Groupe de travail Réseau
Request for Comments : 5219
RFC rendue obsolète : 3119
Catégorie : Sur la voie de la normalisation

R. Finlayson, Live Networks, Inc.

février 2008
Traduction Claude Brière de L'Isle

Formats de charge utile RTP plus tolérants à la perte pour l'audio MP3

Statut du présent mémoire
Le présent document spécifie un protocole Internet sur la voie de la normalisation pour la communauté de l'Internet, et appelle à des discussions et suggestions pour son amélioration. Prière de se référer à l'édition en cours des "Normes officielles des protocoles de l'Internet" (STD 1) pour connaître l'état de la normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
Le présent document décrit un format de charge utile RTP (protocole de transport en temps réel) pour transporter l'audio de couche 3 du groupe d'experts en images animées (MPEG, *Moving Picture Experts Group*) (généralement appelé "MP3"). Ce format est une solution de remplacement de celui décrit dans la RFC 2250, et a de meilleures performances si il y a des pertes de paquets. Le présent document rend obsolète la RFC 3119, corrigeant des erreurs typographiques dans la section "Usage de SDP" et dans le pseudo-code des appendices.

Table des Matières

1. Introduction

Bien que le format de charge utile RTP défini dans la [RFC2250] soit généralement applicable à toutes les formes d'audio ou vidéo MPEG, il est sous optimal pour l'audio MPEG-1 ou 2, de couche 3 (généralement appelé "MP3"). La raison en est qu'une trame MP3 n'est pas une vraie "unité de données d'application" -- elle contient un pointeur arrière sur les données de trames antérieures, et donc ne peut pas être décodée indépendemment de ces trames antérieures. Parce que la RFC 2250 définit que les limites de paquet coincident avec les limites de trame, elle traite la perte de paquet de façon inefficace quand

on porte des données MP3. La perte d'une trame MP3 va rendre certaines données de trames antérieures (ou futures) sans objet, même si elles sont reçues sans pertes.

Dans le présent document, on définit un format de charge utile RTP de remplacement pour l'audio MP3. Ce format utilise un réarrangement préservant les données des trames MPEG originales, afin que les limites de paquet coincident maintenant avec les vraies "unités de données d'application" MP3, qui peuvent aussi (facultativement) être réarrangées dans un schéma d'entrelaçage. Ce nouveau format est donc plus efficace que celui de la RFC 2250 en présence de pertes de paquets.

2. Terminologie

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

3. Structure de la trame MP3

Dans cette section, on donne une brève vue d'ensemble de la structure d'une trame MP3. (Pour une description plus détaillée, voir les spécifications MPEG-1 audio [ISO11172-3] et MPEG-2 audio [ISO13818-3].)

Chaque trame audio MPEG commence par un en-tête de 4 octets. Les informations définies par cet en-tête incluent :
- si l'audio est MPEG-1 ou MPEG-2,
- si l'audio est de couche I, II, ou III. (Le reste de ce document suppose la couche III, c'est-à-dire, des "trames MP3".)
- si l'audio est mono ou stéréo,
- si il y a ou non un champ de 2 octets de CRC à la suite de l'en-tête,
- (indirectement) la taille de la trame.

Les structures suivantes apparaissent après l'en-tête :
- (facultativement) un champ de 2 octets de contrôle de redondance cyclique (CRC, *Cyclic Redundancy Check*)
- une structure "d'informations annexes". Celles-ci ont la longueur suivante :
  - 32 octets pour MPEG-1 stéréo,
  - 17 octets pour MPEG-1 mono, ou pour MPEG-2 stéréo,
  - 9 octets pour MPEG-2 mono.
- des données audio codées, plus des données auxiliaires facultatives (remplissant le reste de la trame).

Pour les besoins du présent document, la structure "d'informations annexes" est la plus importante, parce qu'elle définit la localisation et la taille des unités de données d'application (ADU, *Application Data Unit*) qu'un décodeur MP3 va traiter. En particulier, la structure "d'informations annexes" définit :

- "main_data_begin" *(début des données principales)* : c'est un pointeur arrière (en octets) sur le début de l'ADU. Le pointeur arrière est compté depuis le début de la trame, et compte seulement les données audio codées et toutes données auxiliaires (c'est-à-dire, en ignorant tout champ d'en-tête, CRC, ou "d'informations annexes").

Un décodeur MP3 traite chaque ADU indépendamment. Les ADU vont généralement varier en longueur, mais leur longueur moyenne va, bien sûr, être celle des trames MP3 (moins la longueur des champ d'en-tête, de CRC, et "d'informations annexes"). (Dans la littérature MPEG, cette ADU est parfois appelée un "réservoir de bits".)

4. Nouveau format de charge utile

Comme noté dans la [RFC2736], un format de charge utile devrait être conçu de façon à ce que les limites de paquet coincident avec les "limites de trame de codec" -- c'est-à-dire, avec les ADU. Dans le format de charge utile pour MPEG audio de la [RFC2250], chaque charge utile de paquet RTP contient des trames MP3. Dans ce nouveau format de charge utile pour MP3 audio, cependant, chaque charge utile de paquet RTP contient des "trames ADU", précédées chacune par un "descripteur d'ADU".

4.1 Trames ADU

Une "trame ADU" est définie comme :

- L'en-tête MPEG de 4 octets (le même que celui de la trame MP3 originale, sauf que les onze premiers bits sont (facultativement) remplacés par un "numéro de séquence d'entrelaçage", comme décrit à la Section 7).
- Le champ facultatif de CRC de deux octets (le même que celui de la trame MP3 originale).
- La structure "d'informations annexes" (la même que celle de la trame MP3 originale).
- La séquence complète de données codées audio (et toutes données auxillaires) pour l'ADU (c'est-à-dire, allant du début du pointeur arrière de "main_data_begin" *(début des données principales)* de cette trame MP3, jusqu'au début du pointeur arrière de la prochaine trame MP3).

Noter qu'il y a une transposition bi-univoque entre les trames MP3 et les trames d'ADU. Parce que les trames MP3 sont auto-descriptives, avec le débit binaire (et la fréquence d'échantillonnage) codé dans les quatre octets de l'en-tête MPEG, la même chose est vraie pour les trames d'ADU. Donc, comme avec les flux MP3, le débit binaire peut changer dans un flux et peut être utilisé pour le contrôle d'encombrement.

4.2 Descripteurs d'ADU

Dans chaque charge utile de paquet RTP, chaque "trame ADU" est précédée d'un "descripteur d'ADU" d'un ou deux octets, qui donne la taille de l'ADU et indique si les données de ce paquet sont ou non une continuation des données du paquet précédent. (Cela se produit seulement quand un seul "descripteur d'ADU" + "trame ADU" est trop grand pour tenir dans un paquet RTP.)

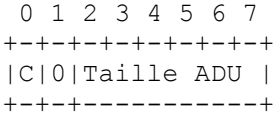
Un descripteur d'ADU consiste en les champs suivants :

- Fanion "C" (Continuation) 1 bit : à 1 si les données qui suivent le descripteur d'ADU sont la continuation d'une trame ADU trop grande pour tenir dans un seul paquet RTP ; 0 autrement.

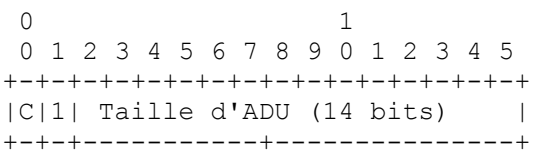
- Fanion "T" (Type de déscripteur) 1 bit : 0 pour un descripteur d'ADU d'un octet ; 1 pour un descripteur d'ADU de deux octets.

- "Taille d'ADU" (6 ou 14 bits) : taille (en octets) de la trame ADU qui va suivre ce descripteur d'ADU (c'est-à-dire, NON incluse la taille du descripteur lui-même). Un descripteur d'ADU de deux octets (avec un champ "taille d'ADU" de 14 bits) est utilisé pour les tailles de trame d'ADU de 64 octets ou plus. Pour les plus petites tailles de trame d'ADU, les envoyeurs PEUVENT autrement utiliser un descripteur d'ADU de un octet (avec un champ "taille d'ADU" de 6 bits). Les receveurs DOIVENT être capables d'accepter un descripteur d'ADU de l'une ou l'autre taille.

Donc, un descripteur d'ADU d'un octet est formaté comme suit :



et un descripteur d'ADU de deux octets est formaté comme suit :



4.3 Règles de mise en paquet

Chaque charge utile de paquet RTP commence par un "descripteur d'ADU", suivi par des données de "trame d'ADU". Normalement, ce "descripteur d'ADU" + "trame ADU" va tenir complètement dans le paquet RTP. Dans ce cas, plus d'un "descripteur d'ADU" + "trame ADU" successifs PEUVENT être empaquetés dans un seul paquet RTP, pourvu qu'ils y tiennent tous complètement.

Si, cependant, un seul "descripteur d'ADU" + "trame ADU" est trop grand pour tenir dans un paquet RTP, alors la "trame d'ADU" est partagée en deux paquets RTP successifs ou plus. Chacun de ces paquets commence par un descripteur d'ADU. Le descripteur du premier paquet a un fanion "C" (continuation) de 0 ; les descripteurs des paquets suivants ont chacun un fanion "C" de 1. Chaque descripteur, dans ce cas, a la même valeur de "taille d'ADU" : la taille de la "trame d'ADU" entière

(pas juste la portion qui tient dans un seul paquet RTP). Chacun de ces paquets (même le dernier) contient seulement un "descripteur d'ADU".

4.4 Champs d'en-tête RTP

Type de charge utile : le type (statique) de charge utile de 14 qui a été défini pour MPEG audio [RFC3551] NE DOIT PAS être utilisé. À la place, un type différent de charge utile dynamique DOIT être utilisé -- c'est-à-dire, un dans la gamme de [96 à 127].

Bit M : ce format de charge utile ne définit pas d'utilisation pour ce bit. Les envoyeurs DEVRAIENT le régler à zéro dans chaque paquet sortant.

Horodatage : c'est un horodatage de 32 bits, à 90 kHz, qui représente l'heure de présentation de la première ADU dans ce paquet.

4.5 Traitement des données reçues

Noter qu'aucune information n'est perdue par la conversion d'une séquence de trames MP3 en une séquence correspondante de "trames d'ADU", donc une mise en œuvre RTP receveuse peut soit fournir directement les trames ADU à un décodeur MP3 modifié de façon appropriée, soit les reconvertir en une séquence de trames MP3, comme décrit à l'Appendice A.2.

5. Traitement de plusieurs couches audio MPEG

Le format de charge utile RTP décrit ici est destiné seulement à l'audio MPEG-1 ou 2, de couche 3 ("MP3"). À l'opposé, les trames de couche 1 et 2 sont auto-contenues, sans un pointeur arrière sur les trames antérieures. Cependant, il est possible (bien qu'inhabituel) qu'une séquence de trames audio consiste en un mélange de trames de couche 3 et de trames de couche 1 ou 2. Quand une telle séquence est transmise, seules les trames de couche 3 sont converties en ADU ; les trames de couche 1 ou 2 sont envoyées "telles qu'elles" (sauf pour l'ajout d'un "descripteur d'ADU"). De même, le receveur d'une séquence de trames -- utilisant ce format de charge utile -- laisse les trames 1 et 2 inchangées (après avoir retiré le "descripteur d'ADU" ajouté devant) mais convertit les trames de couche 3 de "trames ADU" en trames MP3 régulières. (On rappelle que la couche de chaque trame est identifiée à partir de son en-tête MPEG de quatre octets.)

Si on transmet un flux consistant *seulement* en trames de couche 1 ou 2 (c'est-à-dire, sans aucune données MP3) il n'y a alors aucun avantage à utiliser ce format de charge utile, *sauf* si on utilise le mécanisme d'entrelaçage décrit à la Section 7.

6. Empaquetage et désempaquetage de trames

La transmission d'une séquence de trames MP3 suit les étapes suivantes :

Trames MP3
-1-> trames ADU
-2-> trames ADU entrelacées
-3-> paquets RTP

L'étape 1 est la conversion d'une séquence de trames MP3 en une séquence correspondante de trames d'ADU, et a lieu comme décrit dans la Section 3 et au paragaphe 4.1. (Noter aussi le pseudo code de l'Appendice A.1.)

L'étape 2 est le réarrangement de la séquence de trames d'ADU dans un schéma (facultatif) d'entrelaçage, avant la mise en paquets, comme décrit à la Section 7. (Noter aussi le pseudo code de l'Appendice B.1.) L'entrelaçage aide à réduire l'effet de la perte de paquet en répartissant les trames ADU consécutives dans des paquets non consécutifs. (Noter qu'à cause du pointeur arrière dans les trames MP3, l'entrelaçage ne peut être appliqué -- en général -- qu'aux trames ADU. Donc, l'entrelaçage n'était pas possible pour la RFC 2250.)

L'étape 3 est la mise en paquets d'une séquence de trames d'ADU entrelacées en paquets RTP -- comme décrit au paragraphe 4.3. L'horodatage RTP de chaque paquet est l'heure de présentation de la première ADU qui est empaquetée dedans. Noter que si l'entrelaçage a été fait dans l'étape 2, les horodatages RTP sur les paquets sortants ne vont pas nécessairement être monotonement non décroissants.

De même, une séquence de paquets RTP reçue est traitée comme suit :

Paquets RTP
-4-> paquets RTP ordonnés par numéro de séquence RTP
-5-> trames ADU entrelacées
-6-> trames ADU
-7-> trames MP3

L'étape 4 est le tri usuel des paquets RTP entrants utilisant le numéro de séquence RTP.

L'étape 5 est le désempaquetage des trames ADU des paquets RTP -- c'est-à-dire, l'inverse de l'étape 3. Au titre de ce processus, un receveur utilise le fanion "C" (continuation) dans le descripteur d'ADU pour noter quand une trame ADU est partagée sur plus d'un paquet (et pour éliminer entièrement la trame d'ADU si un de ces paquets est perdu).

L'étape 6 est le réarrangement de la séquence des trames ADU dans son ordre original (sauf pour les trames d'ADU manquante à cause d'une perte de paquet) comme décrit à la Section 7. (Noter aussi le pseudo-code à l'Appendice B.2.)

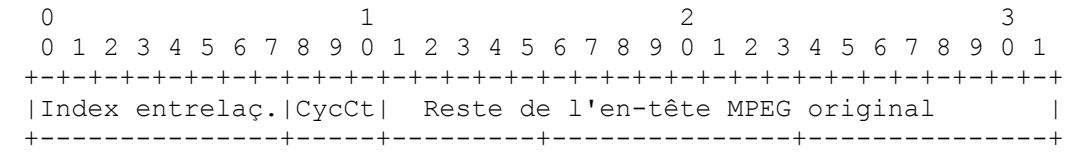
L'étape 7 est la conversion de la séquence de trames ADU en une séquence correspondante de trames MP3 -- c'est-à-dire, l'inverse de l'étape 1. (Noter aussi le pseudo-code de l'Appendice A.2.) Avec un décodeur MP3 modifié de façon appropriée, une mise en œuvre peut omettre cette étape ; à la place elle pourrait envoyer les trames ADU directement au décodeur MP3 (modifié).

7. Entrelaçage de trames ADU

Dans les trames audio MPEG (MPEG-1 ou 2 ; toutes couches) les 11 bits de poids fort de l'en-tête MPEG de quatre octets ("syncword") sont toujours tous à un (c'est-à-dire, 0xFFE). Quand on réordonne une séquence de trames d'ADU pour la transmission, on réutilise ces 11 bits comme un "numéro de séquence d'entrelaçage" (ISN, *Interleaving Sequence Number*). (À réception, ils sont remplacés à nouveau par des 0xFFE.)

La structure de l'ISN est (a,b), où :
- a == bits 0 à 7 : 8 bits d'indice d'entrelaçage (dans le cycle)
- b == bits 8 à 10 : 3 bits de compte de cycle d'entrelaçage

C'est-à-dire que les 4 octets de l'en-tête MPEG sont réutilisés comme suit :



Exemple : Considérons le cycle d'entrelaçage suivant (de taille 8) : 1,3,5,7,0,2,4,6
(Ce schéma particulier a la propriété que toute perte de jusqu'à quatre ADU consécutives dans le flux entrelacé va conduire à un flux désentrelacé sans trou supérieur à un.) Cela produit la séquence d'ISN suivante :

(1,0) (3,0) (5,0) (7,0) (0,0) (2,0) (4,0) (6,0) (1,1) (3,1) (5,1) etc.

Donc, dans cet exemple, une séquence de trames d'ADU de f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 (etc.) va être réordonnée dans l'étape 2, en : (1,0)f1 (3,0)f3 (5,0)f5 (7,0)f7 (0,0)f0 (2,0)f2 (4,0)f4 (6,0)f6 (1,1)f9 (3,1)f11 (5,1)f13 (etc.) et le réarrangement inverse (avec le remplacement des 0xFFE) va se faire à la réception.

La raison pour casser l'ISN en "Compte de cycle d'entrelacement" et "Indice d'entrelacement" (plutôt que de juste le traiter comme un seul compteur de 11 bits) est de donner aux receveurs un moyen de savoir quand une trame d'ADU devrait être "libérée" en le processus de conversion d'ADU en MP3 (l'étape 7 ci-dessus) plutôt que d'attendre que plus de trames d'ADU entrelacées arrivent. Par exemple, dans l'exemple ci-dessus, quand le receveur voit une trame avec l'ISN (<quelquechose>,1) il sait qu'il peut libérer toutes les trames vues précédemment avec l'ISN (<quelquechose>,0) même si une autre trame (<quelquechose>,0) manque encore à cause d'une perte de paquet. Un indice d'entrelaçage de 8 bits permet des cycles d'entrelaçage d'une taille jusqu'à 256.

Le choix d'un ordre d'entrelaçage peut être fait indépendamment de la mise en paquets RTP. Donc, une simple mise en œuvre pourrait choisir d'abord un ordre d'entrelaçage, réordonner les trames d'ADU en conséquence (étape 2) puis simplement les empaqueter en séquence dans des paquets RTP (étape 3). Cependant, la taille des trames d'ADU -- et donc le nombre de trames d'ADU qui vont tenir dans chaque paquet RTP -- va normalement varier en taille, donc une mise en œuvre plus optimale va combiner les étapes 2 et 3, en choisissant un ordre d'entrelaçage qui reflète mieux le nombre de trames ADU empaquetées dans chaque paquet RTP.

Chaque mise en œuvre receveuse de ce format de charge utile DOIT reconnaître l'ISN et être capable d'effectuer le désentrelaçage des trames d'ADU entrantes (étape 6). Cependant, une mise en œuvre envoyeuse de ce format de charge utile PEUT choisir de ne pas effectuer d'entrelaçage -- c'est-à-dire, d'omettre l'étape 2. Dans ce cas, les 11 bits de poids fort dans chaque en-tête MPEG de quatre octets vont rester à 0xFFE. Les mises en œuvre receveuses vont donc voir une séquence d'ISN identiques (tout à 0xFFE). Elles vont traiter cela de la même façon que si le compte de cycle d'entrelaçage changeait à chaque trame d'ADU, en livrant simplement la séquence de trames d'ADU entrantes en séquence au processus de conversion d'ADU en MP3 (étape 7) sans réarrangement. (Noter aussi le pseudo-code de l'Appendice B.2.)

8. Considérations relatives à l'IANA

Nom de type de support : audio
Nom de sous type de support : mpa-robust
Paramètres exigés : aucun
Paramètres facultatifs : aucun
Considérations de codage : ce type est défini seulement pour le transfert via RTP, comme spécifié dans la RFC 5219.
Considérations de sécurité : voir la section de "Considérations sur la sécurité" de la RFC 5219.
Considérations d'interopérabilité : ce codage est incompatible avec les types de support "audio/mpa" et "audio/mpeg".
Spécification publiée : spécification audio ISO/CEI MPEG-1 [ISO11172-3] et MPEG-2 [ISO13818-3], et RFC 5219.
Applications qui utilisent ce type de support : outils de flux audio (émission et réception).
Informations supplémentaires : aucune.
Personne & adresse de messagerie à contacter pour plus d'informations : Ross Finlayson, finlayson@live555.com
Usage prévu : COMMUN
Auteur : Ross Finlayson
Contrôleur des changements : groupe de travail IETF AVT.

9. Utilisation de SDP

Quand les informations sont portées par SDP [RFC4566], le nom de codage DEVRA être "mpa-robust" (le même que le sous type de support). Un exemple de la représentation du support en SDP est :

```
m=audio 49000 RTP/AVP 121
a=rtpmap:121 mpa-robust/90000
```

Noter que la fréquence d'horodatage RTP DOIT être 90 000.

10. Considérations sur la sécurité

Si une session qui utilise ce format de charge utile est chiffrée, et si l'entrelaçage est utilisé, alors l'envoyeur DEVRAIT s'assurer que tout changement de clé de chiffrement coincide avec un début de nouveau cycle d'entrelaçage. À part cela, les considérations de sécurité pour ce format de charge utile sont identiques à celles notées pour la [RFC2250].

11. Remerciements

La suggestion d'ajouter une option d'entrelaçage (utilisant les premiers bits du "syncword" MPEG -- qui serait autrement tout de uns -- comme indice d'entrelaçage) est due à Dave Singer et Stefan Gewinner. De plus, Dave Singer a fourni des retours précieux qui ont aidé à préciser et améliorer la description de ce format de charge utile. Des retours de Chris Sloan ont conduit à l'ajout d'un "descripteur d'ADU" précédant chaque trame ADU dans le paquet RTP.

12. Références normatives

[ISO11172-3] Norme internationale ISO/CEI 11172-3; " Technologie de l'information -- Codage des images animées et de l'audio associé pour support de mémorisation numérique jusqu'à environ 1,5 Mbits/s -- Partie 3 : Audio", Organisation internationale de normalisation (ISO), 1993.

[ISO13818-3] Norme internationale ISO/CEI 13818-3, "Technologie de l'information -- Codage générique des images animées et des informations audio associées -- Partie 3 : Audio", Organisation internationale de normalisation (ISO), 1998.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC2250] D. Hoffman et autres, "Format de charge utile RTP pour la vidéo MPEG1/MPEG2", janvier 1998.

[RFC2736] M. Handley, C. Perkins, "Lignes directrices pour les rédacteurs de spécifications de format de charge utile RTP", décembre 1999. (BCP0036, *MàJ par* RFC8088)

[RFC3551] H. Schulzrinne et S. Casner, "Profil RTP pour conférences audio et vidéo avec contrôle minimal", STD 65, juillet 2003. *(MàJ par* RFC8860)

[RFC4566] M. Handley, V. Jacobson et C. Perkins, "SDP : Protocole de description de session", juillet 2006. *(P.S. ; remplacée par RFC8866)*

Appendice A. Traduction de "trame MP3" en "trame d'ADU"

Le "pseudo code" suivant décrit comment un envoyeur qui utilise ce format de charge utile peut traduire une séquence de "trames MP3" régulières en "trames d'ADU", et comment un receveur peut effectuer la traduction inverse, de "trames d'ADU" en "trames MP3".

On définit d'abord les structures de données abstraites suivantes :

- "Segment" : un enregistrement qui représente une "trame MP3" ou une "trame d'ADU". Il comporte les champs suivants :
  - "header" : en-tête MPEG de quatre octets.
  - "headerSize" : constante (== 4).
  - "sideInfo" : structure "informations annexes", *incluant* le champ facultatif de 2 octets de CRC, s'il est présent.
  - "sideInfoSize" : taille (en octets) de la structure ci-dessus.
  - "frameData" : données restantes dans cette trame.
  - "frameDataSize" : taille (en octets) des données ci-dessus.
  - "backpointer" : valeur (exprimée en octets) du pointeur arrière pour cette trame.
  - "aduDataSize" : taille (en octets) de l'ADU associée à cette trame. (Si la trame est déjà une "trame d'ADU", alors aduDataSize == frameDataSize)
  - "mp3FrameSize" : taille totale (en octets) qu'aurait cette trame si c'était une "trame MP3" régulière. (Si c'est déjà une "trame MP3", alors mp3FrameSize == headerSize + sideInfoSize + frameDataSize) Noter que cette taille peut être déduite complètement de "header".

- "SegmentQueue" : file d'attente FIFO de "Segments", avec les opérations
  - void enqueue(Segment) *(file d'attente vide)*
  - Segment dequeue() *(segment sorti de la file d'attente)*
  - Boolean isEmpty() *(est vide ou non)*
  - Segment head() *(segment de tête)*
  - Segment tail() *(segment de queue)*
  - Segment previous(Segment) : retourne le segment antérieur à celui-ci.
  - Segment next(Segment) : retourne le segment après celui-ci.
  - unsigned totalDataSize() : retourne la somme des champs "frameDataSize" de chaque entrée de la file d'attente.

A.1 Conversion d'une séquence de "trames MP3" en une séquence de "trames d'ADU"

SegmentQueue pendingMP3Frames;

```
// initiallement vide pendant (1) {
// Mise en file d'attente des nouvelles trames MP3, jusqu'à ce qu'il y ait assez de données pour générer l'ADU pour une trame :
        faire {
                int totalDataSizeBefore
                        = pendingMP3Frames.totalDataSize();

                Segment newFrame = "prochaine trame MP3";
                pendingMP3Frames.enqueue(newFrame);

                int totalDataSizeAfter = pendingMP3Frames.totalDataSize();
        } quand (totalDataSizeBefore < newFrame.backpointer ||
                  totalDataSizeAfter < newFrame.aduDataSize);

// On a maintenant assez de données pour générer l'ADU pour la plus récente trame mise en file d'attente (c'est-à-dire, la queue de la file).
// (Les trames antérieures dans la file d'aattente -- si il en est -- doivent être éliminées, parce qu'on a pas assez de données pour générer leurs ADU.)
        Segment tailFrame = pendingMP3Frames.tail();

// Sortir l'en-tête et les informations annexes :
        output(tailFrame.header);
        output(tailFrame.sideInfo);

// Revenir à la trame qui contient le début des données d'ADU :
        int offset = 0;
        Segment curFrame = tailFrame;
        int prevBytes = tailFrame.backpointer;
        quand (prevBytes > 0) {
                curFrame = pendingMP3Frames.previous(curFrame);
                int dataHere = curFrame.frameDataSize;
                si (dataHere < prevBytes) {
                        prevBytes -= dataHere;
                } autrement {
                        décalage = dataHere - prevBytes;
                        couper ;
                }
        }

// Sortir de la file d'attente toutes les trames dont on n'a plus besoin :
        quand (pendingMP3Frames.head() != curFrame) {
                pendingMP3Frames.dequeue();
        }

// Sortir, à partir des trames restantes, les données d'ADU qu'on veut :
        int bytesToUse = tailFrame.aduDataSize;
        quand (bytesToUse > 0) {
                int dataHere = curFrame.frameDataSize - décalage;
                int bytesUsedHere
                        = dataHere < bytesToUse ? dataHere :
                        bytesToUse;

                sortir("bytesUsedHere" octets de curFrame.frameData, en commençant à partir de "décalage");
                bytesToUse -= bytesUsedHere;
                décalage = 0;
                curFrame = pendingMP3Frames.next(curFrame);
        }
  }
```

A.2 Conversion d'une séquence de "trames ADU" en une séquence de "trames MP3"

```
SegmentQueue pendingADUFrames;              // initiallement vide
  quand (1) {
        quand (needToGetAnADU()) {
              Segment newADU = "prochaine trame d'ADU" ;
              pendingADUFrames.enqueue(newADU);
              insertDummyADUsIfNecessary();
        }
        generateFrameFromHeadADU();
  }

Booléen needToGetAnADU() {
// Vérifier si on a besoin de mettre en file d'attente une ou plusieurs nouvelles ADU avant d'avoir assez de données pour générer une trame pour l'ADU de tête.
        Booléen needToEnqueue = Vrai;

        si (!pendingADUFrames.isEmpty()) {
              Segment curADU = pendingADUFrames.head();
              int endOfHeadFrame = curADU.mp3FrameSize - curADU.headerSize - curADU.sideInfoSize;
              int frameOffset = 0;

              quand (1) {
                    int endOfData = frameOffset - curADU.backpointer + curADU.aduDataSize;
                    si (endOfData >= endOfHeadFrame) {
// On a assez de données pour générer une trame.
                          needToEnqueue = Faux ;
                          couper ;
                    }

                    frameOffset += curADU.mp3FrameSize - curADU.headerSize - curADU.sideInfoSize;
                    si (curADU == pendingADUFrames.tail()) couper ;
                    curADU = pendingADUFrames.next(curADU);
              }
        }
      retourner needToEnqueue;
  }

  void generateFrameFromHeadADU() {
        Segment curADU = pendingADUFrames.head();

// Sortir l'en-tête et les informations annexes :
        output(curADU.header);
        output(curADU.sideInfo);

// Commencer par mettre à zéro le reste de la trame, au cas où les données d'ADU ne la rempliraient pas complètement :
        int endOfHeadFrame = curADU.mp3FrameSize - curADU.headerSize - curADU.sideInfoSize;
        sortir("endOfHeadFrame" octets à zéro);

// Remplir la trame avec les données d'ADU appropriées à partir de cette ADU et des suivantes :
        int frameOffset = 0;
        int toOffset = 0;

        quand (toOffset < endOfHeadFrame) {
              int startOfData = frameOffset - curADU.backpointer;
              si (startOfData > endOfHeadFrame) {
                    couper ;                          // aucune autre ADU n'est nécessaire
              }
              int endOfData = startOfData + curADU.aduDataSize;
              si (endOfData > endOfHeadFrame) {
                    endOfData = endOfHeadFrame;
              }
```

```
            int fromOffset;
            si (startOfData <= toOffset) {
                fromOffset = toOffset - startOfData;
                startOfData = toOffset;
                si (endOfData < startOfData) {
                    endOfData = startOfData;
                }
            } autrement {
                fromOffset = 0;
// laisser des octets à zéro devant :
                toOffset = startOfData;
            }

            int bytesUsedHere = endOfData - startOfData;
            sortir(en commençant au décalage de "toOffset", "bytesUsedHere"
                octets à partir de "&curADU.frameData[fromOffset]");
            toOffset += bytesUsedHere;

            frameOffset += curADU.mp3FrameSize - curADU.headerSize - curADU.sideInfoSize;
            curADU = pendingADUFrames.next(curADU);
        }
        pendingADUFrames.dequeue();
    }

    void insertDummyADUsIfNecessary() {
// Le segment (ADU) de queue est supposé avoir été mis récemment en file d'attente. Si son pointeur arrière chevaucherait les données de l'ADU précédente, on a alors besoin d'insérer devant lui une ou plusieurs ADU "factices" vides. (Cette situation devrait ne se produire que si une ADU intermédiaire manque -- par exemple, à cause de la perte d'un paquet.)
        quand (1) {
            Segment tailADU = pendingADUFrames.tail();
            int prevADUend;                                 // par rapport au début de l'ADU de queue
            si (pendingADUFrames.head() != tailADU) {       // il y a une ADU précédente
                Segment prevADU = pendingADUFrames.previous(tailADU);
                prevADUend = prevADU.mp3FrameSize + prevADU.backpointer - prevADU.headerSize
                    - prevADU.sideInfoSize;
                si (prevADU.aduDataSize > prevADUend) {
// cela ne devrait pas arriver si l'ADU précédente était bien formée
                    prevADUend = 0;
                } autrement {
                    prevADUend -= prevADU.aduDataSize;
                }
            } autrement {
                prevADUend = 0;
            }
            si (tailADU.backpointer > prevADUend) {
// Insérer une ADU "factice" devant la queue. Cette ADU peut avoir le même "header" (et donc, "mp3FrameSize") que l'ADU de queue, mais devrait avoir un "pointeur arrière" de "prevADUend", et un "aduDataSize" de zéro. La façon la plus simple de faire cela est de copier les "sideInfo" de l'ADU de queue, de remplacer la valeur de "main_data_begin" par "prevADUend", et de régler tous les champs de "part2_3_length" à zéro.
            } autrement {
                couper ;                                    // aucune autre ADU factice n'a besoin d'être insérée
            }
        }
    }
```

Appendice B. Entrelaçage et désentrelaçage

Le "pseudo code" suivant décrit comment un envoyeur peut réordonner une séquence de "trames d'ADU" selon un schéma d'entrelaçage (étape 2) et comment un receveur peut effectuer le réarrangement inverse (étape 6).

B.1 Entrelaçage d'une séquence de "trames ADU"

On définit d'abord les structures de données abstraites suivantes :
- "interleaveCycleSize" : un entier dans la gamme de [1 à 256]
-- "interleaveCycle" : matrice, de taille "interleaveCycleSize", contenant une permutation des entiers de l'ensemble [0 à interleaveCycleSize-1] par exemple, si "interleaveCycleSize" == 8, "interleaveCycle" pourrait contenir : 1,3,5,7,0,2,4,6.
- "inverseInterleaveCycle" : matrice contenant l'inverse de la permutation dans "interleaveCycle" -- c'est-à-dire, telle que interleaveCycle[inverseInterleaveCycle[i]] == i.
- "ii" : indice actuel d'entrelaçage (initialement 0).
- "icc" : compte actuel de cycle d'entrelaçage (initialement 0).
- "aduFrameBuffer" : matrice, de taille "interleaveCycleSize", des trames d'ADU qui attendent la mise en paquets.

```
quand (1) {
        int positionOfNextFrame = inverseInterleaveCycle[ii];
        aduFrameBuffer[positionOfNextFrame] = la prochaine trame d'ADU ;
        remplacer les 11 bits de poids fort de l'en-tête MPEG de cette trame par (ii,icc) ;
// Note : S'assurer qu'on laisse les 21 bits restants sans changement.
        si (++ii == interleaveCycleSize) {
// On a fini ce cycle, donc on passe toutes les trames en instance à l'étape de mise en paquets.
                pour (int i = 0; i < interleaveCycleSize; ++i) {
                    passer aduFrameBuffer[i] à l'étape de mise en paquets ;
                }
                ii = 0;
                icc = (icc+1)%8;
        }
  }
```

B.2 Désentrelaçage d'une séquence de "trames ADU" (entrelacées)

On définit d'abord les structures de données abstraites suivantes :
- "ii" : indice d'entrelaçage provenant de la trame ADU entrante actuelle.
- "icc" : compte de cycles d'entrelaçage provenant de la trame ADU entrante actuelle.
- "iiLastSeen" : indice d'entrelaçage vu le plus récemment (initialement, un entier *non* dans la gamme de [0 à 255]).
- "iccLastSeen" : compte de cycles d'entrelaçage vu le plus récemment (initialement, un entier *non* dans la gamme de [0 à 7]).
- "aduFrameBuffer" : matrice, de taille 256, de trames d'ADU (pointeurs sur) qui viennent d'être dépaquetisées (initialement, toutes les entrées sont NULLES)

```
quand (1) {
        aduFrame = la prochaine trame d'ADU provenant de l'étape de dépaquetisation ;
        (ii,icc) = "les 11 bits de poids fort de aduFrame de MPEG.
        "header"; "les 11 bits de poids fort de aduFrame de MPEG.
        "header" = 0xFFE;
// Note : S'assurer de laisser les 21 bits comme ils sont.

        si (icc != iccLastSeen || ii == iiLastSeen) {
// On a commencé un nouveau cycle d'entrelaçage (ou l'entrelaçage n'a pas été utilisé). Libérer toutes les trames ADU en instance à l'étape de conversion ADU->MP3 :
                pour (int i = 0; i < 256; ++i) {
                    si (aduFrameBuffer[i] != NULL) {
                        libérer aduFrameBuffer[i];
                        aduFrameBuffer[i] = NULL;
                    }
                }
        }

        iiLastSeen = ii;
        iccLastSeen = icc;
        aduFrameBuffer[ii] = aduFrame;
  }
```

Appendice C. Changements par rapport à la RFC 3119

Le principal changement par rapport à la RFC 3119 est de corriger le nom de codage dans la section "Utilisation de SDP". Le nom de codage correct est "mpa-robust". Aussi, le terme "type de support" remplace le "type mime". Finalement, quelques corrections d'erreurs mineures et des précisions ont été faites au pseudo code (non normatif) dans les Appendices A et B.

Adresse de l'auteur

Ross Finlayson,
Live Networks, Inc.
650 Castro St., suite 120-196
Mountain View, CA 94041
USA

mél : finlayson@live555.com
URI : http://www.live555.com/

Déclaration complète de droits de reproduction

Copyright (C) The IETF Trust (2008).

Le présent document est soumis aux droits, licences et restrictions contenus dans le BCP 78, et à www.rfc-editor.org, et sauf pour ce qui est mentionné ci-après, les auteurs conservent tous leurs droits.

Le présent document et les informations contenues sont fournis sur une base "EN L'ÉTAT" et le contributeur, l'organisation qu'il ou elle représente ou qui le/la finance (s'il en est), la INTERNET SOCIETY et la INTERNET ENGINEERING TASK FORCE déclinent toutes garanties, exprimées ou implicites, y compris mais non limitées à toute garantie que l'utilisation des informations encloses ne viole aucun droit ou aucune garantie implicite de commercialisation ou d'aptitude à un objet particulier.

Propriété intellectuelle
L'IETF ne prend pas position sur la validité et la portée de tout droit de propriété intellectuelle ou autres droits qui pourraient être revendiqués au titre de la mise en œuvre ou l'utilisation de la technologie décrite dans le présent document ou sur la mesure dans laquelle toute licence sur de tels droits pourrait être ou n'être pas disponible ; pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur le répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr.

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf-ipr@ietf.org.