieur gauche (zone ou l'énergie est concentrée) pour se diriger vers le coin inférieur droit. De cette manière, on ne représente pas les zéros terminaux puisque l'on utilise un marqueur de fin de séquence.

Enfin, pour finir, on utilise un codage de type huffman pour réduire l'occupation du total du flux en minimisant l'occupation bits des valeurs les plus présentes [Figure 39].

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 140 | 144 | 147 | 140 | 140 | 155 | 179 | 175 |
| 144 | 152 | 140 | 147 | 140 | 148 | 169 | 179 |
| 152 | 155 | 136 | 167 | 163 | 162 | 152 | 172 |
| 168 | 145 | 156 | 160 | 152 | 155 | 138 | 160 |
| 162 | 148 | 156 | 148 | 140 | 136 | 147 | 162 |
| 147 | 167 | 140 | 155 | 155 | 140 | 136 | 162 |
| 136 | 156 | 123 | 167 | 162 | 144 | 140 | 147 |
| 148 | 155 | 136 | 155 | 152 | 152 | 147 | 136 |

**Figure 33 Matrice Bloc**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1210 | -18 | 15 | -9 | 23 | -9 | -14 | -19 |
| 21 | -34 | 26 | -9 | -11 | 11 | 14 | 7 |
| -10 | -24 | -2 | 6 | -18 | 3 | -20 | -1 |
| -8 | -5 | 14 | -15 | -8 | -3 | -3 | 8 |
| -3 | 10 | 8 | 1 | -11 | 18 | 18 | 15 |
| 4 | -2 | -18 | 8 | 8 | -4 | 1 | -7 |
| 9 | 1 | -3 | 4 | -1 | -7 | -1 | -2 |
| 0 | -8 | -2 | 2 | 1 | 4 | -6 | 0 |

**Figure 34 Matrice DCT**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 |
| 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 |
| 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 |
| 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 |
| 11 | 13 | 15 | 17 | 19 | 21 | 23 | 25 |
| 13 | 15 | 17 | 19 | 21 | 23 | 25 | 27 |
| 15 | 17 | 19 | 21 | 23 | 25 | 27 | 29 |
| 17 | 19 | 21 | 23 | 25 | 27 | 29 | 31 |

**Figure 35 Matrice de quantification**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 403 | -4 | 2 | -1 | 2 | -1 | -1 | -1 |
| 4 | -5 | 3 | -1 | -1 | 1 | 1 | 0 |
| -1 | -3 | 0 | 0 | -1 | 0 | -1 | 0 |
| -1 | 0 | 1 | -1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | -1 | 1 | 1 | 1 |
| 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 36 Matrice DCT quantifiée**

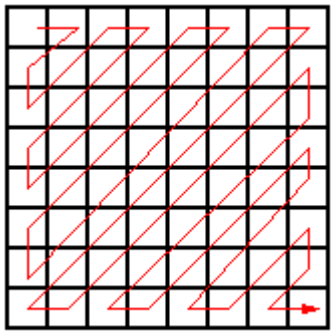


**Figure 37 Schéma d'adressage en zig-zag**

403, -4, 4, -1, -5, 2, -1, 3, -3, -1, 0, 0, 0, -1, 2, -1, -1, 0, 1, 1, 0, 1, 0, 1, -1, -1, 1, -1, -1, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, -1, 0, -1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, FIN.

**Figure 38 adressage en zigzag**

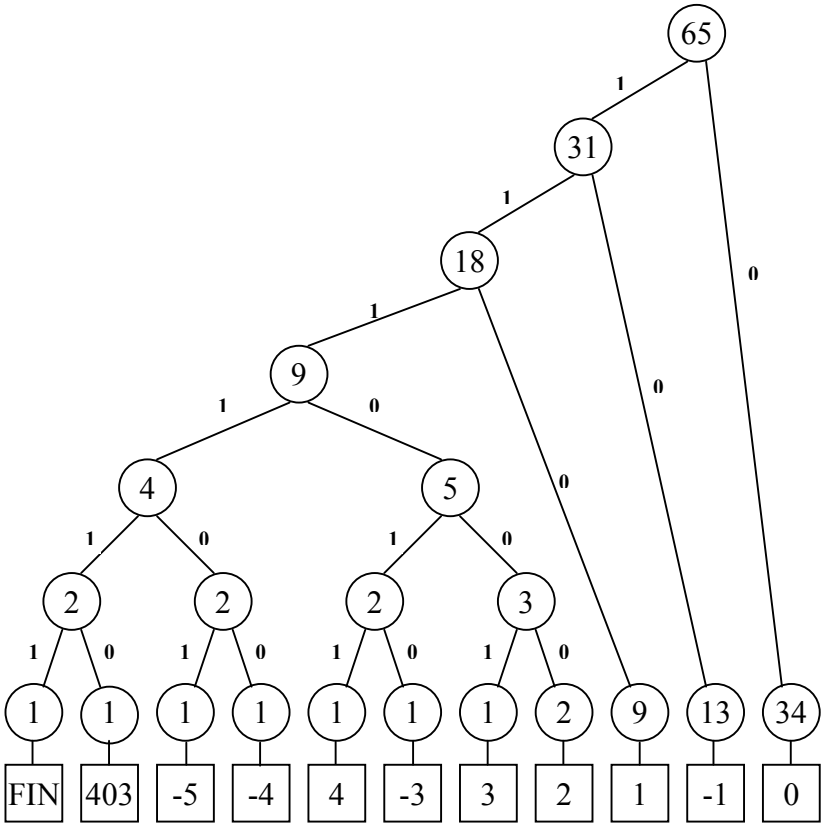



**Figure 39 Arborescence d'Huffman**

### *8.4 Notion d'estampille dans RTP/RTCP*

L'estampille temporelle RTP est une représentation du temps qui indique le début de jeu du premier échantillon d'un paquet contenant un ou plusieurs échantillons de données par rapport au premier échantillon du paquet précédent. Dit plus simplement, elle indique le temps à attendre avant de jouer l'échantillon par rapport au tout premier échantillon. La conversion entre le temps en seconde et le temps en unité estampille est dicté par la fréquence d'incrémentation de l'horloge. Une unité estampille vaut 1/(fréquence d'incrémentation de l'horloge) secondes. Cette fréquence horloge pour l'estampillage est d'ailleurs spécifiée dans les *drafts* pour de nombreux formats multimédia.

De manière pratique, si l'on choisit une fréquence F d'horloge pour l'estampillage. La première estampille A est choisie aléatoirement. Alors, l'estampille suivante si le premier paquet RTP transportait un échantillon « durant » t secondes est de A + t x F. De plus, si l'échantillon à été échantillonné à la même fréquence F que l'horloge pour l'estampillage, alors A + t x F = A + 1. Ainsi, en choisissant une horloge pour l'estampillage identique à la fréquence d'échantillonnage, l'incrément de l'estampille pour un paquet est égal au nombre d'échantillons transportés par le paquet précèdent.

Pour ce qui est du transport de la vidéo, l'échantillon est une image et donc si l'on transporte celle ci en plusieurs paquets, ceux-ci auront la même estampille. D'autre part, l'évolution de l'estampille n'est pas forcément croissante. En effet, dans le cas de vidéos interpolés comme MPEG, l'ordre de transmission n'est pas le même que celui d'échantillonnage (ordre d'échantillonnage égale ordre de jeu). Ceci dit, il faut faire attention

à ces deux derniers points car si on les respecte, certains calculs temporels effectués par RTCP n'ont plus aucun sens (exemple : calcul de la gigue).

L'estampille temporelle RTCP n'a pas la même signification que l'estampille RTP. En effet, ici on indique uniquement le temps NTP d'émission du paquet dans l'unité de mesure de l'horloge d'estampille. De plus, on ajoute la valeur de la première estampille RTP choisie aléatoirement (A).

### *8.5 Calcul du taux de pertes via les informations de RTCP*

Calcul par l'émetteur du taux de pertes du récepteur

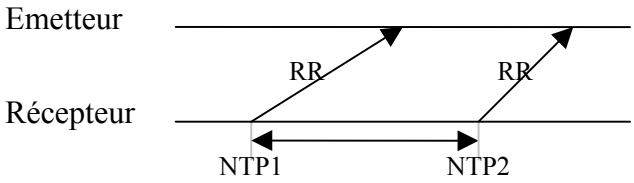



**Figure 40 Deux RR consécutifs**

Le calcul du taux de pertes se base sur l'information fraction de paquet perdue fournie par le récepteur (*feedback* RR). Cette fraction de paquet perdu est une estimation et est mesuré dans l'intervalle de temps séparant deux émissions de RR [figure 40].

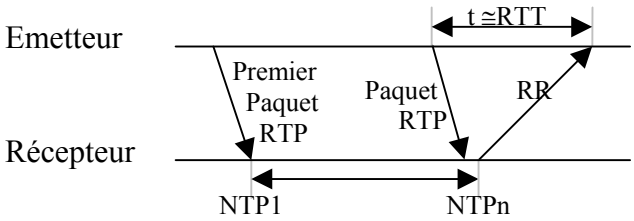



**Figure 41 Calcul du taux de pertes moyen**

Le calcul du taux de perte moyen ce calcul depuis le début de la session. Il suffit de diviser le nombre cumulé de paquets RTP perdus (information contenue dans RR) par le nombre de paquets émis entre NTP1 et NTPn. Le nombre de paquets émis entre NTP1 et NTPn est approximativement égal au nombre de paquets émis depuis le début (au moment de la réception de RR) moins la partie entière supérieure de (RTT x débit en paquets RTP) [Figure 41].

Calcul par le récepteur du taux de pertes du récepteur

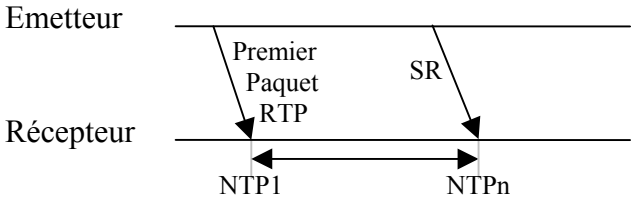



**Figure 42 Calcul du taux de pertes moyen**

Le calcul par le récepteur de son taux de perte est trivial et le calcul du taux de pertes moyen est aussi très aisé puisque c'est la division du nombre de paquets cumulés perdu depuis le début (information locale estimée) par le nombre de paquets émis par l'émetteur (*feedback* SR) [Figure 42].

## *8.6 Calcul du RTT via les informations de RTCP*

### Calcul par l'émetteur

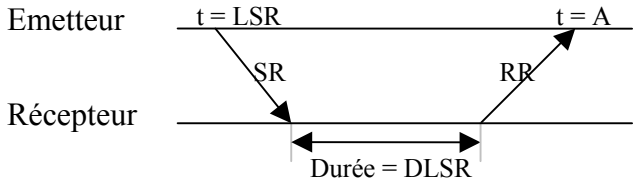



**Figure 43 Calcul du RTT par l'émetteur**

Le calcul par l'émetteur est facile et utilise la formule A – LSR – DLSR [27] [Figure 43] (Cette formule est valable si les paquets SR et RR ne sont pas perdus !). Ceci dit, il se peut que sa fréquence de calcul ne soit pas suffisante. Une solution proposée par l'article [21], lorsqu'il y a perte de paquets (Taux de perte ≤ 16%), est d'estimer le RTT par RTT = *TempsDePropagation* + *DélaisFileAttentes* / 2. Le *TempsDePropagation* étant le plus petit RTT mesuré (on suppose qu'il n'y a pas eu d'attente dans les files d'attente) et *DélaisFileAttentes* est déterminé grâce au RTT précédent et au *TempsDePropagation* (Initialement, on suppose que RTT = *TempsDePropagation* + *DélaisFileAttentes*).

On peut envisager de ce baser sur le calcul de l'article [21] pour avoir une estimation du RTT dans les périodes ou l'on n'a aucune possibilité de l'avoir précisément. Lorsque la mise à jour du RTT est possible de manière précise, on le repositionne. L'algorithme peut être le suivant :

1 Calcul du RTT de manière précise.
2 **FAIRE** Toutes les RTT secondes
  **SI** le calcul du RTT n'est pas possible (pas d'information)
  **ALORS** Estimation du RTT (article [21] )
  **SINON** Calcul du RTT de manière précise.
**FINFAIRE**

### Calcul part le récepteur.

Dans le cas ou le récepteur n'est pas émetteur, on ne peut pas utiliser la formule du paragraphe précédent. Plusieurs solutions peuvent être envisageables. Le seul calcul que l'on peut effectuer est une estimation du temps émetteur récepteur (c'est une estimation car les horloges émetteur récepteur ne sont pas synchronisés). En supposant le lien symétrique, ce qui est faux la plupart du temps, on estime le RTT par 2 x *TempsEmetteurRecepteur*.

*TempsEmetteurRecepteur* = *NTPreceptionSR – NTPemissionSR*
ou bien

*TempsEmetteurRecepteur* = *NTPreceptionRTP* – *NTPemissionRTP*

avec *NTPemissionRTP* = *NTPReçuAuparavant* – *TempsEmetteurRecepteurPrécedent* + ((*EstampilleEmissionRTP* – *EstampilleReçuAuparavant*) / *FrequenceHorlogePourEstampillage*)

Une deuxième solution est d'ajouter une extension au paquets RR et SR qui permettrait de calculer le RTP. Cette solution est d'ailleurs suggérée par le RFC1889 [27] car elle évite la surcharge générée par la définition de nouveaux types de paquets RTCP.

L'article [22] a quant à lui, dans un premier temps, testé la définition de deux nouveaux paquets RTCP (rtt-request et rtt-reply). Ceci dit, quand le nombre de participants augmente, cette méthode ne fournit pas assez souvent de valeur et donc ne permet pas une fréquence de mise à jour suffisante. Il en est de même pour la solution précédente.

Les deux dernières solutions fournissent un RTT mais avec une fréquence de mise à jour qui peut être insuffisante. On peut envisager de ce baser sur le calcul de *TempsEmetteurRecepteur* de la première solution ou bien sur une estimation à vérifier : RTT = t1' + t2 * t1'/t1 (avec t1 le *TempsEmetteurRecepteur* et t2 le *TempsRecepteurEmetteur* lors du calcul précis de RTT et t1' le *TempsEmetteurRecepteur* du dernier paquet RTP) pour avoir une estimation du RTT dans les périodes ou l'on n'a aucune possibilité de l'avoir précisément. Lorsque la mise à jour du RTT est possible de manière précise, on le repositionne.

1 Calcul du RTT de manière précise.
2 **FAIRE** Toutes les RTT secondes
  **SI** le calcul du RTT n'est pas possible (pas d'information)
  **ALORS** Estimation du RTT (2 possibilités).
    (RTT = (1 - α) RTT + α (2 x *TempsEmetteurRecepteur*))
    ou
    (RTT = (1 - α) RTT + α (t1' + t2 * t1'/t1))
  **SINON** Calcul du RTT de manière précise.
**FINFAIRE**

## *8.7 Le Tocken Bucket*

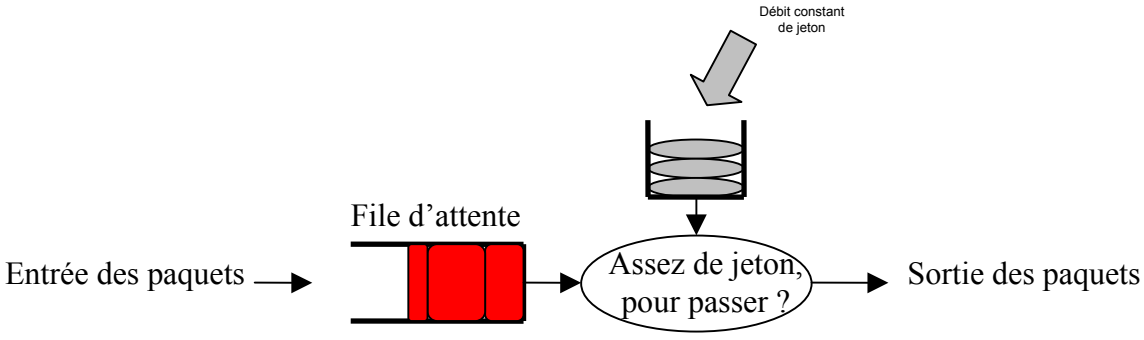



**Figure 44 *Tocken Bucket***

Le *Token Bucket* [Figure 44] à un rôle de mise en forme d'un flux. C'est-à-dire qu'il lui impose un certain débit. Le principe est simple, un seau de jeton se rempli à un débit constant. Lorsqu'un paquet arrive, s'il y a suffisamment de jeton dans le seau, alors le paquet est envoyé sur la sortie et le seau se voit vidé du nombre de jeton nécessaire à ce passage. Dans le cas ou il n'y à pas assez de jeton, le paquet attend dans la file d'attente. Plusieurs solutions peuvent alors être envisagées pour les paquets de la file d'attente. Soit il reste là, en attente de

jeton soit on les envois sur une autre sortie du *Tocken Bucket*. On peut aussi décider de ne les envoyer sur une autre sortie que si la file d'attente est saturée.

Evidement, la description est très imagée, le *Token Bucket* est en réalité une variable qui contient un nombre d'octets qui augmentent à débit constant. Arrivé à une valeur maximum, cette variable n'est plus augmentée. Lorsqu'un paquet arrive dans la file d'attente, on vérifie si la taille du paquet est inférieure à la valeur de la variable. Si c'est le cas alors le paquet est envoyé sur la sortie et la variable voit sa valeur diminuée de la taille du paquet. Sinon, le paquet attend dans la file.

## *8.8 L'Algorithme RED*

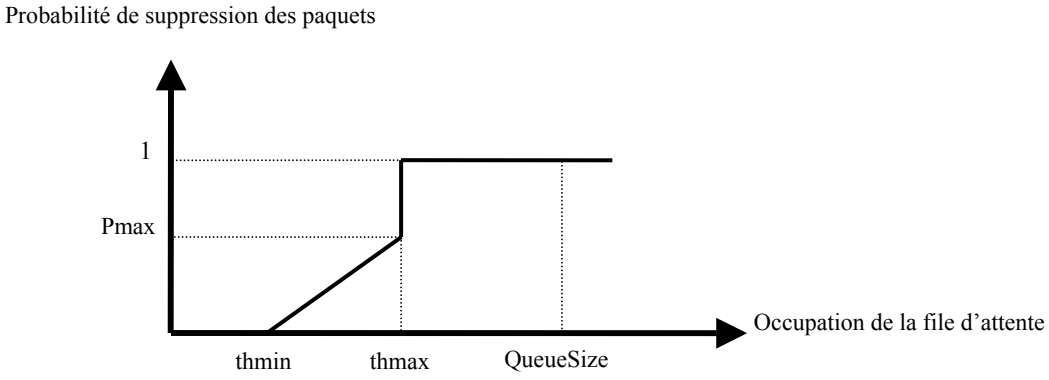



**Figure 45 Algorithme RED**

La figure 45 montre le principe de fonctionnement de l'algorithme de RIO. Les paquets sont jetés avec une probabilité croissante une fois que le seuil d'occupation de la file d'attente est dépassé. On peut remarquer que le calcul de l'occupation de la file d'attente est une valeur moyenne et non instantanée. Cela permet d'éviter des variations de probabilité de suppression trop brusque.

L'intérêt de RED est d'absorber de la meilleure façon possible l'arrivée de paquet en rafale. En effet, sans un tel mécanisme, les paquets sont tout simplement supprimés une fois que la file d'attente est pleine. Cependant, les expériences montrent qu'une file d'attente remplie conserve cet état. Ainsi, le rôle premier de la file d'attente qui est d'absorber les rafales ne fonctionne plus. RED propose une solution qui permet de niveler l'intensité de suppression.

# 9 Bibliographie


[1] Evelyne Michel, « Broadcast, vidéo à la demande, vidéoconférence… De nombreuses applications pour un marché émergeant », Dossier Vidéo d'Entreprise, 1 août 1998.

[2] Sun Microsystems, Inc., « Sun unveils web streaming solution aimed at exploding market for storing and streaming video », Press Releases, 10 avril 2000.

[3] RealSystem G2 SDK, *http://proforma.real.com/mario/devzone/g2sdk.html*.

[4] L. Chiariglione, « MPEG and multimedia communication », www.mpeg.org, août 1996.

[5] D. Y. Pan, « Digital Audio Compression », Digital Technical Journal Vol.5 N°2, été 1993.

[6] Michael Robertson « Top 10 Things Everyone Should Know About MP3 », *www.mp3.com*, 23 juillet 1998.

[7] Fraunhofer-Gesellschaft, « MPEG Audio Layer-3 », *www.iis.fhg.de*.

[8] G. Bouvigne, « Overview of the MP3 technique », *www.mp3-tech.org*.

[9] D.Thom, H. Purnhagen, S. Pfeiffer, « MPEG Audio FAQ », MPEG Audio Subgroup, *www.tnt.uni-hanover.de*, décembre 1999.

[10] W. Howit, « Sub-Band Coding », *www.otolith.com/pub/u/howitt/sbc.tutorial.html*, 1995.

[11] Pedrag Supurovic, « MPEG Audio Frame Header », *www.dv.co.yu/mpgscript/mpeghdr.htm*, septembre 1998.

[12] D. Pans, « A Tutorial on MPEG/Audio Compression », IEE Multimedia Journal, 7 octobre 1996.

[13] K. Youssef, « Services Multimédia dans les réseaux à différentiation de service », Rapport de stage, juin 1999.

[14] O. Medina, « Utilisation des Sousbandes pour la transmission vidéo multi-couches sur ATM », Rapport de DEA, juin 1995.

[15] « Recommandation UI-T H.262 » c'est à dire « Norme international ISO/CEI 13818-2 » (MPEG2-Partie Vidéo).

[16] « MPEG video compression technique », *rnvs.informatik.tu-chemnitz.de/~ja/MPEG/HTML/mpeg_tech.html*.

[17] I. Busse, B. Deffner, H. Schulzrinne, « Dynamic QoS Control of Multimedia Applications based on RTP », Computer Communication, janvier 1996.

[18] R. Rejaie, M. Handley, D . Estrin, « Architectural Consideration for Playback of Quality Adaptive Video over the Internet », Technical report 98-686, CCS-USC, novembre 1998.

[19] J-C. Bolot, T. Turletti, « Experience with Control Mechanisms for Packet Video in the Internet », 1999.

[20] R. Rejaie, M. Handley, D. Estrin « RAP : An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet », IEE INFOCOM, 1999.

[21] D. Sisalem, F. Emanuel, H. Schulzrinne, « The Direct Adjustment Algorithm : A TCP-Friendly Adaptation Scheme », 6 août 1997.

[22] T. Turletti, S. F. Parisis, J-C Bolot, « Experiements with a Layered Transmission Scheme over the Internet. », Rapport INRIA de recherche, novembre 1997.

[23] S. Casner, V. Jacobson, « Compressing IP/UDP/RTP Headers for Low-Speed Serial Links », Internet Draft, 14 juillet 1997.

[24] R. Rejaie, M. Handley, D. Estrin, « Quality Adaptation for Congestion Controlled Video Payback over the Internet », ACM SIGCOMM, volume 29, n°4, Computer Communication Review, octobre 1999.

[25] J-C Bolot, « Adaptive Applications Tutorial », http://www.inira.fr/rodeo/bolot, 1999.

[26] H. ElGebaly, « Reactive Mechanisms for Recovering Audio Performance in Multimedia Conferencing Over Packet Switched Networks », Intel Technology Journal Q3, 1999.

[27] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, « A Transport Protocol for Real-Time Applications», RFC 1889, janvier 1996.

[28] J M Boyce, R. D. Gaglianello, « Packet Loss Effects on MPEG Video Sent Over the Public Internet », ACM Multimedia 98 – Electronic Proceeding.

[29] C. Perkins, O. Hodson, « Option for Repair of Streaming Media », RFC 2354, juin 1998

[30] J. Rosenberg, H.Schulzrinne, « A RTP Payload Format for Generic Forward Error Correction », RFC 2733, décembre 1999.

[31] D. Hoffman, G. Fernando, V. Goyal, M. Civanlar, « RTP Payload Format for MPEG1/MPEG2 Video », RFC 2250, janvier 1998.

[32] M. Civanlar, G. Cash, B. Haskell, « RTP Payload Format for Bundled MPEG », RFC2343, mai 1998.

[33] A. Basso, G. L. Cash, M. R, Civanlar, « Transmission of MPEG-2 Streams over Non-Guaranteed Quality of Service Networks », Processing of Picture Coding Symposium (Belin, Germany), septembre 1997, (*www.cs.columbia.edu/~hgs/rtp/papers)*.

[34] M. F. Speer, S. McCanne, « RTP usage with layered Multimedia Streams », draft, 20 décembre 1996.

[35] Jean-Marie Rifflet, « La communication sous Unix », Applications réparties, ediscience international, 1996.

[36] A. Basso, G. L. Cash, M. R. Civanlar, « Implementation of a Real-Time MPEG-2 Delivery sytem based on RTP », 1998.

[37] S. Blake, D. Black, M.Carlson, E. Davies, Z. Wang, W. Weiss, « An architecture for Differentiated Services », RFC2475, décembre 1998.

[38] J-P. Soulès, "IP peut aussi prétendre à la qualité de service", 01 Informatique n°1574, 4 février 2000.

[39] N. Nichols, S. Blake, F. Baker, D. Black, « Definition of the Differentiated Service Field (DS Field) in the IPv4 and IPv6 Headers », RFC2474, décembre 1998

[40] J. Heinanen, F. Baker, W. Weiss, J.Wroclawski, « Assured Forwarding PHB group », RFC2597, juin 1999

[41] V.Jacobson, K. Nichols, K. Poduri, « An Expedited Forwarding PHB », RFC2598, Juin 1999

[42] K. Nichols, V. Jacobson, L.Zhang, « A Two-bit Differentiated Services Architecture for Internet », Article, 1997

[43] Y. Bernet, A. Smith, S.Blake, « A Conceptual Model for Diffserv Routers », Internet Draft, octobre 1999.

[44] C. Dovrolis, D. Stiliadis, P. Ramanathan, « Proportional Differentiated Services : Delay Differentiation and Packet Scheduling », 1998.

[45] R. Finlayson, « A More Loss-Tolerant RTP Payload Format for MP3 Audio », Internet-Draft, 21 octobre 1999.