RFC 6716 : Definition of the Opus Audio Codec

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 11 septembre 2012

Date de publication du RFC : Septembre 2012

https://www.bortzmeyer.org/6716.html

Il y a désormais plus de deux ans que le groupe de travail codec <http://tools.ietf.org/wg/codec> de l'IETF avait commencé un effort <https://www.bortzmeyer.org/ietf-codec-libre.html> inédit de spécification d'un codec audio libre. Ce nouveau RFC est le couronnement de cet effort : Opus, le codec standard et libre est désormais officiel.

Opus n'est pas un format de fichiers, comme MP3. Il est conçu pour l'audio en temps réel (RFC 6366 [1] pour son cahier des charges). Il est prévu pour un grand nombre d'utilisations audio distinctes : téléphonie sur IP, mais aussi, par exemple, distribution de musique en temps réel. Il est prévu pour être utilisable lorsque la capacité du réseau est minimale (dans les 6 kb/s) aussi bien que lorsqu'elle permet de faire de la haute fidélité (mettons 510 kb/s, cf. section 2.1.1). Ses deux principaux algorithmes sont la prédiction linéaire (plus efficace pour la parole) et la transformation en cosinus discrète (plus efficace pour la musique).

Le projet codec <https://www.bortzmeyer.org/ietf-codec-libre.html> avait déjà produit deux RFC : le RFC 6366 était le cahier des charges du nouveau codec, et le RFC 6569 posait les principes de développement et d'évaluation dudit codec. Je renvoie à ces documents pour l'arrière-plan de la norme Opus. Celle-ci a été acceptée sans problèmes par le groupe de travail (c'est plutôt le mécanisme de tests qui a suscité des controverses.)

Opus présente une particularité rare à l'IETF : la norme est le code. Habituellement, à l'IETF, la norme est le texte du ou des RFC, rédigés en langue naturelle. Il existe souvent une implémentation de référence mais, en cas de désaccord entre cette implémentation et le RFC, c'est ce dernier qui gagne et le programme doit être modifié. Pour Opus, c'est tout le contraire. La moitié du RFC 6716 (180 pages sur

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6366.txt

330, en annexe A, soit 30 000 lignes de code pas toujours bien commentées et avec plein de nombres magiques) est constituée par l'implémentation de référence (en C, langage que le RFC qualifie de « lisible par un humain », ce qui est parfois optimiste) et c'est **elle qui fait autorité** en cas de différence avec l'autre moitié, les explications (voir sections 1 et 6 du RFC). Ce point a suscité beaucoup de débats à l'IETF et on trouve trace des critiques, entre autres, dans l'évaluation détaillée qui avait été faite pour GenArt <http://www.ietf.org/mail-archive/web/ietf/current/msg73333.html>. L'idée de ce choix « le code est la norme » était que l'alternative (faire une spécification en langue naturelle, complète et sans ambiguité), serait bien plus de travail et, dans le cas particulier d'un codec audio, serait peut-être même irréaliste. (À noter que seul le décodeur est normatif. On peut améliorer la sortie de l'encodeur, tant que le décodeur arrive à lire ce qu'il produit.)

Le langage C étant ce qu'il est, il n'est pas étonnant que des bogues aient été découvertes depuis, corrigées dans le RFC 8251.

L'autre moitié du RFC est composé d'explications, avec beaucoup de maths.

Un codec libre est difficile à obtenir, surtout vu le nombre de brevets, la plupart futiles, qui encombrent ce domaine. Plusieurs déclarations de brevet ont été faites contre Opus : #1520 <https://datatracker.ietf.org/ipr/1520/>, #1524 <https://datatracker.ietf.org/ipr/1524/>, #1526 <https://datatracker.ietf.org/ipr/1526/>, #1602 <https://datatracker.ietf.org/ipr/1602/>, #1670 <https://datatracker.ietf.org/ipr/1670/>, #1712 <https://datatracker.ietf.org/ipr/1712/> et #1741 <https://datatracker.ietf.org/ipr/1741/>. Une liste mise à jour est disponible à l'IETF <https://datatracker.ietf.org/ipr/search/?option=rfc_search&rfc_search=6716>.

Comme ces codecs sont très loin de mes domaines de compétence, je ne vais pas me lancer dans des explications détaillées du fonctionnement d'Opus : lisez le code source :-) Notez que le code fait beaucoup d'arithmétique en virgule fixe (et aussi avec des entiers, bien sûr) donc attention si vous le reprogrammez dans un autre langage, les calculs doivent être exacts (voir section 1.1).

Si vous n'avez pas l'ambition de reprogrammer Opus, si vous voulez simplement une idée générale de son fonctionnement, la section 2 est faite pour vous. Opus, comme indiqué plus haut, a deux « couches », LP (prédiction linéaire) et MDCT (transformée en cosinus discrète). L'une ou l'autre (ou même les deux en même temps) est active pour encoder le signal audio, donnant ainsi à Opus la souplesse nécessaire. La couche LP est dérivée du codec Silk. Elle permet le débit variable mais aussi le constant (section 2.1.8). La couche MDCT, elle, est basée sur CELT.

Le codec Opus peut favoriser la qualité de l'encodage ou bien la consommation de ressources (processeur, capacité réseau). Cela permet d'ajuster la qualité du résultat aux ressources disponibles (section 2.1.5). Idem pour la sensibilité aux pertes de paquets : Opus peut être ajusté pour mettre plus ou moins de redondance dans les paquets. Avec le moins de redondance, on augmente la capacité mais on rend le décodage plus sensible : un paquet perdu entraîne une coupure du son.

Le format des paquets est décrit en section 3. À noter qu'Opus ne met pas de délimiteur dans ses paquets (pour gagner de la place). Il compte qu'une couche réseau plus basse (UDP, RTP, etc) le fera pour lui. Une possibilité (optionnelle dans les implémentations) de formatage avec délimiteur figure dans l'annexe B.

Chaque paquet compte au moins un octet, le TOC ("*Table Of Contents*"). Celui-ci contient notamment 5 bits qui indiquent les configurations (couches utilisées, capacité réseau utilisée et taille des trames).

Le fonctionnement du décodeur est en section 4 et celui de l'encodeur en section 5. Mais le programmeur doit aussi lire la section 6, qui traite d'un problème délicat, la conformité à la spécification. Je l'ai dit, la référence d'Opus est un programme, celui contenu dans ce RFC. Cela n'interdit pas de faire d'autres mises en œuvre, par exemple à des fins d'optimisation, à condition qu'elles soient compatibles **avec le décodeur** (c'est-à-dire que le décodeur de référence, celui publié dans l'annexe A du RFC, doit pouvoir lire le flux audio produit par cette mise en œuvre alternative). Pour aider aux tests, des exemples (`testvectorNN.bit`) et un outil de comparaison, `opus_compare`, sont fournis avec l'implémentation de référence (les tests sont en <`https://opus-codec.org/testvectors/`>). Un outil de comparaison est nécessaire car il ne suffit pas de comparer bit à bit la sortie de l'encodeur de référence et d'un nouvel encodeur : elles peuvent différer, mais dans des marges qui sont testées par `opus_compare`. Si vous avez écrit votre propre encodeur, ou modifié celui de référence, les instructions pour tester figurent en section 6.1.

Autre avertissement pour les programmeurs impétueux que les 150 pages d'explication et les 180 pages de code n'ont pas découragés : la sécurité. Opus est utilisé sur l'Internet où on trouve de tout, pas uniquement des gens gentils. La section 7 met donc en garde le programmeur trop optimiste et lui dit de faire attention :

— Si on veut chiffrer un flux Opus, il ne faut pas utiliser d'algorithme vulnérable aux attaques à texte clair connu. En effet, bien des parties du flux (comme les trames de silence ou comme les octets TOC) sont très prévisibles.
— Le décodeur doit être **très** robuste face à des erreurs (volontaires, ou bien provoquées par des bogues) dans le flux qu'il décode. Une programmation naïve peut mener à des accidents comme le dépassement de tampon ou, moins grave, à une consommation exponentielle de ressources (RFC 4732). L'implémentation de référence est censée être propre de ce point de vue, et résister à presque tout. Elle a été testée avec des flux audio aléatoirement modifiés, ainsi qu'avec des encodeurs faisant du "*fuzzing*". Elle a également été testée avec Valgrind.

Le compte-rendu des tests divers se trouve dans l'"*Internet-Draft*" `draft-ietf-codec-results`.

L'annexe A contient le code source de l'implémentation de référence encodée en Base64 (qu'on trouve également en ligne sur le site Web officiel <`https://opus-codec.org/`> et qui est distribuée par git en `git://git.xiph.org/opus.git`). Par défaut, elle dépend de calculs en virgule flottante mais on peut la compiler avec la macro `FIXED_POINT` pour ne faire que de la virgule fixe (voir le fichier `README`). Voici un exemple d'utilisation avec la version de développement (il n'y en avait d'autre prête à ce moment)

```
% export VERSION=1.0.1-rc2
% wget http://downloads.xiph.org/releases/opus/opus-$VERSION.tar.gz
...
% tar xzvf opus-$VERSION.tar.gz
...
% cd opus-$VERSION
% ./configure
...
% make
...
% make check
...
All 11 tests passed
...
```

Il n'existe pas à l'heure actuelle d'autre mises en œuvre d'Opus.

Combien de logiciels intègrent actuellement le code Opus pour lire et produire des flux audios ? Difficile à dire. L'excellent client SIP Csipsimple <`http://code.google.com/p/csipsimple/`> a

Opus depuis plusieurs mois. Autre exemple, le logiciel de voix sur IP Mumble l'a promis pour bientôt <http://mumble.sourceforge.net/Upcoming#Next_generation_audio_codec>.

Un bon article en français sur Opus, notamment ses performances : « Un codec pour les dominer tous <http://sebsauvage.net/rhaa/index.php?2012/09/12/08/30/48-un-codec-pour-les-dominer-to Et un article de synthèse de Numérama <http://www.numerama.com/magazine/23708-opus-est-desormai html> sur ce RFC. Pour une comparaison des performances d'Opus avec les autres, voir <https://www.opus-codec.org/comparison/>. Pour les nouveautés de la version 1.1 (publiée après le RFC), voir la liste officielle <http://people.xiph.org/~xiphmont/demo/opus/demo3.shtml>.

Merci à Régis Montoya pour sa relecture.