Codes AL-FEC et protocoles de diffusion robuste de contenus : composants pour des services performants et passant à l'échelle

Vincent Roca

HABILITATION À DIRIGER DES RECHERCHES

présentée par

Vincent Roca

pour obtenir le diplôme d'Habilitation à Diriger des Recherches
de l'Université de Grenoble, Spécialité Informatique

CODES AL-FEC ET PROTOCOLES DE DIFFUSION ROBUSTE DE CONTENUS

Composants pour des services performants et passant à l'échelle

Soutenue le (TBD) 2014
devant un jury composé de :

Ernst Biersack, PR Eurecom (rapporteur)
Bernard Cousin, PR Rennes-1 (rapporteur)
Marie-José Montpetit, main lecturer, MIT (rapporteur)
Walid Dabbous, DR Inria (examinateur)
Pierre Duhamel, DR CNRS (examinateur)
Jérôme Lacan, PR ISAE (examinateur)
Kavé Salamatian, PR Savoie (examinateur)
TBD (examinateur)

Published by: Ecole doctorale MSTII de l'Université de Grenoble, France

http://openfec.org - *Because open, free, AL-FEC codes and codecs matter*

http://planete-bcast.inrialpes.fr

Version 2.0, February 20, 2014

Contents

Remerciements

Ce document et le travail que j'ai mené durant ces années n'auraient pas été possibles sans l'héritage de tous ceux qui m'ont précédé. J'ai une pensée toute particulière à tous ceux qui ont oeuvré pour le partage libre des connaissances, des technologies et des logiciels. Ce document leur est dédié.

Ce document et le travail que j'ai mené n'auraient pas non plus été possibles sans les personnes qui m'ont guidé et conseillé à différents moment, à commencer par Christophe Diot et Michel Habert, mes encadrants durant ma thèse, à ceux qui m'ont fait confiance en me recrutant à Paris 6 puis à l'Inria, à Serge Fdida avec qui j'ai travaillé durant trois années dans l'équipe Réseaux et Performances, à Walid Dabbous et Claude Castelluccia pour mes 12 ans passés dans l'équipe Planète (où tous les travaux présentés dans ce document ont été effectués), mais aussi aux nombreux étudiants, ingénieurs, doctorants et post-doctorants que j'ai eu le plaisir d'encadrer, ainsi que les collègues avec qui j'ai eu la chance de travailler. Soyez tous remerciés du plus profond du coeur. Ce document vous est également dédié.

Enfin, tout au long de ces années, j'ai tenté d'apporter des contributions à la communauté, au sens large, aussi modestes soient elles, dans une approche constructive, pour le bénéfice de tous. Puisse la communauté continuer d'avancer dans cette voie sans céder à la tentation du verrouillage des idées et des technologies.

J'ai rédigé ce document pour trois raisons :

- bien entendu, afin de passer mon HDR. Je tiens au passage à remercier chaleureusement les membres du jury d'avoir accepté de juger mon travail et d'avoir lu et commenté ce document (un peu long, je le concède). A charge de revanche;

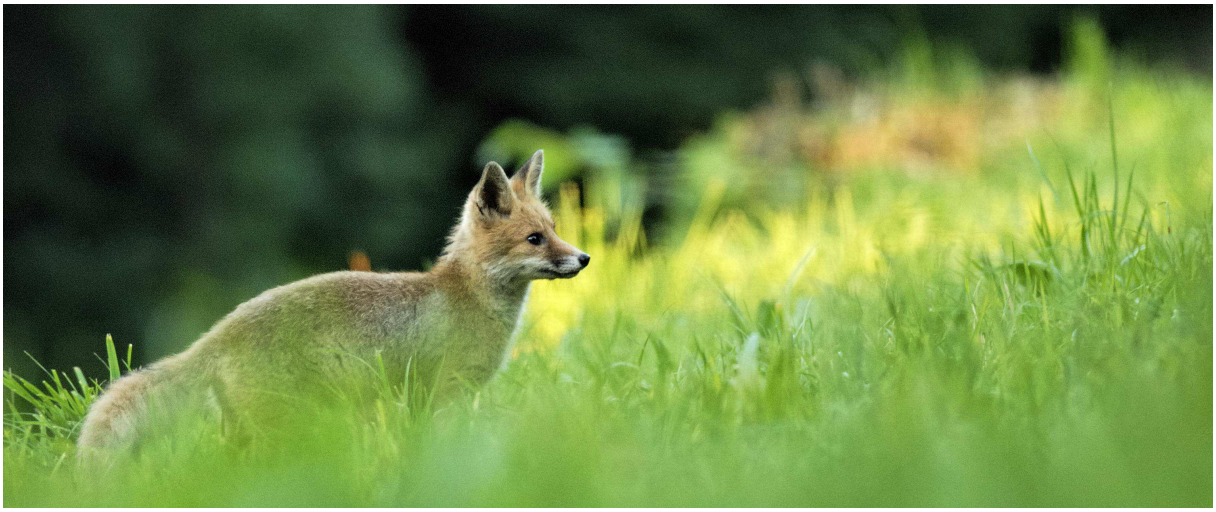
- pour moi même, puisqu'il s'agit d'une rare occasion de lever le nez du guidon afin de faire un bilan et de regarder un peu plus loin;

- mais aussi (surtout ?) afin de créer un document synthétique et compréhensible, qui permette de se faire une idée globale du domaine, et qui soit de ce fait utile à la communauté.

Enfin, j'ai tenté d'égayer ce document par quelques photographies que j'ai prises en milieu naturel, sur des animaux sauvages, sans exception. Merci à tous mes sujets photographiques pour s'être montrés un peu moins farouches le temps de ces quelques clichés. Et au passage merci à la marque noire et jaune pour ses excellents équipements photos ;-) : pour ma part un D200, généralement associé à un 300mm f/4 et son multiplicateur 1.4, sans oublier Capture NX2 pour le développement et le post-traitement.

Vincent

A Catherine, pour tes encouragements. . .



Jeune renard (3 mois), Isère, 2008.

1
Positionnement du travail

Résumé :
Ce chapitre introduit la problématique générale qui nous intéresse. Après un survol du domaine, nous précisons le point de vue qui sera suivi et les contributions qui seront détaillées dans ce document.

1.1 Objectifs: les services de diffusion robuste de contenus

Le problème général auquel nous nous attachons est celui de la fourniture de services de diffusion où des contenus de toute nature doivent être transmis efficacement et de façon robuste à un grand nombre de récepteurs en respectant les besoins applicatifs.

Nous voyons de suite qu'il existe plusieurs dimensions au problème :

- la notion de **robustesse** est essentielle. Le support de transmission, au sens large, va potentiellement générer des problèmes de corruptions (un bit à "1" sera reçu improprement et considéré comme valant "0") et/ou de pertes (une élément d'information est perdu avant d'arriver à l'application réceptrice). Il faut donc mettre en place des techniques pour recouvrir ces problèmes;

- la notion d'**efficacité**. Plusieurs critères d'efficacité sont possibles, parfois contradictoires. Par exemple fortes capacités de récupération sur erreurs et/ou pertes et faible complexité algorithmique sont souvent antagonistes. Il s'agira en général de trouver un compromis adéquat pour un champ d'application donné;

- les **contraintes temporelles** associées aux contenus. Ces contraintes, lorsqu'elles existent, impactent fortement la façon dont le service est fourni, par exemple en limitant fortement, voir en empêchant, toute retransmission de paquet perdu;

- le **nombre de récepteurs**. Certaines techniques de diffusion ne "passeront pas à l'échelle", le service s'effondrant lorsque ce nombre dépasse un certain seuil. A l'inverses d'autres techniques resteront efficaces même pour un nombre important de récepteurs;

Tout au long de ce document nous reviendrons longuement sur ces différents aspects.

1.2 *Services de diffusion et pile de communication TCP/IP*

Continuons en précisant ce que l'on entend par service de diffusion, puisque celui-ci recouvre plusieurs réalités différentes suivant le niveau où l'on se place dans la pile de communication Transmission Control Protocol/Internet Protocol (TCP/IP).

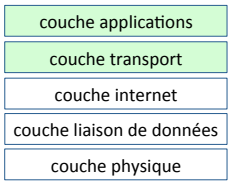


Figure 1.1: Modèle TCP/IP.

Services de diffusion de niveau 4- Transport / 5- Application :
Considérons la pile de communication TCP/IP en cinq couches de l'Internet Engineering Task Force (IETF) (figure 1.1)[1]. Les services de diffusion qui nous intéressent se situent au niveau Transport / Application, soit dans les couches "hautes" de la pile de communication. Ils reposent en général sur les services offerts par User Datagram Protocol (UDP), à savoir la possibilité d'envoyer des blocs de données (ou datagramme UDP) en préservant leurs frontières, et avec la fonction de multiplexage/démultiplexage apportée par le numéro de port UDP. Tous les autres services (par exemple la fiabilité des données transmises, le re-séquencement de blocs de données reçus dans le désordre, ou le fait de réagir face aux situations de congestion) seront pris en charge, de façon adéquate, par le service de diffusion lui même.[2]

Services de diffusion de niveau "3- internet" :
Ce service de diffusion de niveau Transport/Application repose en général sur un service de diffusion de niveau "3- internet". Ce service de niveau "3- internet" permet alors d'acheminer chaque paquet transmis par la source à plusieurs récepteurs simultanément, avec le cas échéant duplications du paquet à un point de branchement si des chemins différents doivent être suivis pour certains destinataires. En revanche la source n'aura jamais à transmettre plusieurs fois le même paquet.

Deux variantes de ce service de diffusion existent :

- **multicast :** ici une station doit se déclarer explicitement comme étant intéressée avant de pouvoir recevoir les paquets correspondants. Si aucune station ne s'est déclarée, les paquets transmis par la source ne seront pas acheminés au delà du réseau local de la source, le premier routeur évitant de les propager sur d'autres

[1] Certaines mauvaises langues objecteront qu'il y a 7 couches dans le modèle Open Systems Interconnection (OSI)... Nos travaux se plaçant dans le contexte IETF, nous suivrons la version pragmatique de modèle en cinq couches. A noter que l'un des documents fondateurs, le RFC 1122 [12], fusionne les couches "liaison" et "physique" pour ne considérer qu'une couche "lien", soit un modèle en 4 couches.

[2] Il est à noter que nous ne considérerons pas les services de transmission fiable de Transmission Control Protocol (TCP), ce protocole, tel qu'il est défini par l'IETF, n'étant pas compatibles avec la notion de diffusion autrement que par l'usage de multiples connexions indépendantes.
A noter cependant que des travaux académiques ont proposé différentes extensions à TCP afin d'étendre ce dernier à des scénarios multicast, notamment TCP-XM [36]. Cependant cela reste à ce jour des travaux académiques.

liens. La source n'est pas informée du nombre de récepteurs et le flux continuera à être transmis de la même façon quel que soit la situation[3]. Ce service de transmission "multicast" implique donc l'échange de messages de signalisation explicites entre chaque station intéressée et le routeur associé au réseau local. Dans sa forme la plus générale, dénotée Any Source Multicast (ASM), le trafic est de type $m \rightarrow n$, depuis m sources vers n destinataires. Dans un forme plus restreinte, dénotée Source Specific Multicast (SSM), le trafic est de type $1 \rightarrow n$, la source étant unique, ce qui facilite grandement le déploiement de ce service[4].

[3] Informer la source de la présence ou non de récepteur ne peut se faire qu'au travers d'un mécanisme supplémentaire. Nous avions proposé en 2000 le protocole "On Demand Layer Addition" (ODL) pour cela [72]. L'idée a été développée au sein de l'IETF au travers du service "Multicast Source Notification of Interest Protocol" (MSNIP), sans que cela aboutisse à un Requests For Comments (RFC) [26].

[4] Par exemple il n'est plus nécessaire d'avoir un protocole de gestion des adresses multicast au niveau Internet, chaque couple {adresse IP source; adresse IP multicast} désignant un flux différent, sans risque de collision.

- **broadcast :** ici le trafic est automatiquement transmis à l'ensemble des stations et équipements attachés au réseau local, sans qu'il soit nécessaire pour chaque station de se signaler préalablement. Le réseau local en question n'est pas nécessairement celui de la source : il correspond au réseau désigné par la partie "réseau" de l'adresse IP.

Ces distinctions se reflètent au niveau des adresses puisque ces services sont associées respectivement à des adresses IPv4/IPv6 de type "multicast" et "broadcast" ([17] et [33] décrivent les règles générales d'allocation d'adresses multicast IPv4 et IPv6). Il est important de noter qu'à la différence d'une adresse classique, une adresse "multicast" n'identifie aucune station particulière. Elle ne fait qu'identifier (associé à l'adresse IP source dans le cas SSM) l'entité abstraite de "groupe de diffusion multicast" à destination de laquelle se fait la transmission.

Services de diffusion au niveau des "couches basses" :
A son tour, le service de diffusion au niveau "3- internet" repose sur un service similaire au sein des "couches basses", à savoir les couches "2- liaison de données" et "1- physique". Des services de multicast et broadcast sont également définis dans les couches basses. Ceci peut être plus ou moins aisé :

- Dans le cas de **technologies de transmission sans fil** (par ex. de type Wifi ou satellite), le signal est naturellement diffusé vers l'ensemble des récepteurs présents dans la zone de couverture. Mettre en place au sein d'une zone de couverture un service de diffusion est alors largement facilité par les propriétés physiques de transmission du signal;

- Dans le case de **technologies de transmission filaire commutée** (par ex. un Local Area Network (LAN) Ethernet moderne), le signal est naturellement localisé à un sous ensemble du réseau afin de permettre des communications en parallèle[5]. Par ce que c'est essentiel à de

[5] Ainsi un commutateur Ethernet (ou "switch") doté de huit ports physiques permet à 4 couples de machines distinctes de communiquer en bénéficiant de la totalité de la bande passante, comme si chaque couple de machines était directement connecté. Ce n'est plus vrai si plusieurs machines s'adressent à la même machine (par exemple un serveur Network File System (NFS)).

nombreux protocoles et services, un mode broadcast est ajouté qui permet à une trame de type multicast ou broadcast d'être transmise à l'ensemble des noeuds attachés au réseau physique[6].

[6] Ainsi au lieu de restreindre la transmission d'une trame au port de sortie permettant d'atteindre sa destination, un commutateur Ethernet dupliquera les trames ayant une adresse destination de type broadcast vers l'ensemble des ports de sortie. Avec un simple "hub" Ethernet, il n'y a pas de séparation de trafic et toute trame est acheminée sur tous les ports de sortie

Disponibilité des services de diffusion :
Nous voyons qu'existe une large variété de situations, aux contraintes techniques variées, ce qui impacte largement l'accès à un service opérationnel de diffusion. Ainsi :

- il est aisé et efficace de fournir un service de diffusion local, en particulier dans le cas d'un support de transmission sans fil;

- il est bien moins simple de fournir un service équivalent au niveau d'Internet, puisque cela nécessite :

  – d'utiliser des protocoles permettant de créer et maintenir un arbre de diffusion permettant d'atteindre tous les récepteurs intéressés par un flux (notion de "routage multicast");

  – de faire face à la dynamique du service, les récepteurs arrivant et partant à tout moment, sachant que certains protocoles de contrôle de congestions nécessitent une réaction rapide qu'il est difficile de fournir [7];

  – l'usage de services de diffusion et TCP étant incompatibles, l'usage d'UDP est de facto quasi inévitable, ce qui signifie qu'il n'y aura pas au sein du protocole de transport, de mécanisme de contrôle de congestion. Les flux diffusés ne seront donc peut être pas réactifs face aux indications de congestion, ce qui constitue un frein majeur à leur déploiement.

[7] Voir à ce sujet la discussion sur WEBRC, section 10.2.1.

Ceci fait partie des données du problème et explique que ces services ne soient pas disponibles de base sur Internet. Ceci explique également que des solutions alternatives de diffusion, efficaces, telles que BitTorrent, se soient développées.

Cependant, les services de diffusion IP Television (IPTV) d'un Fournisseur d'Accès Internet (FAI) reposent de facto sur un service de routage multicast sous-jacent [61]. De même, les services de diffusion de contenus multimédia sur les réseaux 3G/4G, Multimedia Broadcast and Multicast Services (MBMS) (sur lesquels nous reviendrons) reposent largement sur ces technologies. Même si ce sont des cas particuliers, même si le service de routage multicast n'est pas déployé à large échelle sur Internet, il n'en reste pas moins que *les services de diffusion de contenus ont largement pénétré notre vie quotidienne*.

1.3 *Point de vue adopté*

Le domaine est très vaste, aussi nous commençons par préciser les sujets qui seront abordés ou non dans ce document.

1.3.1 *Ce document et ce travail ne s'intéressent pas...*

Ce document et ce travail **ne s'intéressent pas** à la fourniture du service de diffusion sur Internet pour les couches "3- internet" et "couches basses". Nous ne parlerons pas de routage multicast ASM ou SSM, ni de connexion d'îlots multicast distants au travers de tunnels UDP, par exemple via l'approche AMT [13] (en général lorsque le service de routage multicast n'est pas disponible de bout à bout).

Nous ne parlerons pas non plus des approches de type pair-à-pairs qui permettent également de répliquer des contenus numériques, parfois très efficacement comme on le permet des approches type BitTorrent. C'est un domaine de travail à part entière, avec des solutions très efficaces et utiles. En revanche ces solutions ont en général la particularité de reposer sur de multiples connexions point-à-point, ce qui constitue un élément différenciateur.

Nous ne détaillerons pas non plus les approches de streaming de contenus multimédia en point à point, que ce soit la solution classique HTML Progressive Download, ou du Dynamic Adaptive Streaming over HTTP (DASH), plus performante dans un contexte d'accès ubiquitaire aux contenus multimédia [95].

Nous ne parlerons pas plus de "network coding" lorsque cela consiste en l'ajout de fonctionnalités de ré-encodage au sein même du réseau, bien que ce soit une approche prometteuse [62]. Par contre les codes utilisés habituellement dans ces solutions, RLC, consistant en des combinaisons linéaires avec coefficients aléatoires sur un corps fini donné, sont en lien avec notre travail. Nous reviendrons dessus dans la section 12 de conclusions et perspectives. De même, une approche bout à bout des solutions de "network coding" telle que Tetrys [101], qui ne nécessite aucun ajout de fonctionnalité dans le coeur de réseau, est en lien avec notre travail.

Nous ne parlerons pas, lorsque l'on abordera les codes correcteurs, des canaux à erreurs. Ceci est un conséquence directe du point de vue adopté, puisque dans les couches 4- Transport/5- Application, les erreurs ont soit été corrigées dans les couches basses, soit ont conduit à la destruction du paquet[8].

Parmi les codes correcteurs pour canaux à effacements, nous laisserons de coté les champs d'application de type stockage distribué, même si le canal est également de type effacement (un noeud de stockage devenu indisponible peut être vu comme un effacement de sym-

[8] UDP-Lite [42] est une exception notable, puisque cette version d'UDP permet de remonter à l'application des données pouvant être partiellement endommagées et pour lesquels le récepteur ne fera pas de vérification de checksum sur l'ensemble des données du datagramme. Couplé à une couche basse tolérante aux erreurs, cette approche est intéressant pour tous les flux (par exemple avec certains codecs audio/vidéo) où ne pas recevoir des données est pire que recevoir des données partiellement endommagées.

boles, le symbole étant ici un tronçon de fichier) [70]. En effet, ces codes seront largement différents de ceux auxquels nous nous intéressons, du fait de contraintes très spécifiques[9].

[9] Ainsi le nombre de "symboles" (donc le nombre de noeuds de stockage) est en général relativement faible. Il sera également souvent nécessaire de minimiser le nombre de transferts de données, par exemple dans une opération de ré-encodage suite à la disparition de certains noeuds de stockage, car les latences de communication réseaux sont élevées.

1.3.2 *Par contre, ce document et ce travail s'intéressent...*

Ce document et ce travail s'intéressent au problème de la transmission robuste d'informations de tout type, avec ou sans contraintes temps réel, à un nombre élevé de récepteurs. Le point de vue adopté est double :

- nous nous intéresserons aux composants essentiels que sont les **codes correcteurs de niveau applicatif**, ou AL-FEC, sans lesquels l'efficacité du service serait fortement compromise. Nous nous limiterons au cas du canal à effacement de paquets;
- nous nous intéresserons aux **protocoles de transport** qui définissent les règles d'échanges d'informations entre source(s) et récepteur(s) et garantissent le niveau de fiabilité nécessaire;

Puisque le passage à l'échelle de la solution de diffusion en termes de nombre de récepteurs est un aspect clef, je parlerai de **"services de diffusion à large échelle"**.

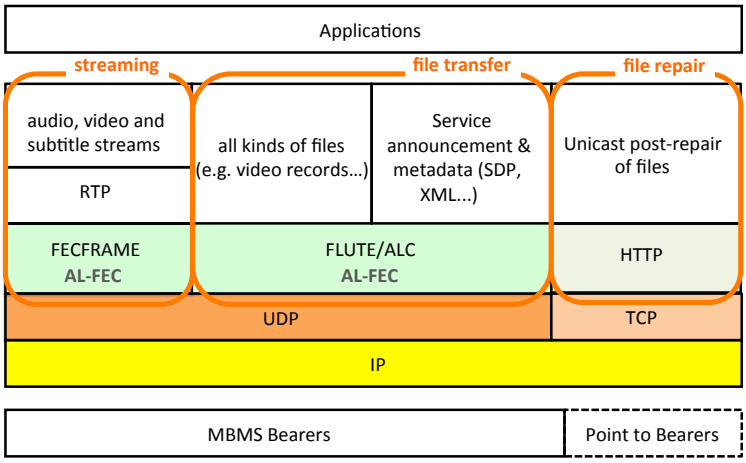


Figure 1.2: Pile protocolaire des services 3GPP/MBMS.

Il est agréable de constater que plusieurs des codes et protocoles sur lesquels nous avons travaillé ont depuis le milieu des années 2000 **connu un certain succès**. Ainsi les protocoles de transport File Delivery over Unidirectional Transport (FLUTE)/Asynchronous Layered Coding (ALC) ont été largement déployés via les services de diffusion fiable à destination de terminaux mobiles (typiquement smartphone), via le service 3rd Generation Partnership Project (3GPP) MBMS (ou ses

déclinaisons) (figure 1.2). De même, nos codes correcteurs Low Density Parity Check-Staircase (LDPC-Staircase) sont depuis avril 2012 déployés commercialement au Japon, via le standard Integrated Services Digital Broadcasting - Terrestrial mobile multi-media (ISDB-Tmm).

1.4 *Organisation du document*

Ce document s'articule autour des deux axes mentionnés ci-dessus.

Nous commençons par les codes AL-FEC. Nous introduisons les notions de *codes pour le canal à effacements*, leurs intérêts et en quoi ils se distinguent des codes FEC de la couche physique (chapitre 2). Nous définissons alors ce que nous entendons par *bon code AL-FEC*, en distinguant les critères liés à l'usage de ces codes, d'ordre qualitatifs, de ceux liés aux performances brutes, d'ordre quantitatifs (chapitre 3). Nous continuons alors (chapitre 4) avec les *codes LDPC-Staircase* sur lesquels nous avons énormément travaillé. Malgré la simplicité de leur structure, nous montrons que ces codes offrent des performances de correction très élevées en condition réelle, et pas seulement de façon asymptotique (chapitre 5). Nous expliquons ensuite comment réaliser un codec logiciel hautes performances et détaillons les résultats obtenus, que ce soit dans un environnement type poste de travail, ou type smartphone. Nous en profitons pour comparer les vitesses obtenues avec celles des codecs Raptor et RaptorQ, suite à la compétition du 3GPP-MBMS (chapitre 6). Nous montrons que pour des usages classiques la combinaison LDPC-Staircase plus Reed-Solomon (Reed-Solomon) (pour les petites dimensions de codes) a des niveaux de performances de premier plan : elle a été jugée nettement plus intéressante que les codes Raptor et largement au niveau des codes RaptorQ lors de la compétition du 3GPP-MBMS). Afin d'aller plus loin, nous introduisons enfin les *codes Generalized LDPC-Staircase (GLDPC-Staircase)*, qui constituent une fusion des codes LDPC-Staircase et Reed-Solomon (chapitre 7).

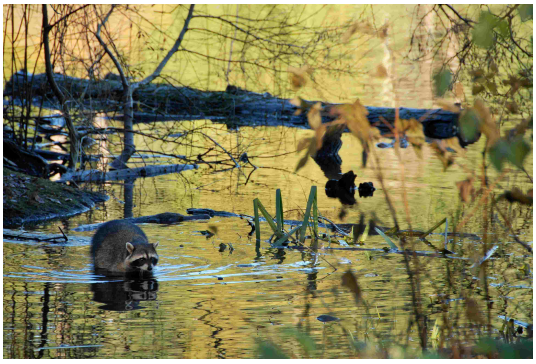
Ce document continue avec un aspect souvent négligé en théorie de l'information, *la signalisation*. Nous l'abordons au travers de la notion de "schémas FEC" qui au delà de la définition précise des codes, détaille la façon dont ils sont utilisés et les informations devant être échangées entre encodeur et décodeur à des fins de synchronisation (chapitre 8). Nous montrons alors comment construire un service AL-FEC évolué offrant protection inégale et protection groupée d'un ensemble d'objets (par exemple de petite taille) avec l'approche GOE, uniquement en jouant sur des techniques de signalisation (chapitre 9).

Puis nous passons au deuxième axe, à savoir les protocoles de transport. Nous détaillons l'approche FLUTE/ALC qui permet une diffusion fiable de contenus statiques, sans limitation de passage à l'échelle,

et à notre alternative Scalable Object Delivery over ALC and NORM Protocols (FCAST) qui corrige certains défauts de FLUTE (chapitre 10). Nous continuons avec l'approche FECFRAME adaptée à la transmission robuste de contenus arrivant en continu coté émission (chapitre 11). Dans ces deux cas, la brique AL-FEC joue un rôle prépondérant, puisque c'est elle qui apporte fiabilité et efficacité.

Enfin nous concluons ce document, en mettant en évidence les principaux enseignements que l'on peut en retirer, et nous ouvrons sur des perspectives (chapitre 12).

Deux annexes offrent des éclairages supplémentaires, d'une part sur les performances des codes LDPC-Staircase via l'étude de leur probabilité d'échec de décodage, et d'autre part sur l'environnement d'évaluation de performances que nous avons élaboré distribuons via le projet OpenFEC : sans ces outils de nombreux résultats présentés dans ce document n'auraient pu être obtenus.



La traversée du raton laveur, Canada, 2007.

Part I

Codes AL-FEC pour le canal à effacements

2
Introduction aux codes AL-FEC

Crédits : *Cette introduction a largement bénéficié de discussions avec des collègues, tout au long de ces années, en particulier dans le cadre du projet ANR/CAPRIFEC. Merci à vous.*

Résumé :
Ce chapitre constitue une introduction aux codes FEC en général, et au cas particulier des codes AL-FEC en particulier. Nous montrons quels en sont les spécificités et bénéfices, faisons le lien entre ces codes AL-FEC et ceux de la couche PHYsical layer (PHY), et finissons avec une introduction aux notions de base en matière d'espaces vectoriels, matrices et graphes.

2.1 *Erreurs ou bien effacements ?*

Considérons un système de transmission au sein d'un "canal de communication", malheureusement imparfait, qui peut introduire toutes sortes d'erreurs, voir perdre purement et simplement des informations. La nature des imperfections pouvant être corrigées est essentielle.

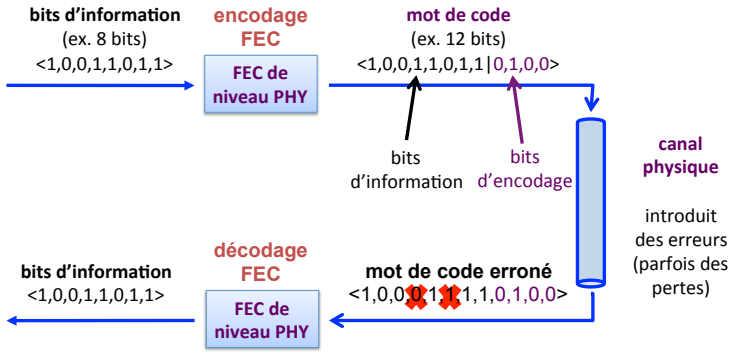


Figure 2.1: Codes FEC de la couche PHY.

Commençons par dire quelques mots des **canaux à erreurs**. Sur le canal à erreurs, des informations, en général des bits, peuvent changer de valeur durant la transmission. La Fig. 2.1 représente une transmission avec encodage FEC sur un tel canal. Un bloc comprenant 8 bits

est donné à l'encodeur FEC, qui ajoute 4 bits de redondance. Le tout forme ce que l'on appelle un "mot de code" (ou "codeword"). Durant la transmission deux bits sont modifiés (deux erreurs). Le nombre d'erreurs étant faible par rapport à la redondance ajoutée, le décodeur FEC peut corriger ces erreurs et reconstituer le bloc d'origine [1].

Les codes FEC apportent dès lors deux niveaux de protection :

- **détecter** des erreurs dans les données reçues. Il s'agit du premier niveau de protection [2];
- **corriger** les erreurs et récupérer les données d'origine. Il s'agit du deuxième niveau de protection;

A l'inverse, sur un **canal à effacements**, un effacement est aisément identifié en tant que tel. Le niveau de protection apporté par les codes FEC sera donc uniquement de **récupérer l'information manquante**. Nous nous intéresserons uniquement à ce type de canal dans la suite de ce document.

[1] Si l'on regarde plus en détail, différents modèles de canaux à erreurs existent tels que les canaux Binary Symmetric Channel (BSC) et Additive White Gaussian Noise (AWGN). Nous ne les détaillerons pas ici.

[2] Par exemple la fonction de vérification d'intégrité peut être assurée par un Cyclic Redundancy Check (CRC) [3]. Ainsi avec un CRC16, où la somme de contrôle est de taille 16 bits, le récepteur pourra vérifier l'intégrité d'un contenu avec une probabilité 1/65536 de ne pas identifier un contenu erroné.

2.2 *Le canal à effacements*

2.2.1 *Définitions*

Le canal à effacements est défini ainsi :

L'information transmise sur un canal à effacements peut être soit totalement perdue, soit parfaitement reçue, sans erreur.

Le canal à effacement le plus élémentaire est nommé **Binary Erasure Channel (BEC), ou canal à effacement de bits** (Fig. 2.2). Dans ce canal, un bit donné peut être soit totalement perdu durant la transmission (le récepteur n'a aucune information sur sa valeur réelle), soit reçu sans erreur. Le principe est le même dans le cas d'un canal à effacements de "paquets", ces derniers remplaçant les bits. Nous verrons (section 2.3.4) que travailler sur des bits n'est pas réducteur et qu'il existe un lien direct entre les symboles manipulés par le codec et les bits du code sous-jacent. Nous admettons donc pour le moment que l'unité de données pourra être le bit (cas du canal BEC) ou un ensemble plus ou moins important d'octets, tel un datagramme IP, et appelons cette unité de données *"symbole"*.

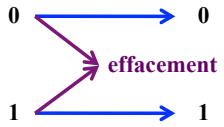


Figure 2.2: Principes du canal à effacement au niveau binaire (ou BEC).

2.2.2 *Exemples de canaux à effacements*

Le canal à effacements n'est pas qu'un canal théorique. Il se retrouve dans de nombreux systèmes :

- Internet peut être vu comme un canal à effacements, puisqu'un datagramme IP est soit reçu, soit perdu, sachant que les checksum de niveau PHY (par exemple le Frame Check Sequence (FCS) d'Ethernet) ou transport (par exemple les checksums Internet Protocol (IP) (sur l'entête), TCP ou UDP (sur la zone de données)) permettent de garantir l'intégrité des données reçues au niveau de la couche transport.

- un récepteur connecté à Internet de manière intermittente verra un canal à effacements. Un exemple est celui d'un mobile disposant d'une qualité de réception 3rd Generation (3G) variable. Ce mobile ne sera peut être pas connecté en permanence et de nombreux paquets transmis seront perdus dans les phases de déconnexion.

- considérons un service de téléchargement de contenus, en mode diffusion, dans lequel un serveur émet un flux continu de données, à destination d'un groupe multicast. Chaque récepteur potentiel peut s'abonner à sa guise à ce groupe et recevoir le contenu. Si l'application cliente s'arrête (suite à une suspension du PC ou à un crash), de nombreux paquets sont alors perdus durant la période de déconnexion. Du point de vue du récepteur, nous avons là encore un canal à effacements.

- une baie de disques et un système de stockage distribué peut être vue comme un canal à effacement, puisque perdre un ou plusieurs disque reviendra à un ou plusieurs effacements. Avoir une machine d'un service de stockage distribué indisponible constitue également à un effacement.

Dans ce document nous nous focaliserons sur les deux premiers champs d'application.

2.3 *Les codes AL-FEC*

2.3.1 *Intérêt des codes AL-FEC*

Au sein de la grande famille des codes correcteurs, ou codes FEC, nous nous intéressons aux codes AL-FEC qui sont intimement liés au canal à effacements.

Les codes AL-FEC travaillent systématiquement sur un canal à effacements.

Nous retrouverons ces codes :

- dans la couche **"Application"** (d'où leur nom);

- dans la couche **"Transport"**, par exemple au sein du protocole de transport ALC ((chapitre 10) ou de l'architecture FECFRAME (chapitre 11)[3];
- dans la couche **"Medium Access Control (MAC)"**, par exemple dans le contexte du Digital Video Broadcasting - Handheld (DVB-H) (resp. Digital Video Broadcasting - Satellite Handheld (DVB-SH)) au sein du système Multiprotocol Encapsulation - FEC (MPE-FEC) (resp. Multiprotocol Encapsulation – Inter-burst FEC (MPE-IFEC))[4].

[3] La frontière entre "Application" et "Transport" reste en pratique assez floue. Par exemple ALC et FECFRAME opèrent tous deux au dessus du protocole de transport UDP !

[4] Notons que dans ce cas les codes AL-FEC sont aussi appelés Upper-Layer Forward Erasure Correction (UL-FEC) pour les distinguer des usages au sein de la couche PHY qui seront eu qualifiés de codes "Lower Layer FEC".

Ces codes ont deux rôles. Ils permettent :

- **d'améliorer la fiabilité des transmissions :** ajouter de la redondance au flux de données permettra de faire face à des pertes, sans avoir à recourir à des retransmissions ce qui est un avantage majeur en termes de latence. De plus ceci se fait simplement, et plusieurs récepteurs observant des pertes différentes pourront bénéficier des mêmes symboles de redondance. Il s'agit là d'un avantage majeur par rapport à un système basé sur des retransmissions qui nécessairement répondent à des requêtes distinctes. Dans le cas où le canal est de qualité variable, il suffira pour s'adapter de moduler dynamiquement le volume de données redondantes;
- **d'optimiser le service de diffusion lui même :** ajouter de la redondance permet de fournir un service de diffusion de type un carrousel efficace, quel que soit le comportement d'un récepteur. Un récepteur disposant d'une connectivité intermittente sera juste contraint d'attendre un plus longtemps jusqu'à recevoir le nombre nécessaire de symboles (au moins k) par rapport à un bon récepteur.

Ces avantages sont essentiels et montrent la simplicité, l'élégance et l'efficacité de la solution, en particulier dans le cas de scénarios de diffusion (multicast et broadcast) d'informations. Continuons avec un exemple montrant ces codes en action lorsqu'ils sont utilisés au niveau "Application"/"Transport".

2.3.2 *Exemple : protection d'un objet lors d'une transmission FLUTE/ALC*

Considérons un objet transmis vers plusieurs destinataires au moyen du protocole de transmission "multicast fiable" FLUTE/ALC (que nous détaillerons au chapitre 10). Chaque paquet FLUTE/ALC est transmis au sein d'un datagramme UDP/IP sur Internet. De ce fait, certains d'entre eux pourront être perdus durant la transmission. La fiabilité est alors obtenue grace à l'usage de codes AL-FEC (Fig. 2.3). Le principe est le suivant :

- **Etape 1 :** Nous commençons par découper l'objet à transmettre en unités de données de taille fixe[5], appelées **symboles sources**. Dans

[5] Tous les symboles d'un objet ont la même taille, sauf éventuellement le dernier si la taille de l'objet n'est pas multiple de la taille des symboles. Un bourrage est alors appliqué à ce dernier symbole lors des opérations d'encodage/décodage FEC. En revanche ce bourrage n'est jamais transmis.

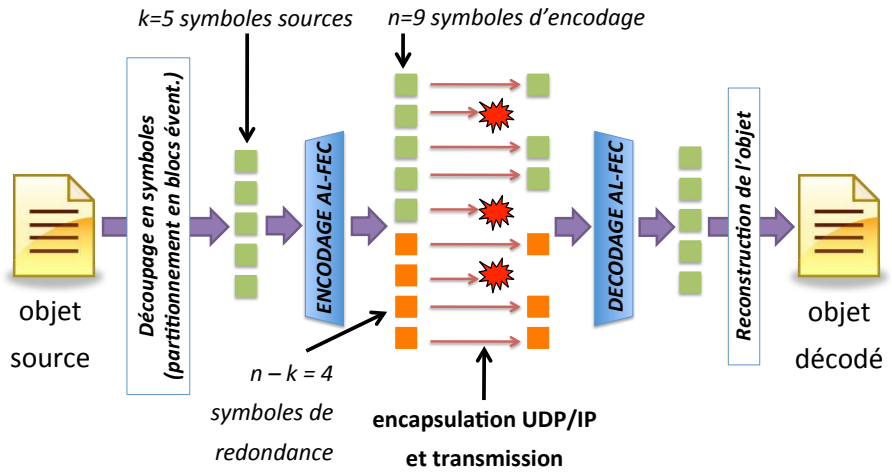


Figure 2.3: Principes de fonctionnement d'un code AL-FEC lors d'une transmission de datagramme UDP/IP sur Internet. Cas simple où le nombre de symboles sources est suffisamment faible pour ne considérer qu'un seul bloc.

cet exemple nous avons $k = 5$ symboles sources. Il est à noter que généralement la taille des symboles correspondra au Maximum Transmission Unit (MTU) minimum sur le chemin diminué des tailles des différentes entêtes (ALC, UDP, et IP dans ce cas).

- **Etape 2 :** Plaçons nous dans le cas simple où ce nombre de symboles sources est compatible le nombre maximum pris en charge par le codec AL-FEC. L'encodage AL-FEC a alors lieu sur cet unique bloc de k symboles (si ce n'est pas le cas, l'objet est partitionné en plusieurs blocs selon un algorithme précis[6], l'encodage se faisant indépendamment sur chaque bloc). A ces symboles source, l'encodeur AL-FEC ajoute $n - k = 4$ **symboles de redondance**, pour un total de $n = 9$ **symboles d'encodage** en sortie d'encodeur.

  Nous parlons ici d'un code AL-FEC (n, k), où k est la **dimension** du code et n sa **longueur**, notions qui font référence à l'espace vectoriel associé (voir la section 2.4).

- **Etape 3 :** Chaque symbole est alors encapsulé dans un paquet (en fait un datagramme ALC/UDP/IP)[7] puis transmis sur Internet.

- **Etape 4 :** Considérons l'un quelconque des destinataires de la transmission multicast. Ce récepteur reçoit un sous ensemble des paquets transmis (et donc des n symboles), trois d'entre eux ayant été perdus par le "canal à effacement" qu'est Internet. Si l'on fait l'hypothèse que le code est idéal, dès lors que le récepteur a reçu au moins $k = 5$ symboles quelconques parmi les $n = 9$ possibles, il est en mesure d'effectuer le décodage AL-FEC et donc de reconstruire les deux symboles sources manquants[8].

- **Etape 5 :** Le décodage étant réussi, l'objet est reconstruit. Dans le cas où plusieurs blocs ont été nécessaires, cette opération de dé-

[6] Voir la section 8.4 pour la description de l'algorithme de partitionnement d'ALC et la section 3.2.1 pour la discussion sur les codes "petits" et "grands" blocs.

[7] Il y a souvent bijection entre symbole et paquet, un paquet contenant un unique symbole. Cependant dans certaines situations, en particulier avec de petits objets, certains codes pourront choisir de stocker plusieurs symboles dans un unique paquet pour des raisons de performances.

[8] On notera qu'il n'est pas nécessaire de reconstruire le symbole de redondance manquant, à moins que ce ne soit une étape intermédiaire indispensable au décodage.

codage se fait sur chaque bloc, indépendamment. Lorsque tous les blocs ont été décodés, l'objet est alors reconstruit. Nous avons donc masqué les pertes de transmission sans avoir à effectuer la moindre retransmission.

Un autre récepteur, recevant un sous ensemble distinct de symboles car situé ailleurs sur Internet, pourra lui aussi reconstuire l'objet dès lors qu'il a reçu au moins k symboles parmi n. Comme l'identité des symboles reçus n'intervient pas dans le décodage (du moins avec des codes ideaux), nous voyons qu'un unique flux permettra à différents récepteurs de bénéficier d'une transmission fiable.

2.3.3 *Un codec logiciel pour une souplesse inégalée*

Du fait de leur position "haute" dans les couches protocolaires, les codes AL-FEC sont implantés au sein d'un **codec logiciel**, ce qui apporte une très large flexibilité à la solution. Ainsi :

- **les paramètres sont définis dynamiquement et le code AL-FEC est construit à la volée**, en fonction des besoins précis. Les techniques de poinçonnage et raccourcissement de la couche PHY deviennent inutiles dans le cas général. Le code correspond à chaque fois idéalement aux besoins;
- il devient possible de **travailler sur de très gros objets** car la mémoire utilisée est celle de l'hôte, en général relativement abondante, et non la mémoire du chipset, nécessairement limitée pour des raisons de coûts. Cet aspect peut être un critère déterminant dans certains champs d'application;
- il devient possible de **prendre en compte les spécificités de l'application**, par exemple pour offrir une protection différenciée à une partie du flux ou de l'objet.

Cette flexibilité est un avantage majeurs par rapport aux codes de la couche PHY, qui seront eux implantés au sein de modules matériels.

En revanche produire à la volée le code AL-FEC limite les éventuelles optimisations pouvant être apportées. En effet, le temps de création du code est en général inclus dans les temps d'initialisation des encodeur et décodeur, temps qu'il est indispensable de minimiser pour des performances en vitesse élevées[9].

Il existe deux types d'approches pour pallier à cette lacune :

- pré-calculer tous les codes possibles, correspondant à toutes les combinaisons de paramètres possibles, et sélectionner les plus pertinentes (cette étape pourra nécessiter plusieurs semaines de calculs). Il suffit alors de mémoriser les paramètres associés aux codes

[9] Ainsi retirer certains cycles de la matrice de parité d'un code Low Density Parity Check (LDPC) nécessite déjà de les identifier, ce qui reste une opération fort coûteuse de recherche au sein de cette matrice. Cette phase est en général incompatible avec une génération à la volée et rapide de ces codes.

les plus intéressants pour ensuite générer aisément le bon code en toutes circonstances[10];

- définir une technique de construction globalement satisfaisante, permettant de bonnes performances statistiques. C'est l'approche suivie par les codes LDPC-Staircase [81]. Si ces codes n'ont pas fait l'objet d'une présélection précise parmi un ensemble de candidats, nous montrons en revanche que leurs performances sont de tout premier plan du fait de leur structure efficace (chapitre 5);

[10] C'est l'approche suivie par les codes Raptor^{tm} et RaptorQ^{tm}, qui ont fait l'objet d'optimisations poussées. On trouve alors dans les spécifications [48][49] une table qui fournit une graine permettant, via un Pseudo-Random Number Generator (PRNG) précis, de recréer le code le mieux adapté (c'est à dire celui ayant montré les meilleurs résultats lors de la phase d'études préalables).

2.3.4 *Et les symboles dans tout cela ?*

Revenons sur la notion de symbole, déjà mentionnée dans l'exemple de la section 2.3.2. Nous rapellons que :

Les symboles sont les unités de données que l'on fournit ou récupère d'un codec AL-FEC via l'Application Programming Interface (API).

Il s'agit donc des unités de données en entrée/sortie du codec. Dans certains cas, les éléments sur lesquels fonctionne le code AL-FEC en interne pourront être différents, même si il y a souvent équivalence[11].

Plus précisément, nous distinguons, en nous appuyant sur la terminologie [105] :

- les *symboles sources* (ou "source symbols") sont des unités de données de l'objet d'origine, que l'on soit coté encodeur ou décodeur;
- les *symboles de redondance* (ou "repair symbols") sont des unités de données créées par l'encodeur, qui contiennent le résultat de l'encodage des symboles source. Lorsque l'on travaille sur des codes binaires, on parlera aussi de **symboles de parité**;
- les *symboles d'encodage* (ou "encoding symbols") sont les symboles résultant de l'encodage AL-FEC. Ces symboles, dans le cas d'un code systématique, incluent à la fois les symboles source et les symboles de redondance[12].

[11] Les codes Reed-Solomon sur $GF(2^8)$ sont un bon exemple de codes où il y a un différence entre les deux notions : en interne le code manipule des octets, éléments du corps fini $GF(2^8)$. En revanche, l'application ou le protocole faisant usage du codec via son API fournit des symboles faisant souvent plusieurs centaines d'octets au codec.

[12] Il s'agit là de la définition même d'un code systématique. En pratique les codes systématiques seront préférés, puisque dans le cas idéal où aucun symbole n'est perdu, aucun décodage n'est nécessaire. Un autre avantage d'un code systématique est la possibilité de transmettre immédiatement les paquets contenant les symboles source, sans avoir à attendre un d'effectuer un encodage AL-FEC. Ceci est essentiel lorsque l'on manipule des flux ayant des contraintes temps réel afin de minimiser la latence d'encodage.

La nature des données constituant les symboles sources varie énormément en fonction du champ d'application visé. Lorsqu'un code AL-FEC est utilisé avec FLUTE/ALC, un symbole source sera souvent constitué de quelques centaines d'octets issus d'un fichier. Ce symbole sera ensuite transmis dans la partie données utiles ("payload") d'un datagramme ALC/UDP/IP (section 2.3.2).

Dans le cas d'un usage au sein de FECFRAME (section 11.2), avec des applications de streaming temps réel, il s'agira souvent d'un paquet Real-Time Protocol (RTP), entête compris[13]. Ici la taille s'exprimera

[13] Nous verrons que le composant FECFRAME est souvent positionné entre RTP et UDP. Le symbole, constitué essentiellement d'un paquet RTP est alors transmis dans la partie données utiles d'un datagramme UDP/IP

en dizaines d'octets (cas d'un flux audio) ou en centaines d'octets (cas d'un flux vidéo) suivant la nature du média.

En revanche, dans le cas d'une application des codes Reed-Solomon au sein de la couche MAC `MPE-FEC` du DVB-H, le symbole sera un octet de la table `MPE-FEC`, elle même remplie avec les datagrammes IP successifs (après encapsulation et entrelacement) [89].

Enfin le symbole peut également être constitué d'un bit. C'est le cas avec le canal (théorique) BEC.

Liens entre symboles et mots de code :

Ainsi qu'on l'a vu en introduction (section 2.1), il est courant en théorie de l'information de parler de "mot de code" (ou "codeword") de longueur n bits : il s'agit de l'ensemble des bits d'information et de redondance à la sortie de l'encodeur (Fig. 2.1). C'est aussi un "mot du code FEC" car il fait partie de l'espace d'arrivée de l'espace vectoriel associé au code.

Dans le cas d'un usage au niveau de la couche PHY d'un canal de transmission, la notion de mot de code est clairement établie. En revanche, dans le cas d'un usage plus haut dans les couches de communication, par exemple au sein du protocole ALC/Layered Coding Transport (LCT), les symboles manipulés ne sont pas des bits mais des unités de données de plusieurs centaines d'octets la plupart du temps. Pourtant nous pouvons faire le lien entre la notion de "mot de code" et "symboles" comme nous allons maintenant le voir.

Le cas des codes binaires :

Avec des codes binaires (tels les codes LDPC-Staircase), le lien entre mots de code et symboles est le suivant (Fig. 2.4). Considérons des symboles de taille 1000 octets, soit 8000 bits. Dès lors, l'ensemble des i^{mes} bits de position i dans les n symboles d'encodage constitue le i^{me} mot de code. Il existe 8000 tels mots de codes, puisque $i \in \{0, .., 7999\}$.

Le gros avantage de cette approche est que si un symbole est perdu (par exemple suite à la perte d'un datagramme IP), les 8000 mots de code sont tous affectés de la perte d'un bit à la même position dans le mot de code (les trois "?" de la Fig. 2.4). Ceci signifie que le système linéaire n'est résolu qu'une seule fois (par exemple par pivot de Gauss), ce qui est un avantage décisif.

Une fois la succession d'opérations élémentaires (des simples eXclusive OR (XOR) dans le cas d'un code binaire) connue pour effectuer le décodage, cette succession est appliquée à l'identique aux 8000 mots de code. En pratique ceci s'implante efficacement avec un codec logiciel : chaque opération XOR, au lieu de se faire entre deux bits du même

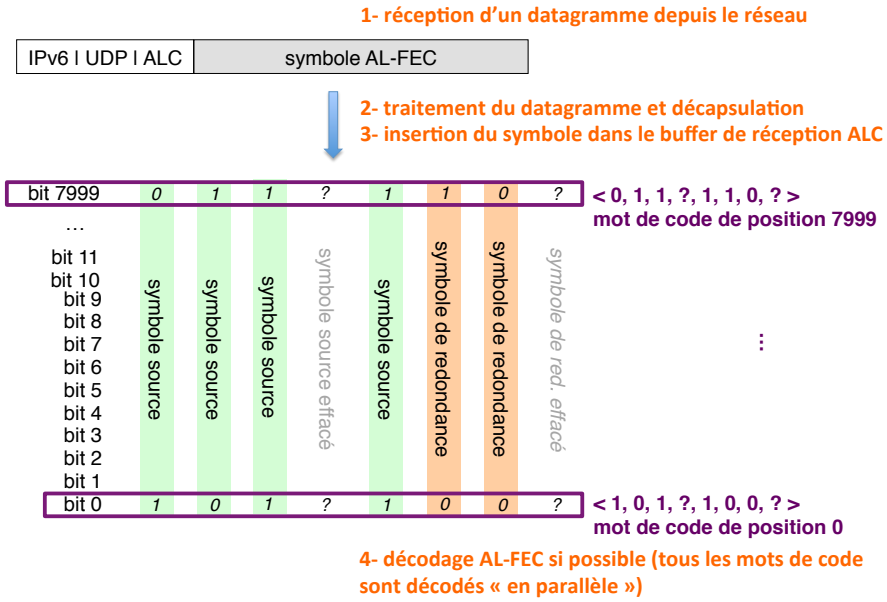


Figure 2.4: Traitements des datagrammes reçus et liens entre mot de code et symbole avec un code binaire.

mot de code, se fera *en parallèle* sur tous les bits des deux symboles (voir l'exemple de fonction XORant des symboles de la section 6.3).

Le cas des codes non binaires :

Considérons des codes Reed-Solomon sur $GF(2^8)$. Le code travaille alors sur l'élément de 8-bits (octet), et chaque opération élémentaire au sein du code se fait octet par octet. Le mot de code est donc considéré comme étant constitué d'une suite d'octets.

Les liens entre symbole et mot de code sont similaires au cas binaire. Considérons des symboles de taille 1000 octets. Dès lors, l'ensemble des i^{mes} octets (et non bits) de position i dans les n symboles d'encodage constitue le i^{me} mot de code[14]. Il existe 1000 tels mots de codes, puisque $i \in \{0, .., 999\}$. Ici aussi, le gros avantage de cette approche est que, dans le cas d'effacements durant la transmission (par exemple dans le cas d'une perte d'un datagramme IP), les 1000 mots de code seront tous affectés des mêmes effacements, aux mêmes positions. Le même schéma de décodage pourra alors être appliqué aux 1000 mots de code. C'est l'approche utilisée en pratique [40].

[14] Il est possible de définir d'autres façons de lier éléments et symboles. Par exemple il est possible de considérer les différents octets d'un symbole comme étant des éléments consécutifs d'un unique mot de code. Ainsi, en reprenant l'exemple, les 1000 octets du premier symbole constituent les 1000 premiers éléments du mot de code. Avec un code sur $GF(2^8)$, ceci excède bien sûr la dimension maximum du code. En revanche cela devient possible avec un code sur $GF(2^{16})$, puisque dans ce cas chaque symbole contribue pour 500 éléments (de taille 16-bits maintenant) dans le mot de code. Cependant cette approche est largement sous optimale du fait qu'elle oblige à travailler sur $GF(2^{16})$ au lieu de $GF(2^8)$, avec un code de dimension plus élevée. De plus la perte d'un symbole se répercute par la perte de 1000 éléments consécutifs. Ce n'est donc clairement pas l'approche utilisée en pratique.

2.4 *Espace vectoriel, matrices et graphes : applications aux codes bloc linéaires*

Avant d'aller plus loin, faisons quelques rappels pratiques en matière de codes linéaires (essentiellement pour les codes bloc qui constituent l'essentiel du document).

Les codes AL-FEC qui nous intéressent travaillent sur un "corps fini" (ou "corps de Galois") à q éléments noté $GF(q)$ (la notation $\mathbb{F}_q$ est parfois rencontrée dans la littérature).

> → *En pratique nous nous limitons au cas où q est une puissance de 2, car l'addition et soustraction correspondent à des XOR bits à bits, facilement implémentables. Nous travaillons donc sur $GF(2^m)$ où m est le nombre de bits de chaque élément du corps.*

> → *Dans le cas de codes binaires (LDPC-Staircase par exemple), nous nous restraignons au cas où $m = 1$. Ainsi les espaces vectoriels $GF(2)^k$ et $GF(2)^n$ sont des vecteurs constitués respectivement de k et n bits.*

Définition : un **code linéaire** $\mathcal{C}$ de longueur n et de dimension k est un sous-espace vectoriel de dimension k de l'espace vectoriel $GF(2^m)^n$. On le note code (n, k).

Définition : on appelle les éléments de $\mathcal{C}$ des **mots de code**.

> → *Les mots de code sont par exemple les séquences de bits après encodage FEC de la Fig. 2.1 dans le cas d'un code binaire, où $m = 1$.*

Les codes étant linéaires, par définition *toute combinaison linéaire de mots de code est également un mot de code*. C'est une propriété importante de ces codes.

Définition : On appelle **matrice génératrice**, $G \in \mathcal{M}_{k,n}(GF(2^m))$, de taille $k \times n$, la matrice de l'application linéaire de

$$GF(2^m)^k \rightarrow GF(2^m)^n$$

dont l'image est le code, c'est à dire générant tous les mots de code possibles. Les lignes de la matrice G d'un code (n, k) linéaire forment une base de l'espace vectoriel du code.

Dès lors, un élément X de $GF(2^m)^k$ est encodé en un élément Y de $GF(2^m)^n$ par :

$$Y = X\,G \tag{2.1}$$

Définition : Lorsque X se retrouve dans le vecteur encodé Y, on parle de **code systématique**. La matrice génératrice d'un tel code contient alors la matrice identité et peut s'écrire :

$$G_{\text{syst}} = [Id_k|C] \tag{2.2}$$

→ *En pratique, ceci signifie que les symboles sources font également partie des symboles de redondance, ainsi que c'est le cas dans l'exemple de la Fig. 2.3. Cette propriété est importante dans les champs d'application que nous considérons car un récepteur n'observant pas de perte doit être en mesure d'exploiter immédiatement les symboles source reçus sans effectuer de décodage.*

Définition : Soit $\mathcal{C}$ un code (n,k). On dit que $H \in \mathcal{M}_{(n-k),n}(\mathbb{F}_q)$ est une **matrice de parité** de ce code ssi $\forall\, X \in \mathcal{C}$, $HX = 0$ et H est de rang plein.

La matrice H définit donc un **système de** $(n-k)$ **équations linéaires de** n **variables**, que chaque mot de code devra satisfaire.

→ *Ce système (ou de façon équivalent la matrice G) sera utilisé lors de l'encodage, sachant que les positions sources du mot de code sont connues alors que les positions de redondance sont inconnues.*

→ *Ce système sera aussi utilisé lors du décodage, et les positions effacées du mot de code (qu'ils correspondent à des informations source ou redondance) sont alors reconstruites à partir des positions connues car reçues.*

Les matrices H et G_{syst} sont liées l'une à l'autre par :

$$H = [-C^T | Id_{n-k}] \tag{2.3}$$

→ *Les codes LDPC sont définis à partir de leur matrice de parité H, en suivant des règles de construction spécifiques. Comme H ne fait pas nécessairement apparaître la sous-matrice Id_{n-k} dans le cas général, l'encodage nécessite de résoudre le système linéaire (voir ci-dessus) afin de produire G. Nous verrons que les codes LDPC-Staircase sont spécifiés de façon à ce que l'obtention de G soit immédiate (chapitre 4.3).*

Définition : Les relations établies par la matrice H entre symboles source et symboles de redondance peuvent être représentées au moyen d'un **graphe biparti** (Fig. 2.5), encore appelé **graphe de Tanner** dans le cas des codes LDPC que nous considérerons [98].

L'ensemble U du graphe biparti est constitué des "noeuds d'information" (à savoir l'ensemble des symboles source et redondance), alors que l'ensemble V est constitué des "noeuds de contraintes" (à savoir les équations du système linéaire). Pour chaque équation (noeud de contrainte), un arc entre U et V représente la participation du symbole (source ou redondance) dans l'équation.

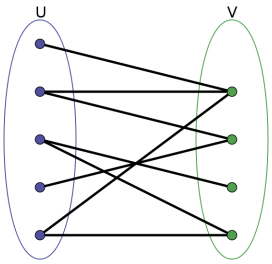


Figure 2.5: Expemple de graphe biparti (http://fr.wikipedia.org/wiki/Graphe_biparti).

2.5 *Terminologie*

Finissons en rappelant la terminologie qui sera utilisée dans tout ce document, et qui suit celle de l'IETF [105] (table 2.1).

Nom	*Description*
schéma FEC	spécification IETF décrivant les aspects protocolaires requis pour utiliser une famille de code donnée dans un système de transmission. Le schéma associé aux codes LDPC-Staircase pour l'usage au sein des protocoles ALC/LCT et Negative acknowledgment Oriented Reliable Multicast Protocol (NORM) est décrit dans le RFC 5170
codec	implantation (logicielle dans notre cas) des parties codeur et décodeur d'un code AL-FEC
symbole	unité de données que l'on fournit ou récupère du codec, via l'API dédiée
symbole source	symbole de données de l'utilisateur fourni à l'encodeur AL-FEC (cas d'un émetteur)
symbole d'encodage	symbole en sortie d'encodeur AL-FEC (cas d'un émetteur). Encore appelé "symbole de parité" avec des codes binaires
symbole de redondance	symbole d'encodage qui n'est pas un symbole source. Ils sont en nombre $n - k$ dans le cas d'un code systématique
dimension du code	nombre de symboles sources, noté k
longueur du code	nombre de symboles d'encodage, noté n
taux de codage, CR (ou "code rate")	ratio : $CR = k/n \in]0;1]$. La valeur est d'autant plus proche de 0 qu'il y a un grand nombre de symboles de redondance
bloc AL-FEC	groupe de k symboles sources considérés ensemble lors d'un encodage
code systématique	code pour lequel les symboles sources font partie des symboles d'encodage (NB : tous les codes de ce document sont systématiques)
code Maximum Distance Separable (MDS)	code doté de capacités de correction idéales (par ex. les codes Reed-Solomon)
canal à effacement de paquets	extension du canal BEC au cas où l'unité de transmission (et de pertes) est le paquet et non le bit

Table 2.1: Terminologie AL-FEC [105].

Isabelle sur sa branche perchée, au petit matin, Hautes Alpes, mai 2011.

3
Qu'est-ce qu'un bon code AL-FEC ? Critères d'usage et de performance

Résumé :
Ce chapitre présente les critères d'usage et de performance permettant d'évaluer des codes AL-FEC. Nous montrons que le choix de codes AL-FEC est nécessairement un problème multi-critères, techniques et non techniques, souvent antagonistes. Dès lors le choix final résultera forcément d'un compromis adapté au champ d'application visé.

3.1 *Introduction : critères d'usage et métriques de performance*

Afin de répondre à la question "Qu'est-ce qu'un bon code AL-FEC ?", nous distinguons deux types de critères techniques :

- *des critères d'usage :* ce sont des critères qui visent à déterminer de façon **qualitative** si, pour un champ d'application visé, les codes AL-FEC envisagés peuvent être utilisés efficacement ou non[1]. Ces critères, qui sont des appréciations globales sur l'adéquation d'une solution AL-FEC pour un usage donné, seront détaillés en section 3.2.
- *des critères de performance :* ce sont des critères qui **quantifient** les performances des codes AL-FEC envisagés, sous différents points de vue et de façon objective. Ces critères prennent la forme de métriques quantitatives que nous détaillerons en section 3.3.

[1] Par exemple, un code qui limite strictement la valeur maximale de dimension (k) ou de longueur (n) aura du mal à être efficace pour protéger un gros fichier, même si ce code a des capacités de correction optimales (code MDS). C'est le problème du collecteur de coupons (voir plus bas, section 3.2.1).

3.2 *Critères d'usage*

3.2.1 *Petit bloc ou grand bloc ?*

Les codes AL-FEC sont susceptibles d'être utilisés dans de nombreux contextes différents, pour protéger des contenus variés. Ces différences

vont impacter la taille du bloc source, à savoir le paramètre k (ou "dimension" du code AL-FEC).

Cas des applications de type streaming : Dans le cas d'un usage au sein d'une application de streaming, pour un flux multimédia, il faudra contenir la latence de décodage dans des limites compatibles avec les éventuelles contraintes temps réel du flux. Pour cela il n'y aura d'autre solution que de limiter la taille des blocs et des AL-FEC de type "petit bloc" seront en général suffisants. Le cas des très hauts débits ou de systèmes de protection globale d'un grand nombre de flux peut malgré tout nécessiter l'usage de codes grand bloc, du fait du nombre élevé de paquets par seconde généré[2].

[2] Nous verrons au chapitre 11 l'approche FECFRAME qui permet par exemple de protéger globalement un grand nombre de flux.

Exemple :
Soit l la latence de décodage maximum permise (on ignore les autres facteurs de délai), par exemple $l = 1$ seconde. Soit d la durée couverte par un symbole du flux de données : par exemple $d = 20$ millisecondes. Alors la taille maximum du bloc ne doit pas excéder :

$$k_{max\_rt} = \frac{l}{d}$$

On trouve donc : $k_{max\_rt} = 50$ symboles. Après encodage, les n paquets résultant devront être transmis à un débit au moins égal à n/l symboles par seconde, avec $n = k_{max\_rt}/CR$. Moyennant ces précautions, il faudra un délai au plus de l secondes pour recevoir un sous ensemble (du fait des pertes) des n symboles, et donc décoder et reconstruire les symboles source perdus le cas échéant (si l'on néglige le temps de décodage). En pratique, k_{max_rt} limite à quelques dizaines (rarement centaines) les tailles de blocs dans ce type d'usage.

Cas des applications de type transfert de fichiers : Dans le cas d'un usage au sein d'une application de transfert de fichiers (plus généralement d'objets, voir chapitre 10), la situation est très différente. Il est possible que l'on ait à transférer à la fois de petits objets et de (très) gros objets. Obtenir des performances de correction optimales impose que l'objet dans sa totalité soit encodé en une passe, au sein d'un unique encodage FEC[3]. Une même session pourra comporter à la fois des petits et des "gros" blocs, dans les limites des possibilités des codes AL-FEC.

[3] Lorsqu'un objet est encodé via un unique bloc, un symbole de redondance quelconque peut être utilisé pour recouvrer n'importe quel symbole source perdu. En revanche, si l'objet a dû être découpé en plusieurs blocs, chacun de taille compatible avec les limites de la solution AL-FEC, un symbole de redondance choisi aléatoirement aura un pouvoir correcteur limité à son bloc d'origine. Nous tombons alors sur le classique **problème du "collecteur de coupons"** qui va impacter les performances de correction lors d'un usage pratique des codes.

Bilan : Il apparaît que la solution AL-FEC doit être en mesure de travailler efficacement parfois sur des blocs de petite taille (quelques dizaines de symboles), parfois sur des blocs de taille importante (quelques dizaines de milliers de symboles). Les codes Reed-Solomon sur $GF(2^4)$ et même $GF(2^8)$ sont typiquement des codes dits "petit bloc",

alors que les codes LDPC-Staircase sont des codes dits "grand bloc". Pouvoir basculer entre ces deux codes sera souvent indispensable en pratique. Avoir une solution AL-FEC unique qui soit aussi efficace dans les deux cas apportera un surplus de confort.

Précisons ce que signifie "grand bloc" :
Avec des codes type LDPC-Staircase, la taille maximum du bloc dépend grandement de l'algorithme de décodage utilisé et des ressources disponibles (CPU surtout, dans une moindre mesure besoins mémoire crête car il existe des techniques pour limiter cette consommation). Ainsi, sur un smartphone et notre codec LDPC v2.2, décoder un bloc avec $k = 8192$, $CR = 2/3$, $N1 = 7$ sera possible avec une vitesse de 134 Mbps en décodage ML (taux de pertes de 33,28%), contre 609 Mbps en décodage ITerative decoding (IT) (taux de pertes de 5%) (cf. chapitre 6). Le ratio est de 4,5 entre les deux techniques pour ces paramètres.

3.2.2 *De l'utilité des petits code rates*

Certains champs d'application des codes AL-FEC sont susceptibles de nécessiter de transmettre un nombre important de symboles de redondance pour un bloc donné. Dit autrement, ces champs d'application vont bénéficier de la capacité de codes à travailler avec un petit code rate, $CR = k/n$.

Exemple :
Considérons une transmission de contenus au sein d'un carrousel FLUTE/ALC, en boucle, sur une longue période de temps [1] [2] [89]. Dans ce cas il peut être bénéfique de pouvoir produire un nombre important de symboles de redondance afin de minimiser le risque qu'un récepteur, qui piocherait aléatoirement des symboles durant un laps de temps élevé, reçoive deux fois le même. Il y a donc une incitation à pouvoir travailler sur un code rate relativement faible, voir dans l'idéal de pouvoir produire une infinité de symboles de redondance.

Typiquement, nous pouvons qualifier de "petit" code rate une valeur inférieure ou égale à 1/3, c'est à dire lorsque $n \geq 3 * k$. Le volume de redondance excède alors deux fois celui des données sources.

A t-on besoin d'aller bien plus bas ? Il est tentant de pousser plus loin le concept et de vouloir produire une quasi-infinité de symboles de redondance. Nous parlerons alors de codes "rateless", dont les codes Luby Transform (LT), Raptor$^{(tm)}$ [48] et RaptorQ$^{(tm)}$ [49][91] sont les dignes représentants. Si l'idée semble intéressante, en a t-on vraiment

besoin ? Sans vouloir (ni pouvoir) répondre définitivement à cette question, prenons juste un exemple pour comparer une approche rateless et une approche où le code rate est fixé, et comparons les résultats pratiques.

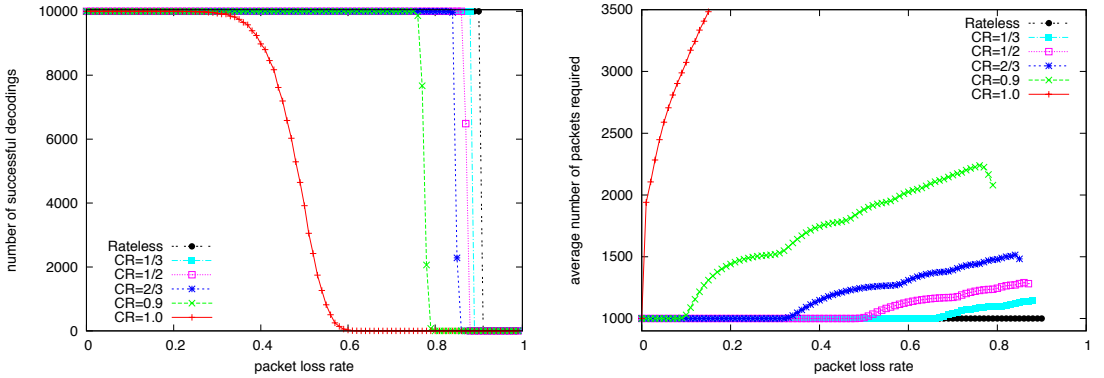
Suite de l'exemple :
Soit un objet composé d'un unique bloc de taille k symboles. Nous désirons transmettre cet objet selon une approche de type FLUTE/carrousel. Le serveur désire transmettre N paquets en tout au sein d'un ou plusieurs cycles de ce carrousel. Pendant ce temps, les clients intéressés viennent télécharger à leur guise l'objet, peut être en se connectant plusieurs fois[4]*. Deux possibilités s'offrent :*

- *code rateless : le serveur choisit bien sûr un code rate $CR = k/N$, nécessaire et suffisant pour transmettre des symboles tous distincts;*
- *code à code rate non nul supérieur à k/N: l'encodeur choisit le code rate minimum possible. Le serveur dispose donc de $n = k/CR$ symboles, qu'il devra répéter autant de fois que nécessaire jusqu'à avoir transmis N symboles. Nous supposons que le serveur a définit un ordre de transmission totalement aléatoire, pour l'ensemble de ces N symboles.*

Le premier cas est optimum dans le sens où il évite toute réception dupliquée de paquets. Dans le deuxième cas, l'approche induit une probabilité non nulle qu'un récepteur observe des symboles dupliqués. La question est de quantifier la perte d'efficacité en pratique liée à cette probabilité.
Nous supposons une distribution Independent and Identically Distributed (IID) des pertes sur le canal avec une probabilité p_e d'effacement ($p_e \in [0;1]$) et des codes idéaux. Nous choisissons $k = 1000$ symboles source ainsi qu'un nombre total de paquets transmis de $N = 10000$. Ceci signifie qu'un code pouvant descendre à un code rate $k/N = 0.1$ sera en mesure de transmettre N paquets distincts et pourra ainsi tolérer jusqu'à $N - k = 9000$ effacements (soit $p_e = 0.9$). A l'inverse, un code ne descendant pas en dessous d'un code rate $CR = 2/3$ peut générer $n = k/CR$ paquets distincts ce qui conduira à $N/n = 6.667$ cycles de carrousel. Il y aura alors potentiellement des réceptions dupliquées de paquets et le taux de pertes maximum toléré sera alors inférieur. Pour évaluer ce phénomène, pour un code rate donné et un taux de pertes donné, nous effectuons 10000 tests, puis nous comptabilisons le nombre de tests ayant permis un décodage.

La figure 3.1-(a) montre les résultats obtenus pour différents code rates. Sans surprise, ne pas disposer d'encodage FEC conduit à de piètres performances. En revanche, l'ajout d'un très faible nombre de symboles de redondance ($CR = 0.9$) apporte d'énormes bénéfices. Reduire d'avantage le code rate

[4] Une situation typique dans le cas d'un accès via une connexion sans fil de qualité variable.



(a) nombre de décodages réussis = f(code rate)

(b) nombre moyen de paquets nécessaires = f(code rate)

Figure 3.1: Impacts du code rate sur le nombre de décodages réussis (gauche) et le nombre moyen de paquets nécessaires au décodage en mode carrousel.

augmente ce bénéfice, mais dans des proportions qui se réduisent à mesure que le code rate diminue.
La table 3.1 synthétise les taux de pertes maximums permettant une probabilité de succès de décodage au moins égale à 0.5*, ainsi que le gain permis par un code rate donné par rapport au code rate immédiatement supérieur. Nous voyons qu'il ne sera pas nécessaire de descendre en dessous de* $2/3$*, voire* $1/2$*, en pratique, le gain se réduisant significativement. En revanche nous voyons, figure 3.1-(b), que le nombre moyen de paquets devant être reçus pour pouvoir décoder est significativement plus élevé lorsque l'on s'approche de la limite du taux de pertes maximum toléré, ce qui se traduit par un temps d'attente plus élevé. Disposer de codes pouvant descendre à un code rate de* $1/2$*, voire* $1/3$*, sera intéressant. Mais le phénomène est surtout sensible lorsque l'on approche du taux de perte maximum. Pour des taux de pertes plus faibles, la différence est très limitée ou nulle : à* $p_e = 0.2$ *et si* $CR = 2/3$*, le nombre moyen de paquets pour effectuer le décodage est identique à un code rateless, à* $p_e = 0.33$*, il en faudra* 1008.4 *(moins de* 1% *en plus), alors qu'à* $p_e = 0.40$ *il en faudra* 1148.2 *(*11.5% *de plus).*

Code rate	p_e *max pour décoder avec une proba.* 0.5	*Gain par rapp. au précédant*	*Nb. moyen de paquets nécessaires si* $p_e = 0.2$	*Nb. moyen de paquets nécessaires à la limite*
1.0 (pas de redondance)	0.48	N/A	3810.6	4674.5
0.9	0.77	60.4%	1441.1	2225.5
2/3	0.84	9.1%	1000.0	1514.6
1/2	0.87	3.6%	1000.0	1281.8
1/3	0.88	1.1%	1000.0	1144.7
0.0 (code rateless)	0.90	2.3%	1000.0	1000.0

Table 3.1: Performances obtenues en mode carrousel en fonction du code rate du code.

Cette étude, menée dans des conditions favorables aux codes rateless (le code rate optimal, 0.1, est très faible), montre que **la propriété rateless n'apporte qu'un faible gain de performance vis à vis de**

codes AL-FEC dotés d'un code rate $2/3$ **(par exemple) pour un usage au sein de carrousels.** La robustesse est quasiment la même, y compris face à des taux de pertes très élevés. Le délai de décodage pour les taux de pertes classiques est également très bon. Cette deuxième propriété est cependant moins vrai si le taux de pertes est très élevé, mais nous sommes là dans des situations très particulières.

3.2.3 *Code rate prédéfini ou non ?*

Au delà de la notion de petit code rate, il existe une deuxième question : le code rate est il prédéfini, c'est à dire figé une fois le code créé, ou est-il ajustable dynamiquement, en fonction des besoins ?

Exemple :
Une transmission est en cours, un encodage LDPC-Staircase de code rate 0.9 *a été fait, les symboles de redondance transmis, lorsque l'émetteur se rend compte que le taux de pertes est supérieur à* 10%*. Une première solution est pour l'émetteur de demander la retransmission des symboles source manquant, dont il connaît l'identité. Bien sûr, cette approche nécessite une signalisation lourde, individuelle, récepteur par récepteur, ce qui a des limites. Une seconde solution est de transmettre de nouveaux symboles de redondance, après un nouvel encodage, avec l'inconvénient majeur que l'on se retrouve avec deux systèmes disjoints à résoudre coté récepteur, un par encodage, ce qui est peu efficace. Enfin, une troisième solution est de prévoir une marge de sécurité, c'est à dire d'encoder avec un code rate significativement plus faible que prévu, "au cas où", quitte à ne jamais transmettre ces symboles de redondance excédentaires. Cela répond au besoin, mais a l'inconvénient de conduire à manipuler des systèmes linéaires de taille supérieure ce qui a un coût.*

Pouvoir disposer d'un code dont le code rate n'est pas prédéfini, comme c'est le cas pour les codes LT, Raptor$^{(tm)}$ et RaptorQ$^{(tm)}$, est donc un avantage.

Nous verrons au chapitre 7 que les codes *GLDPC-Staircase* ont également la bonne propriété de permettre de générer à la volée, en fonction des besoins, des symboles de redondance supplémentaires. Ce peut être une alternative crédible lorsque le champ d'application exige une adaptation dynamique du code rate.

Notons cependant que le standard 3GPP/MBMS offre un service de de retransmission qui permet à tout client, n'ayant pu décoder un objet lors d'une session de diffusion, de demander la retransmission via une connexion HTTP/TCP vers un serveur dédié des symboles manquants (NB : ce service est actuellement en usage). C'est la partie droite de la pile de communication de la figure 1.2.

3.2.4 *Libres ou verrouillés ?*

Le domaine de la théorie de l'information n'échappe pas, loin de là, à la volonté de s'approprier des techniques au travers de brevets. Utiliser des codes AL-FEC connus comme étant verrouillés par des brevets est

donc un choix qui pourra être justifié dans certains contextes, mais qui a des conséquences majeures.

Une alternative est de tirer partie des travaux antérieurs, très nombreux, dans le domaine de la théorie de l'information, et d'éviter soigneusement toute technique présentant un risque significatif de tomber sous le coup de brevets. Cette approche a bien sûr ses limites puisqu'il est impossible d'avoir connaissance de la totalité des brevets pouvant s'exercer à l'encontre d'une solution AL-FEC donnée, d'autant que certains de ces brevets peuvent encore être en phase d'étude et non publiques. En revanche cette approche permet de largement limiter les risques.

Nous détaillons dans la section 4.7 notre position par rapport à cette question à nos yeux essentielle.

3.3 *Métriques de performance*

Nous avons passé en revue les critères relatifs à l'usage. Nous nous intéressons maintenant aux critères de performance. Plus précisément, nous définissons les métriques quantitatives suivantes :

- **Métrique 1 : overhead de décodage**. Il s'agit de la principale métrique pour apprécier les capacités de correction d'effacement. Elle est définie comme :

  $$overhead = \frac{nombre\_de\_symboles\_requis\_pour\_dcoder}{k} - 1$$

  Elle s'évalue en effectuant un certain nombre de tests, puis en calculant différentes statistique sur les résultats : overhead moyen (probabilité d'échec 1/2), overhead médian, et variance/percentiles autour de la moyenne ou de la médiane.

  > *→ C'est une grandeur très facile à calculer (on définit un paramétrage et l'on effectue autant de tests que nécessaire pour un précision donnée), qui donne une excellente idées des performances de correction globales des codes. Par contre cette métrique est insuffisante pour quantifier les comportements plus rares des codes, et l'on utilisera les probabilités d'échec de décodage pour cela.*

- **Métrique 2 : distance à la capacité**. Il s'agit d'une métrique très proche de l'overhead de décodage. Par définition, un code MDS est un code qui atteint la capacité du canal, ce qui, appliqué à un canal à effacements, signifie que le code $\{n; k\}$ peut recouvrer $n - k$ effacements. Dans le cas d'un code non-MDS, cette métrique, $\triangle$, mesure sa distance à un code idéal. Si $C_{\text{canal}} = 1 - p$ est la capacité du canal, où p est la probabilité d'erreur, et $C_{\text{code}} = 1 - P^*$ est

la capacité du code, où P^* est la probabilité de pertes maximum pouvant être corrigées par le code, alors on a :

$$\triangle = C_{\text{code}} - C_{\text{canal}} = (1 - P^*) - (1 - p) = p - P^*$$

Nous n'utiliserons pas cette métrique dans ce document, lui préférant l'overhead par exemple, qui est très proche.

- **Métrique 3 : probabilité d'échec de décodage**. Il s'agit ici de définir, pour un paramétrage donné, la probabilité d'échec de décodage (c'est à dire la probabilité que le décodeur n'arrive pas à recouvrer tous les symboles source perdus) soit en fonction du taux de pertes observé (figure 3.2), soit en fonction du nombre de symboles reçus (figure 3.3).

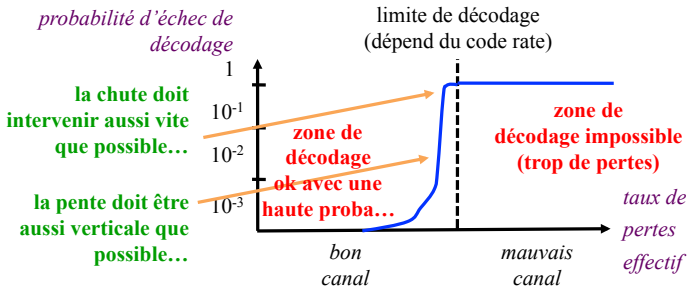


Figure 3.2: Courbe montrant la probabilité d'échec du décodage en fonction du taux de pertes effectif sur le canal.

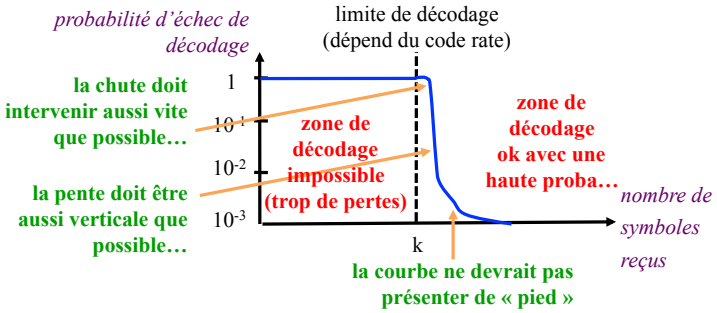


Figure 3.3: Courbe montrant la probabilité d'échec du décodage en fonction du nombre de symboles reçus.

Les deux représentations sont équivalentes et un passage de l'une à l'autre est possible. Il s'agit là d'une métrique qui va préciser le comportement des codes face à des phénomènes potentiellement rares[5].

De bons codes, du point de vue de leur capacités de correction, auront une chute rapide de la courte, la plus proche possible de la limite théorique (overhead à 0, après réception de k symboles), avec la pente la plus raide possible, et un pied de courbe le plus bas possible, voir invisible.

[5] Par exemple le comportement dans le pied de courbe, vers des probabilités d'échec de 10^{-5}, correspond à des situations dans lesquelles le système linéaire ne peut être résolu, et qui se passent typiquement dans un cas sur 100000. Avec certains codes moins performants, ou avec des décodeurs de type IT, les phénomènes en question peuvent être plus fréquents.

→ L'obtention de ces courbes est coûteuse en calculs. Elle nécessite, pour un triolet {code; ordonnancement de symboles; canal} donné, de définir le nombre minimum de symboles nécessaires pour effectuer le décodage (plusieurs tentatives de décodage sont en général nécessaires pour déterminer ce seuil). Soit ϵ_i le seuil trouvé au i^{eme} test. Avec ce résultat, nous mettons à jour un tableau comptabilisant le nombre d'échecs de décodage en fonction du nombre de symboles reçus. Ceci se fait en incrémentant toutes les entrées d'indice strictement inférieur à ϵ_i puisque le décodage a échoué en recevant ce nombre de symboles sur ce i^{eme} test. Afin d'obtenir une précision suffisante (par exemple une probabilité de 10^{-4}), nous effectuons 100 fois plus de tests unitaires (10^6 tests pour notre exemple) en faisant varier à chaque fois la graine définissant la position précise des pertes sur le canal (en pratique le code est lui aussi changé afin d'avoir un résultat moyen pour une famille de codes). Plusieurs jours de calculs sont nécessaires sur une machine multi-coeurs pour chaque courbe de ce type.

- **Métrique 4 : capacités de décodage partiel**. Cette métrique vise à évaluer les capacités du décodeur à reconstruire des symboles source manquants, même si la totalité ne peut être décodé (du point de vue de la métrique 2, il s'agirait là d'une "erreur").

- **Métrique 5 : vitesse de décodage, et complexité algorithmique de décodage**. Cette métrique évalue la complexité du décodage, ce qui se traduit de trois façons : la vitesse maximum de réception possible, la consommation en ressources de calcul (Central Processing Unit (CPU)) nécessaires pour le codec (ce codec n'est peut être qu'un élément dans une chaîne de traitement plus importante, par exemple pour un décodage vidéo), et par la consommation électrique, qui dans le cas d'un terminal autonome est un aspect clef qui en détermine l'autonomie.

- **Métrique 6 : vitesse d'encodage, et complexité algorithmique d'encodage**. Cette métrique évalue la complexité de l'encodage. Les mêmes remarques que pour le décodage peuvent être faites ici. Pour des applications de diffusion, l'un des usages majeurs des codes AL-FEC, ceci peut paraître moins critique puisque l'encodage se fait alors sur des serveurs de diffusion, disposant de ressources de calcul importantes. En revanche, la tendance à aller sur des applications pair à pair, la transmission de flux multimédia à partir d'un terminal sur des réseaux sans fils, va progressivement nécessiter que des terminaux légers effectuent un encodage FEC. Cette métrique va probablement revêtir une importance croissante dans les années à venir.

- **Métrique 7 : besoins mémoires crêtes lors du décodage**. La mé-

moire disponible n'est pas une ressource infinie, en particulier lorsque le décodage a lieu sur un terminal léger. Cette métrique vise donc à évaluer les besoins mémoires crêtes lors du décodage, afin de pouvoir dimensionner la taille du bloc en conséquence coté émetteur.

- **Métrique 8 : besoins mémoires crêtes lors de l'encodage**. Cette métrique évalue les besoins mémoires crêtes lors de l'encodage, ce qui peut être un point important lorsque cet encodage se fait sur une terminal léger.

3.4 *Au final*

Nous voyons donc que de nombreux paramètres existent. Certains sont techniques : nous avons longuement insisté sur les critères de performance, la taille maximum de blocs, le code rate minimal et le caractère fixe ou non de ce dernier. D'autres paramètres, non techniques, existent également : nous avons cité le caractère libre ou non de la solution AL-FEC, auquel nous pouvons ajouter la présence de spécifications ouvertes, la présence d'implantations open-source ou commerciales. Certains de ces paramètres sont de surcroît antagonistes, et dépendent de toutes façons du champ d'application visé. Le choix de codes AL-FEC devient alors un problème multi-critères, techniques et non techniques, et le choix final résultera forcément d'un compromis adapté au champ d'application ciblé.

Blaireau, Isère, juillet 2013.

4
Les codes LDPC-Staircase : une solution AL-FEC simple, libre et efficace

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux menés avec J. Labouré, C. Neumann, P. Moniot, D. Furodet, M. Cunche, V. Savin, J. Lacan, J. Detchart et F. Mattoussi*

Résumé :
Ce chapitre décrit les codes LDPC-Staircase sur lesquels nous avons travaillé durant plusieurs années, que ce soit pour leur spécification IETF, leur évaluation, le développement de codecs logiciels, ou leur application dans différents systèmes de transmission. Bien que très simples, nous montrons tout au long de ce document qu'ils présentent des performances qui les font s'approcher de solutions propriétaires dont la complexité est largement supérieure. Ces codes constituent donc une base solide sur laquelle nous nous sommes largement appuyés.

4.1 *Les codes "Parity Check"*

Les codes de type "Parity Check" (avec contraintes de parité) constituent une très grande classe. Ces codes sont définis par des équations de parité (ou "contraintes") que doivent respecter certaines sommes, au sens XOR, de symboles (figure 4.1, extrait de [30]). L'ensemble de ces équations définit alors un système linéaire qui devra permettre à la fois l'encodage (production des symboles de redondances connaissant la totalité des symboles source) et le décodage (récupération des symboles sources perdus connaissant un nombre suffisant de symboles). La question essentielle est dès lors comment créer ces équations pour obtenir de bons codes FEC (au sens du chapitre 3) ?

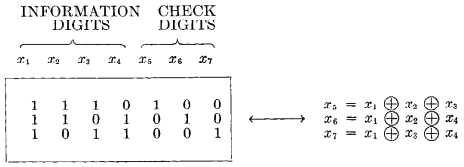


Figure 4.1: Codes "parity check".

Exemple 1 : les codes de Hamming :
Les codes de Hamming constituent un des premiers codes linéaires binaires, mis au point en 1950 par Richard Hamming. Ces codes sont de dimension $k = 2^r - r - 1$, où r est un entier ≥ 2, et de longueur $n = 2^r - 1$. Le représentant le plus connu est le code de Hamming (7,4) (figure 4.2), de code rate $CR = 0,5714$.

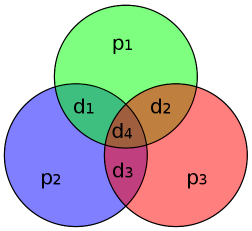


Figure 4.2: Codes de Hamming(7,4), avec (d1, d2, d3, d4) comme symboles source, et (p1, p2, p3) comme symboles de parité (tiré de `http://en.wikipedia.org/wiki/File:Hamming(7,4).svg`, sous licence CC-Attribution, Share Alike).

Pour tout ensemble de quatre symboles source (d1, d2, d3, d4), on calcule trois symboles de parité (p1, p2, p3) comme suit :

$$p1 = d1 \oplus d2 \qquad \oplus d4$$

$$p2 = d1 \qquad \oplus d3 \oplus d4$$

$$p3 = \qquad d2 \oplus d3 \oplus d4$$

La Fig. 4.2 illustre ce principe, la somme des quatre symboles intervenant dans chacun des trois cercles étant nulle. Sur le canal à erreurs, ce sont des codes capables de détecter deux erreurs et d'en corriger une, ceci quelle que soit la valeur de r. Ils sont donc utiles dès lors que le canal est de bonne qualité[1].

[1] Les codes de Hamming ont par exemple été largement utilisés dans les mémoires Error-Correcting Code (ECC) capables de détecter/corriger certaines erreurs bits souvent dues aux rayonnements cosmiques.

Exemple 2 : les codes aléatoires binaires :
Une autre famille de codes linéaires est tout particulièrement intéressante : les codes aléatoires binaires. Les équations de parités sont alors choisies aléatoirement, chaque symbole source contribuant à la somme XOR d'un symbole de parité donné avec une probabilité 1/2. Les performances sont relativement élevées et constituent habituellement la référence que tout code LDPC binaire (voir ci-dessous) cherchera à atteindre[2]. Nous reviendrons sur ces codes dans la section 5.1.1.

[2] Ces codes aléatoires binaires s'étendent naturellement au cas où les coefficients sont choisis sur un corps fini non binaire (section 4.2.1), pour des performances en termes de corrections d'effacements plus élevées encore.

4.2 *Codes LDPC*

Intéressons nous maintenant à la célèbre famille des codes LDPC, plus proches de nos champs d'application que les codes de Hamming. Ces codes LDPC ont été inventés par R. Gallager (Fig. 4.3) au début des années 1960 [29], et après une longue période où ils ont été quelque peu délaissés, ils ont connus à partir de 1995 un regain d'intérêt qui ne se dément pas depuis. Leur particularité distinctive est d'être des codes linéaires dont la matrice de parité, H, est creuse[3]. Cette matrice H définit un système d'équations linéaires entre les symboles source (ou données) et les symboles de parité (ou redondance), une entrée non nulle dans H signifiant que le symbole correspondant de la colonne est pris en compte dans l'équation (voir la section 2.4 pour un rappel sur les notions de codes linéaires bloc et matrices génératrice et parité).

Figure 4.3: Robert Gallager (`http://www.rle.mit.edu/rgallager/`)

[3] Ceci signifie que le nombre de coefficients non nuls (donc égaux à 1 avec des codes binaires) est très faible.

Il est à noter que disposer d'une matrice de parité creuse n'est pas une fin en soi. Il s'agit par contre d'un pré-requis :

- pour que l'algorithme de décodage appelé IT, de faible complexité algorithmique, fonctionne efficacement (section 4.6), et
- pour que globalement le nombre d'opérations XOR nécessaires au décodage puisse être contenu, indépendamment de l'approche choisie[4].

[4] En particulier avec l'approche de décodage Structured Gaussian Elimination (SGE) qui permet de contenir encore plus ce nombre d'opérations (section 6.2.2).

Si la matrice de parité H d'un code LDPC est creuse, il n'en est généralement pas de même de la matrice génératrice associée, G, qui permet de produire les symboles d'encodage :

$$Y = XG$$

où Y et X sont respectivement les vecteurs de symboles d'encodage et de vecteurs sources (voir par exemple les figures 4.6). Cependant il existe une exception majeure, les codes Low Density Generator Matrix (LDGM), dont chaque symbole repair est exprimé directement en tant que combinaison linéaire des symboles source dont il dépend. Ces codes LDGM présentent dès lors des matrice H et G toutes deux creuses puisque identiques à une transposition près (section 2.4). Ceci peut être un avantage lors de certaines techniques de décodage. En revanche leurs capacités de correction sont généralement faibles.

4.2.1 Codes binaires et non binaires

Deux familles de codes LDPC existent : les codes qui opèrent sur le corps fini $GF(2)$, et on parlera alors de "codes LDPC binaires", et ceux qui opèrent sur des corps finis non binaires.

Avec les versions binaires, les opérations entre symboles se font au moyen d'opérations XOR. Les vitesses d'encodage/décodage de ces codes sont en général élevées, car les opérations XOR entre symboles, du type :

$$symb_A = symb_A \oplus symb_B$$

peuvent facilement être optimisées en travaillant sur des mots de taille élevées (section 6.3).

A l'inverse, travailler sur un corps fini non binaire nécessitera de travailler indépendamment sur chaque élément des symboles. Avec $GF(2^8)$, on travaillera donc octet par octet, taille des éléments du corps fini $GF(2^8)$. Avec $GF(2^4)$, on travaillera par groupe de 4 bits (que l'on pourra recombiner par deux pour des accès mémoires plus efficaces). L'usage de tables précalculées accélère les opérations entre symboles, qui cette fois-ci sont du type :

$$symb_A = symb_A \oplus (c * symb_B)$$

où c est un coefficient sur $GF(2^8)$ [5]. Nous voyons que tout se ramène à des accès à des tables précalculées et à des opérations XOR. Pour autant, les vitesses seront en retrait par rapport aux vitesse obtenues avec des codes binaires qui ne sont pas limités par tailles des éléments du corps fini, ou la taille en mémoire des tables précalculées.

[5] Ces opérations interviennent lors du produit entre un coefficient (c) d'une ligne de la matrice de décodage par un symbole ($symb_B$) appartenant au vecteur des symboles reçus.

Dans la suite de ce chapitre, nous considérerons les codes binaires LDPC-Staircase, puis au chapitre 7, les codes GLDPC-Staircase qui sont eux non binaires.

4.2.2 *Quelques introductions aux codes LDPC pour aller plus loin...*

Plusieurs documents introduisent les codes LDPC, parmi lesquels : [90] est une introduction intéressante, qui donne une vision globale du domaine, à la fois pour les applications au canal à erreurs et à effacements, via le décodage IT classique aux codes LDPC (le décodage ML, pourtant essentiel en pratique, n'est pas considéré)[6].

[8] fournit une introduction conséquente au domaine des codes LDPC, équilibrée entre canal à erreurs et effacements, y compris le canal à effacement de paquets qui nous intéresse. A noter que, bizarrement, les auteurs semblent considérer les notions de "codes à effacements" et de "codes rateless" comme étant synonymes (p. 25).

[38] détaille les principales techniques de construction, optimisation et évaluation des codes LDPC. Si le canal BEC est considéré, entre autres, en revanche rien n'est dit des canaux à effacement de paquets. Le propos est largement orienté codes pour la couche physique.

Ces articles sont tous complémentaire à ce que l'on peut trouver dans le présent document (par exemple pour ce qui concerne les aspects théoriques).

[6] A noter que l'auteur attribue à la référence [10] la paternité du décodage Belief Propagation (BP) sur le BEC (page 13). Le décodage IT sur le canal BEC (qui est équivalent) fut en fait décrit en 1974 dans [108] (section 4.6).

4.3 *Les codes LDPC-Staircase*

Figure 4.4: David MacKay (http://www.inference.phy.cam.ac.uk/mackay/)

NOUS N'AVONS RIEN INVENTÉ ! David MacKay (Fig. 4.4), dans son livre [54], section 47.7, décrit succinctement des codes qu'il appelle "LDPC-Staircase". Il l'illustre au moyen d'une matrice H pour un code (28, 16) (reproduite Fig. 4.5) très semblable aux matrices qui seraient produites en suivant les spécifications que nous ferons plus tard (RFC 5170, [81]). Plus précisément la matrice H est constituée de deux sous matrices, H_1 (qui ne fait intervenir que les k symboles sources) et H_2 (qui ne fait intervenir que les $n-k$ symboles de parité). H_1 est remplie avec N_1 entrées à 1 par colonne ($N1 = 3$, Fig. 4.5), dont la position est choisie aléatoirement tout en garantissant une répartition la plus régulière possible sur les lignes. H_2 est dotée pour sa part d'une structure de type "escalier", encore appelée "double diagonale".

C'est de cet ouvrage que nous est venu l'idée de nous intéresser à cette famille, qui dès le départ présentait l'avantage d'être facilement encodable, et, ce dont nous nous sommes rapidement aperçu, fournit des performances en termes de capacités de correction très convenables au moyen d'un simple décodeur type IT.

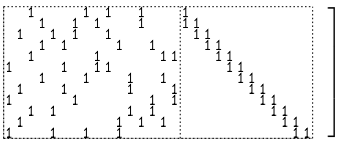
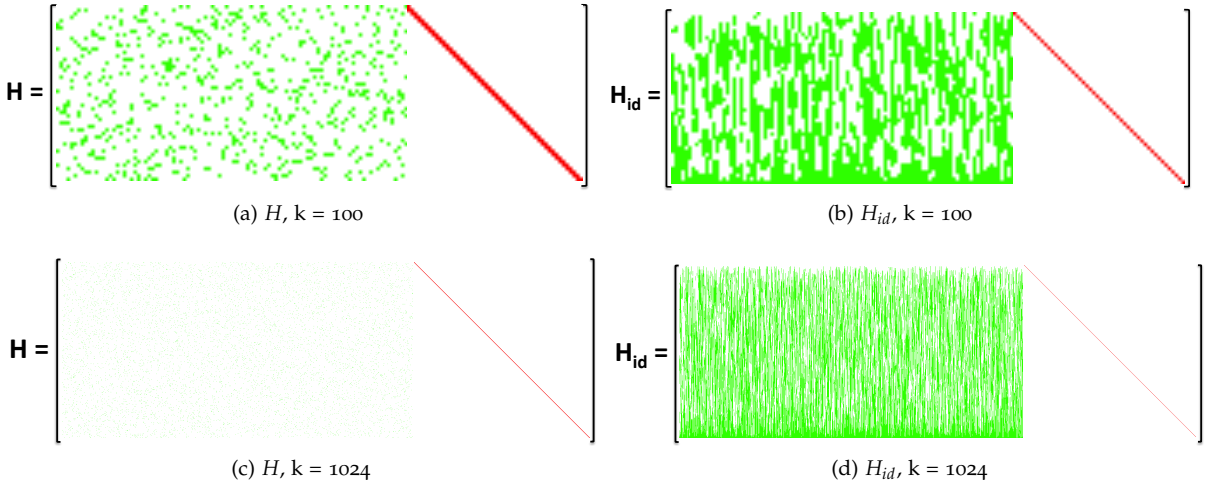
$$\mathbf{H} = \left[\right.$$  $$\left.\right]$$

Figure 4.5: Matrice de parité de type LDPC-Staircase (n=28, k=16), extraite du livre de D. MacKay.

Un autre avantage, tout aussi déterminant, est que cette famille de codes est totalement régulière. En effet, chaque colonne de H_1 possède exactement N_1 entrées (il s'agit également du degré "gauche", ou degré des noeuds de variables correspondant aux symboles sources dans une représentation par graphe biparti), et chaque ligne de H_1 (correspondant à un noeud de contrainte du graphe biparti) possède exactement :

$$d_c = \frac{N_1}{\frac{1}{CR} - 1}$$

entrées, auxquelles on peut ajouter les entrées correspondant à l'escalier dans H_2.



(a) H, k = 100 (b) H_{id}, k = 100

(c) H, k = 1024 (d) H_{id}, k = 1024

Figure 4.6: Matrices de parité d'un même code LDPC-Staircase, avec escalier ou identité en partie droite (k = 100 ou k = 1024, CR=2/3, N1 = 7).

La matrice de parité d'un code n'est pas unique. Ainsi, si par des substitutions nous remplaçons l'escalier d'un code LDPC-Staircase donné (Fig. 4.6-(a) et (c)) par une simple sous-matrice identité, nous obtenons la variante H_{id} de ce même code (Fig. 4.6-(b) et (d)).
Les densités de ces deux matrices, pour le cas d'un "petit" code LDPC-Staircase, $k = 100$, $CR = 2/3$ et $N_1 = 7$, sont largement différentes. Si la densité de la sous matrice gauche H_1 de la forme initiale est de $N_1/(n-k) = 0,14$, cette densité passe à $2640/5000 = 0.528$ avec la forme H_{id}. La densité de cette deuxième forme s'approche donc de celle d'un code binaire aléatoire (égale à 0.5), et est largement supérieure à celle de la version initiale. Avec un code LDPC-Staircase de taille plus importante, $k = 1024$, $CR = 2/3$, $N_1 = 7$, nous avons des densités de H_1 de 0.0136 et 0.495 pour les deux formes de matrices de parité. Les bénéfices apportés par la structure escalier sont d'autant plus visibles.

On voit directement les bénéfices que l'on a à travailler sur la forme classique (moins d'opérations XOR, décodage IT plus efficace), tout en bénéficiant des mêmes capacités de correction en décodage ML que la version avec la forme H_{id}, plus proche d'un code binaire aléatoire.

On notera au passage qu'avec N_1 impair (Fig. 4.6-(b) et (d)), la dernière ligne de H_{id} est nécessairement égale à la somme XOR de tous les symboles sources, puisqu'ils interviennent chacun exactement un nombre impair de fois dans les sommes XOR. A l'inverse, avec N_1 pair, cette même ligne ne fera intervenir aucun symbole source, ce qui signifie que le dernier symbole de redondance est nécessairement égal à 0 dans ce cas. Il devient inutile de le transmettre, et un récepteur pourra l'intégrer aux symboles "connus" avant même toute réception.

4.4 Travaux associés

Nous introduisons ici quelques codes remarquables, qui sont en lien direct avec les codes LDPC-Staircase.

4.4.1 Les codes "Repeat Accumulate"

Les codes LDPC-Staircase et les codes "Repeat Accumulate" relèvent du même principe constructif :

- un symbole source est **répété** car il intervient dans plusieurs équations,
- un symbole de redondance **accumule** les précédents symboles de redondance.

Le tout est associé à un mécanisme de permutation (ou "interleaving") qui permet d'apporter de la diversité dans la génération des symboles de redondance.

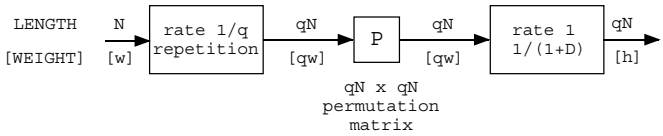


Figure 4.7: Représentation fonctionnelle des codes RA, extraite de [23].

Les codes "repeat accumulate" ont été largement popularisés avec deux travaux. En 1998, [23] décrit les codes Repeat-Accumulate (RA) qui relèvent exactement du principe ci-dessus, mais appliqué au canal AWGN. La figure 4.7 montre le bloc de répétition (de code rate $1/q$ puisque chaque symbole source est répété q fois, ce qui correspond

d'ailleurs au paramètre N_1 des codes LDPC-Staircase, section 5.1), le bloc de permutation aléatoire ("interleaving"), et enfin le bloc d'accumulation défini par :

$$y_1 = x_1$$

$$y_2 = x_1 + x_2$$

$$y_3 = x_1 + x_2 + x_3$$

$$etc.$$

Plus tard, en 2000, [37] décrit une variante irrégulière pour les canaux BEC, les codes Irregular Repeat Accumulate (IRA). La figure 4.8 montre un chaînage entre les noeuds de contrainte successifs au moyen de noeuds de parité. Ce chaînage correspond exactement à la structure escalier des codes LDPC-Staircase. Chaque noeud de contrainte a le même degré gauche, a (qui correspond à la grandeur d_c, cf. section 4.3). En revanche le degré droit des symboles source suit une distribution irrégulière : une fraction f_i de symboles sources sont impliqués dans exactement i équations. Le choix d'une distribution pour les fractions f_i est donc un élément clef pour la constitution de codes performants.

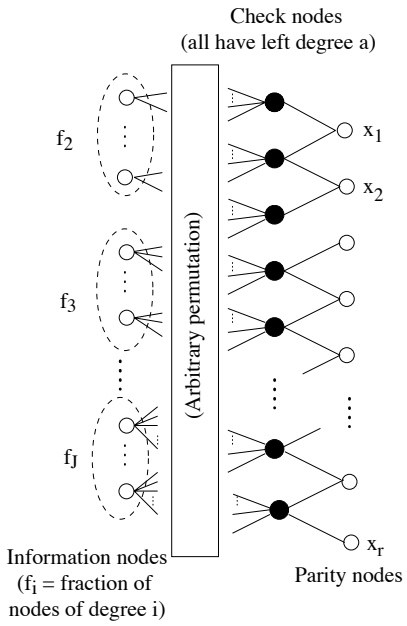


Figure 4.8: Graphe biparti d'un code IRA, extraite de [37].

Les codes LDPC-Staircase et IRA sont donc similaires, à l'exception de la construction de la matrice H_1 : régulière dans le premier cas, irrégulière dans le deuxième cas.

4.4.2 *Les codes LDPC à triple diagonale, selon Zivkoic*

Dans [107], publié en 1991, les codes suivants sont décrits :

> *Let C be a linear (512, 100) code [...], whose codewords are vectors* $x \in V_{512}$ *satisfying the following parity-checks:*
>
> $$x_i + x_{i+37} + x_{i+100} = 0, \qquad 0 \leq i \leq 411$$

(NB : les indices i ont été adaptés afin de démarrer à 0 et non à 1 comme dans [107].) La matrice de parité (Fig. 4.9) est donc constituée de trois diagonales, la première issue de s_0, la seconde de s_{37}, et la troisième de s_{100}, sachant que s_{100} est le premier symbole de redondance du code. Cette notion de diagonale qui parcourt les symboles sources et de redondance présente des analogies avec la double identité des codes LDPC-Staircase.

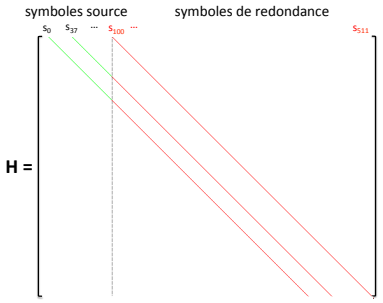


Figure 4.9: Matrice de parité (512, 100) des codes à triple diagonale de Zivkoic.

Si par substitutions on passe à la représentation équivalente H_{id} avec la matrice H_2 sous forme d'identité, la distribution des degrés droits (des symboles de redondance) devient irrégulière (Fig. 4.10) :

- les symboles s_{100} à s_{162} sont la somme de deux symboles source;
- les symboles s_{163} à s_{199} sont la somme de trois symboles source;
- les symboles s_{200} à s_{225} sont la somme de deux symboles source (une colonne s'élimine du fait des XOR;
- et cela continue, avec degrés qui varient entre trois et cinq.

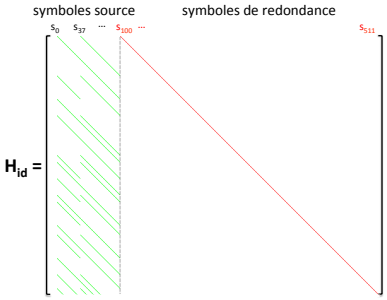


Figure 4.10: Version avec identité de la matrice de parité (512, 100) des codes à triple diagonale de Zivkoic.

[7] Plus généralement l'encodage de tout code dont H_2 a une structure triangulaire inférieure est trivial.

4.5 *LDPC-Staircase : un encodage naturellement rapide*

Du fait de la structure de H_2, l'encodage des codes LDPC-Staircase est trivial[7]. Il suffit de produire le premier symbole de parité, somme XOR de deux symboles sources (au minimum) dont l'identité est définie par les entrées à 1 dans H_1. Puis on continue avec le deuxième symbole de parité, somme du précédant symbole de parité et d'un certain nombre de symboles source, et ainsi de suite. L'encodage a donc une complexité linéaire.

En reprenant l'exemple de la figure 4.5, si on note $\{s_0, .., s_{15}\}$ les symboles source, et $\{s_{16}, .., s_{27}\}$ les symboles de redondance [8] , on a :

$$s_{16} = s_2 \oplus s_7 \oplus s_9 \oplus s_{12}$$

$$s_{17} = s_{16} \oplus s_3 \oplus s_6 \oplus s_8 \oplus s_{12}$$

$$etc.$$

[8] A noter que l'on note parfois $\{p_0, .., p_{12}\}$ les symboles de redondance (ou encore parités).

La principale limite pratique est lié aux accès aléatoires en mémoire, puisque l'on doit ajouter au symbole de parité en cours de construction un symbole source qui peut se trouver n'importe où dans l'objet d'origine. Par défaut, il est souhaitable de disposer d'une quantité de mémoire (RAM) permettant de stocker l'objet dans sa globalité.

Si ce n'est pas le cas, il peut être nécessaire d'encoder en plusieurs fois, après avoir découpé l'objet source en B blocs indépendants, la taille de ces blocs étant choisie compatible avec la mémoire disponible[9]. On y perd alors en capacité de correction mais on y gagne en vitesse d'encodage/décodage.

[9] Il est à noter que des codes tels que ceux de Zivkoic (section 4.4.2) ne créent pas de soucis d'accès aléatoire en mémoire. C'est vrai également des approches utilisant des codes convolutifs (section 12.3) du fait de la fenêtre d'encodage glissant sur les symboles sources.

4.6 *Décodage hybride : une clef pour atteindre des performances élevées*

Considérons maintenant le décodeur. Il s'agit de résoudre un système d'équations linéaires dont les variables sont les symboles source et parité inconnus (car perdus sur le canal), même si au final seules les variables sources sont pertinentes. Tous les symboles connus (car reçus) sont pour leur part sommés dans les termes constants des équations.

Deux grandes familles de décodage existent :

- les algorithmes de type itératif, ou IT, et
- les algorithmes de type ML, par exemple au moyen d'un pivot de Gauss.

Figure 4.11: Victor Zyablov (http://www.iitp.ru/en/users/100.htm)

L'approche IT (aussi connue sous le terme de "Peeling Decoder"[87] ou "Belief Propagation", BP) est la technique triviale et classique de décodage pour les codes LDPC sur un canal à effacements. Elle fut introduite en 1974 par Zyablov et Pinsker [108] ainsi :

§3. Erasure-Correction Algorithm

The algorithm described here is based on the use of check relations involving only one erased symbol (equation in one unknown). We call these check relations correction checks [2]. In this case the value of the erased symbol is equal to the value of the correction check, i.e., the sum of the values of the unerased symbols entering into the correction check, and erasure correction entails the replacement of the erased symbol with the values of the correction check.

Figure 4.12: Description du décodage IT par Zyablov et Pinsker (extrait), 1974.

En d'autres termes l'idée est de chercher une équation faisant intervenir une unique variable inconnue. Cette variable est alors égale au terme constant, somme XOR de toutes les variable connues. On réinjecte alors cette valeur dans toutes les équations où la variable apparaît et on réitère. Cette approche est très rapide mais certains systèmes, pourtant de rang plein, ne peuvent être résolus. D'où l'intérêt de l'approche ML pour achever le décodage. Ici la complexité est significative mais en revanche la capacité de correction est élevée puisque tout système de rang plein est résolu.

L'idée [19] consiste à pousser le plus loin possible l'usage du décodage IT, tant que de nouveaux symboles sont disponibles. Si l'approche IT n'a pu aboutir, nous passons alors à un décodage ML, en utilisant le système (généralement simplifié) résultant de la phase IT. Ce décodage hybride offre une *flexibilité idéale*, avec la possibilité d'optimiser soit les capacités de correction, soit la charge CPU et/ou l'autonomie énergétique. Le curseur peut en effet être poussé plus ou moins loin en fonction des capacités de traitement ou de l'autonomie restante de l'hôte. Ainsi un terminal mobile peut décider de ne décoder en ML que les systèmes simplifiés de taille inférieure à un certain seuil, seuil qui peut être dynamique et dépendre également du type d'alimentation (secteur ou batterie).

Il est aussi possible, avec une approche "Elimination de Gauss Structurée" de rendre le passage entre décodage IT et ML naturel, les deux techniques étant imbriquées l'une dans l'autre.

Nous reviendrons plus en détails sur ces aspects pratiques au sein du chapitre 6.

4.7 Quid des brevets ?

Tout au long de notre travail, nous nous sommes efforcés de **rester le plus éloignés possible de brevets** *dont nous avions connaissance. Il s'agit là d'un choix assumé, très impactant, qui a guidé notre démarche scientifique durant toutes ces années.*

Ainsi, la structure régulière des codes LDPC-Staircase nous a permis de nous éloigner des brevets relatifs à la construction de codes LDPC irréguliers. Ce choix est **impactant** car il est connu que des codes irréguliers se comporteront mieux face à un décodage IT. Il est **impactant** car plus généralement nous nous interdisons de nombreuses pistes qui auraient inévitablement permis d'obtenir de meilleurs résultats. Enfin il est **impactant** d'un point de vue publications, car travailler sur des codes aussi simples ouvre la porte à des critiques aisées.

Mais d'un autre coté, ce choix est aussi un **challenge** : jusqu'où peut on aller avec des solutions souvent très simples et (à priori) libres de droits ? Nous allons montrer que l'on peut approcher de très près des codes qui ont été largement optimisés, et même, sur certains critères et dans certaines conditions, les dépasser... Il s'agit là d'une de nos principales contributions. Dans tous les cas, cette démarche est la seule qui nous permette de construire des solutions technologiques ayant le maximum de chances d'être libres d'usage, même si des risques subsistent inévitablement.

4.8 Conception de codecs logiciels pour les codes LDPC

Nous avons durant ces années développé plusieurs codecs logiciels `C/C++` (Table 4.1) :

Table 4.1: Différents codecs LDPC que nous avons développés.

Nom	Statut	Description
LDPC, `http://planete-bcast.inrialpes.fr`	open-source	codec LDPC-Staircase et LDPC-Triangle, langage C++, décodeur de type IT
OpenFEC, `http://openfec.org`	open-source	codec LDPC-Staircase, 2D parity check, Reed-Solomon sur $GF(2^4)$ et $GF(2^8)$, langage C, décodeur IT et ML
LDPC-Staircase avancé	commercial	codec LDPC-Staircase optimisé, langage C, décodeur IT et ML

Ces codecs ont largement bénéficié du simulateur LDPC développé par Radford Neal (figure 4.13), disponible sur : `http://www.cs.utoronto.ca/~radford/ldpc.software.html`.

Figure 4.13: Radford Neal (`http://www.cs.utoronto.ca/~radford/`)

Plus précisément nous avons repris plusieurs structures de données et méthodes associées de ce simulateur (représentations creuse et dense des matrices, méthodes de conversion, et algorithme de construction de la matrice de parité). Seule la version commerciale s'en éloigne et utilise des structures plus élaborées afin de gagner en compacité et en vitesse d'encodage/décodage (chapitre 6).

Ces codecs sont essentiels dans notre travail et servent de base aux évaluations de performance des chapitres 5 et 6.

4.9 *Pour plus d'informations...*

Nos publications relatives au thème :

- "RS + LDPC-Staircase Codes for the Erasure Channel: Standards, Usage and Performance", V. Roca, M. Cunche, C. Thienot, J. Detchart, et J. Lacan, 2013, [77]
  → *Cette publication détaille un certain nombre d'optimisations mises en oeuvre ainsi que les résultats de la compétition du 3GPP qui a vu notre proposition RS+LDPC-Staircase arriver en tête.*

- **[RFC 6816]** "LDPC-Staircase Forward Error Correction (FEC) Schemes for FECFRAME", V. Roca, M. Cunche, J. Lacan, [75]
  → *Ce document explique comment utiliser les codes LDPC-Staircase (RFC 5170) dans un contexte streaming.*

- **[RFC 5170]** "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes", V. Roca, C. Neumann, D. Furodet [81]
  → *C'est le standard de référence des codes correcteurs LDPC-Staircase, qui en spécifie tous les détails.*

- "Optimizing the Error Recovery Capabilities of LDPC-staircase Codes Featuring a Gaussian Elimination Decoding Scheme", M. Cunche et V. Roca, 2008, [22]
  → *Cette publication montre pour la première fois que l'on peut obtenir des performances en capacité de correction excellentes avec les codes LDPC-Staircase au moyen d'un décodeur IT/ML, tout en ayant une complexité de décodage raisonnable (NB: notre codec a depuis été considérablement amélioré).*

- "Le RFC 5170 en pratique : conception et évaluation d'un codec AL-FEC LDPC-Staircase hautes performances", M. Cunche and V. Roca", 2008, [20]

- "Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon Small Block FEC Codec", V. Roca et C. Neumann, 2004,

[80]
→ *Ce rapport de recherche de 2004 fait le point sur ce qui va devenir les codes LDPC-Staircase et Triangle du RFC 5170.*

A l'écoute... Haute Savoie, 2012.
"Bon sang, c'est quoi ce truc vert qui fait clac ?"

5
Performances des codes LDPC-Staircase : capacités de corrections

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux communs menés avec C. Neumann, M. Cunche, J. Detchart, F. Mattoussi, V. Savin.*

Résumé :
Ce chapitre est consacré aux performances des codes LDPC-Staircase du point de vue de ses capacités de correction aux effacements. Après une présentation des paramètres impactants, nous fournissons des résultats de performances obtenues au moyen des outils du projet OpenFEC.org.

5.1 Paramètres impactant les capacités de correction

Décoder au moyen d'un code LDPC-Staircase revient à résoudre un système linéaire sur $GF(2)$, le code étant un "code binaire". Toute technique de résolution de système linéaire peut donc être considérée, avec des impacts cependant en termes de performances obtenues. Dans ce chapitre nous nous focaliserons sur les capacités de correction et nous laisserons au chapitre 6 les autres métrique de performance.

Les codes LDPC-Staircase ont des performances en termes de capacité de correction d'effacements qui dépendent de plusieurs paramètres :

- **la dimension du code (k) :** les codes LDPC (et donc ceux qui nous intéressent) sont connus pour avoir des performances assymptotiquement bonnes. Il faut donc s'attendre à avoir de moins bons résultats sur de petites valeurs de k, puis une amélioration progressive à mesure que k augmente.

  Une solution pratique pour augmenter (quelque peu artificiellement, certes) la valeur k pour un objet donné, est de considérer des symboles plus petits, et donc plus nombreux. Ces symboles sont alors regroupés à plusieurs dans un paquet de données (afin de limiter le surcoût lié aux entêtes réseaux). Plus précisement les petits objets auront $G \geq 1$ symboles par paquet [81] [1].

- **la longueur du code (n), ou de façon équivalente, le code rate**

[1] Les symboles à insérer dans un paquet seront choisis aléatoirement s'il s'agit de symboles de redondance, alors qu'il seront en séquence s'il s'agit de symboles source. Ceci est fait afin de préserver de bonnes capacités de correction [81].

$CR = k/n$ **:** les codes LDPC-Staircase ont un code rate prédéterminé lors de leur construction : la matrice de parité a ainsi une taille $(n-k) \times n$. Modifier n (ou CR) revient à modifier profondément la matrice de parité, ce qui impacte les performances du code[2].

[2] A l'inverse, les codes Raptor^{tm} (Raptor), RaptorQ^{tm} (RaptorQ) ou LT ne font pas intervenir de code rate *prédéfini* (on parle de code "rateless"), et les performances ne sont plus tributaires du code rate effectif. Cette particularité simplifie indéniablement les évaluations de performances, sans pour autant être décisif en tant que tel en général.

- **le type de décodage effectué, de type itératif (IT) et/ou à maximum de vraisemblance (ML) :** le décodage de type IT ne permettra pas nécessairement d'atteindre les capacités de correction intrinsèques du code, pour un schéma de transmission donné. Les performances obtenues seront donc moindres, mais la complexité algorithmique sera également plus faible ce qui peut être un avantage décisif pour certains usages. Pour atteindre les capacités de corrections optimales il faudra recourir à un décodage de type ML, ceci au détriment de la complexité algorithmique. Pour les codes sur le canal à effacement, une approche de type "élimination de Gauss" (permet d'effectuer ce décodage ML. La variante SGE (section 6.2.2) permet d'atteindre le même résultat tout en ayant une moindre complexité algorithmique (voir le chapitre 6). Notons que les approches IT et ML ne sont pas antagonistes. Le décodage peut démarrer par un décodage IT, puis s'il n'aboutit pas, finir par un décodage ML sur le système simplifié le cas échéant par le décodage IT. L'approche SGE peut aussi être vue comme une passerelle entre les deux décodeurs, grâce à un subtil mélange entre les techniques IT et élimination de Gauss.

- **la densité du système linéaire, contrôlée par le paramètre N_1 :** un décodage IT est tributaire de la possibilité de trouver dans le système linéaire à résoudre des équations dont il ne reste plus qu'une seule variable. La densité du système linéaire défini par la matrice H impacte fortement les capacités de correction : à mesure que le décodage progresse, la probabilité de trouver une équation qui a la propriété recherchée est d'autant plus grande que le système linéaire initial était creux.

  La situation est totalement opposée dans le cas d'un décodage ML: la probabilité que le système soit non singulier et donc inversible augmente avec sa densité.

  Avec les codes LDPC-Staircase, la densité de la matrice de parité, et donc in-fine, du système linéaire à résoudre, est contrôlée par le paramètre N_1 (cf. section 5.2). Ainsi le choix d'une valeur de N_1 permet de trouver le bon compromis entre capacités de correction et complexité algorithmique. Ce choix sera fait par l'encodeur, pour un champ d'application donné, et l'information N_1 transmise au décodeur (voir le chapitre 8).

- **le jeu de symboles disponibles pour effectuer le décodage :** ce jeu

est constitué des symboles transmis qui n'ont pas été perdus sur le canal. Dans le cas des codes LDPC-Staircase, la structure en escalier crée un chaînage entre symboles de redondances. Avec un décodage IT, les performances seront largement impactées par l'identité et l'ordre dans lequel les symboles seront donnés au décodeur [63]. Ceci est moins vrai avec un décodeur ML qui ne sera pas sensible à l'ordre de soumission des symboles reçus au décodeur. Néanmoins, pour un même nombre de symboles reçus, les performances seront variables selon l'identité des symboles reçus.

Deux paramètres impactent ce jeu de symboles disponibles : le séquencement des symboles transmis, sur lequel l'émetteur peut agir, et la nature des pertes, qui elle est une donnée du canal et sur lequel on a moins de contrôle.

- **le corps fini sur lequel les opérations se font :** les codes LDPC-Staircase sont des codes binaires, travaillant sur $GF(2)$: soit un symbole intervient dans une équation, soit il n'intervient pas. Travailler sur un corps fini de dimension supérieure, par exemple $GF(2^8)$, augmente considérablement la probabilité d'avoir un système non singulier car apparaît un coefficient multiplicateur (sur $GF(2^8)$ dans notre exemple) pour chaque symbole. Les capacités de correction sont donc largement améliorées. En revanche, ceci se fait au détriment de la complexité algorithmique puisque chaque opération sur les symboles requiert d'effectuer ces opérations arithmétiques sur le corps fini.

  Nos travaux, dans le contexte des codes LDPC-Staircase, se limitent à $GF(2)$. Nous verrons au chapitre 7 comment étendre ces codes pour intégrer intelligemment des opérations sur $GF(2^8)$ au moyen d'une approche GLDPC. Toute la difficulté consiste alors à introduire des opérations sur $GF(2^8)$ de telle façon que la très vaste majorité des opérations arithmétiques restent sur $GF(2)$ afin de limiter les impacts en terme de complexité algorithmique. [3]

Le nombre important de paramètres à considérer, en d'autre termes la richesse du champ du possible, rend l'étude des performances des codes complexe. Pour ce qui concerne le dernier paramètre, le jeu de symboles disponibles, nous considérerons en général une permutation aléatoire des symboles avant transmission. Ceci offre également l'avantage de ne pas avoir à considérer de modèle de pertes précis : seul le taux de pertes intervient alors, le fait qu'il s'agisse de pertes IID ou en rafale n'ayant pas d'impact puisque le jeu de symboles disponible est dans tous les cas un sous ensemble aléatoire des symboles d'encodage. L'étude portera alors uniquement sur les autres paramètres.

[3] L'évolution des codes Raptor, qui opèrent sur $GF(2)$, vers les codes RaptorQ, qui opèrent en partie sur $GF(2^8)$, relève de la même volonté d'améliorer significativement les capacités de correction des codes.

5.1.1 *Le cas des codes binaires aléatoires*

Les codes binaires aléatoires (de densité 1/2 donc) ont d'excellentes performances de correction. Plus précisément Studholme et Blake [7][96][97], s'appuyant sur les travaux de Kolchin et de mathématiciens Russes, rappellent que :

- la probabilité qu'un code binaire aléatoire puisse décoder avec un overhead de 0 (réception d'exactement k symboles) est de 0.28879;
- l'overhead moyen d'un code binaire aléatoire est de 1.60669 symboles;

Les auteurs notent que ces résultats tiennent pour de faibles tailles de matrices ($k = 10$ symboles) et pas seulement assymptotiquement, ce qui est remarquable. Ainsi :

Avec un code binaire aléatoire, recevoir 2 symboles en plus de k permet d'avoir un décodage qui réussit plus d'une fois sur deux, indépendamment de k, alors que l'on a une probabilité un peu inférieure à 1/3 de réussir le décodage après avoir reçu exactement k symboles.

Ensuite les auteurs notent (sans le démontrer, seule une conjecture est posée) que ces résultats tiennent également pour des codes aléatoires où la probabilité d'avoir un "1" est comprise entre 1/2 (comme ci-dessus) et $2\frac{ln(k)}{k}$ [4].

Les codes LDPC en général, et LDPC-Staircase en particulier, étant générés en grande partie de façon aléatoire, peuvent donc au mieux se comporter comme ces codes.

[4] La borne inférieure est particulièrement intéressante car en diminuant la densité de la matrice (égale à $2ln(k)$), on diminue également la complexité algorithmique en encodage/décodage.

5.2 *Résultats de performance*

Nous présentons dans cette section les résultats obtenus au moyen des outils de simulation disponibles dans le projet OpenFEC (chapitre 12.4.3). Le détail des résultats est donné en annexe 12.3.4 et l'on y trouvera les courbes de probabilité d'erreur de décodage dans leur totalité. Nous nous contentons ici de présenter les résultats principaux.

Nous nous plaçons également dans le cas simple où nous avons un seul symbole par paquet ($G = 1$, voir section 5.1). De meilleurs résultats pour les petits blocs (k faible) pourraient être obtenus en fixant une taille inférieure aux symboles afin d'en augmenter (artificiellement) le nombre, c'est à dire en choisissant une valeur $G > 1$ [5].

[5] Le cas où $G > 1$ peut être estimé en considérant la valeur correspondante de k dans les tables que nous fournissons (par exemple l'overhead moyen pour $k = 400$ sera applicable au cas où les symboles on été redécoupés en $G = 4$ symboles afin de passer de 100 à $k = 400$.

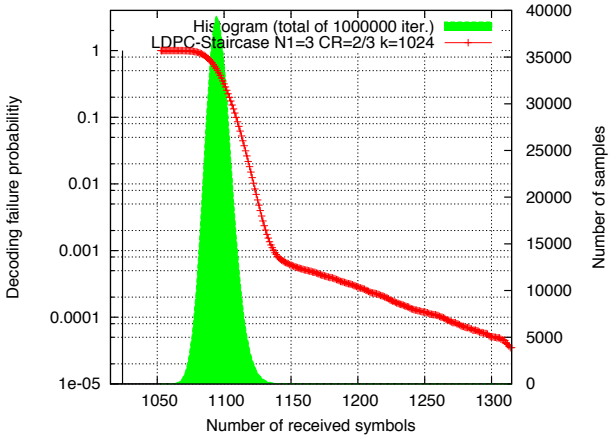
Il est à noter que chaque test (en particulier dans l'évaluation des probabilités d'échec de décodage) correspond à la fois à un code différent et à un séquencement de paquets différent. A l'inverse nous aurions pu figer un code en optant pour un ensemble $\{n, k, N_1, graine\ du\ PRNG\}$ donné, et évaluer les performance de ce code. Ce que nous montrons ici doit plutôt être vu comme des performances moyennes sur l'ensemble des codes possibles pour un triplet $\{n, k, N_1\}$ donné.

5.2.1 De l'importance du décodage ML

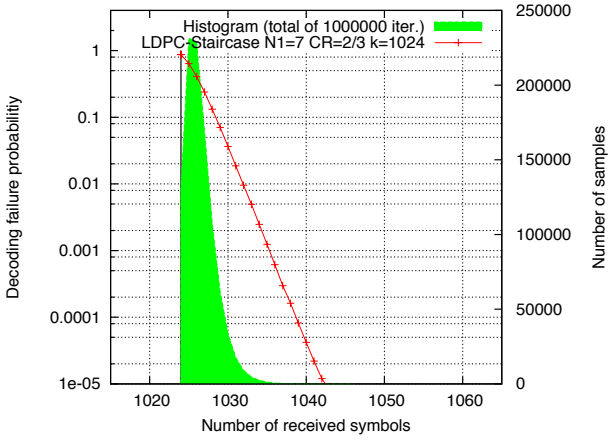
Le décodage IT (section 4.6) est intéressant en terme de faible complexité algorithmique de décodage, mais obtenir des probabilités d'échec de décodage faibles avec ce type de décodeur relève de l'impossible avec les codes LDPC-Staircase[6]. La courbe 5.1-(a), obtenue pour $N_1 = 3$ (valeur optimale en IT), montre un pied de courbe qui apparaît dès 10^{-3}. Passer en dessous d'une probabilité d'erreur de décodage de 10^{-4} requiert alors un overhead de 240 symboles, soit 23,438%.

A l'inverse, en décodage ML (section 4.6) les performances sont proches de l'idéal ainsi que le montre la figure 5.1-(b), où $N_1 = 5$. L'overhead nécessaire pour passer en dessous d'une probabilité d'erreur de décodage de 10^{-4} descend alors à seulement 1,465% lorsque $N_1 = 7$ (cf. table 5.1).

[6] Rappelons qu'utiliser des codes irréguliers, avec une distribution de degrés gauche/droite optimisées pour l'IT, limiterait ce problème (par ex., les codes IRA, section 4.4.1). Ce n'est pas notre choix (section 4.7).



(a) décodage IT



(b) décodage ML

Figure 5.1: Pr_echec = f(overhead) en IT (avec $N_1 = 3$) et ML (avec $N_1 = 5$) des codes LDPC-Staircase avec ($k = 1024$, $CR = 2/3$ et $G = 1$).

→ *Attention : les échelles horizontales des figures 5.1-(a) (300 symboles sur l'axe des abscisses) et (b) (50 symboles) diffèrent.*

5.2.2 *Overhead et probabilités d'échec de décodage*

Regardons maintenant dans le détail les performances en décodage ML Les overheads moyens et afin d'obtenir une probabilité d'échec de décodage inférieure à 10^{-4} sont résumés dans la table 5.1 et illustrés dans la figure 5.2 lorsque $G = 1$.

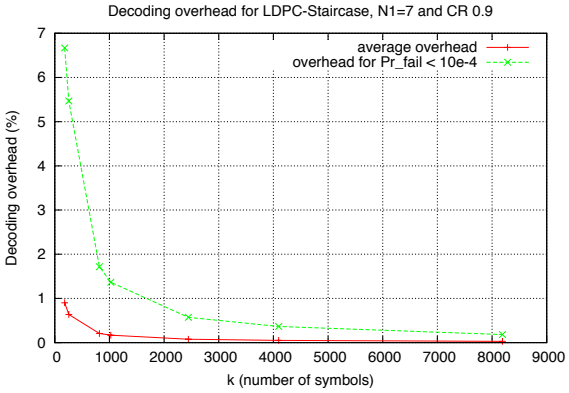
Paramètres	***Overhead moyen*** ($Pr_{fail} = 0.5$)	***Overhead pour*** $Pr_{fail} \leq 10^{-4}$
CR = 0.9		
$k = 180$, $N_1 = 7$	0.902%, soit 1.623 symboles suppl.	6.667%, soit 12 symboles suppl., $Pr_{fail} = 3.8 * 10^{-5}$
$k = 256$, $N_1 = 7$	0.638%, soit 1.633 symboles suppl.	5.469%, soit 14 symboles suppl., $Pr_{fail} = 6.2 * 10^{-5}$
$k = 256$, $N_1 = 9$	0.629%, soit 1.610 symboles suppl.	5.469%, soit 14 symboles suppl., $Pr_{fail} = 6.6 * 10^{-5}$
$k = 1024$, $N_1 = 5$	0.245%, soit 2.509 symboles suppl.	1.758%, soit 18 symboles suppl., $Pr_{fail} = 9.5 * 10^{-5}$
$k = 1024$, $N_1 = 7$	0.168%, soit 1.720 symboles suppl.	1.367%, soit 14 symboles suppl., $Pr_{fail} = 7.9 * 10^{-5}$
$k = 4096$, $N_1 = 5$	0.129%, soit 5.284 symboles suppl.	0.488%, soit 20 symboles suppl., $Pr_{fail} = 6.8 * 10^{-5}$
$k = 4096$, $N_1 = 7$	0.051%, soit 2.089 symboles suppl.	0.366%, soit 15 symboles suppl., $Pr_{fail} = 6.4 * 10^{-5}$
$k = 8192$, $N_1 = 5$	0.099%, soit 8.180 symboles suppl.	0,293%, soit 24 symboles suppl., $Pr_{fail} = 7.1 * 10^{-5}$
$k = 8192$, $N_1 = 7$	0.030%, soit 2.474 symboles suppl.	0,183%, soit 15 symboles suppl., $Pr_{fail} = 8.4 * 10^{-5}$
CR = 2/3		
$k = 180$, $N_1 = 7$	0.971%, soit 1.748 symboles suppl.	8.333%, soit 15 symboles suppl., $Pr_{fail} = 7.6 * 10^{-5}$
$k = 256$, $N_1 = 7$	0.706%, soit 1.807 symboles suppl.	5.859%, soit 15 symboles suppl., $Pr_{fail} = 5.9 * 10^{-5}$
$k = 256$, $N_1 = 9$	0.642%, soit 1.644 symboles suppl.	5.469%, soit 14 symboles suppl., $Pr_{fail} = 8.9 * 10^{-5}$
$k = 1024$, $N_1 = 5$	0.636%, soit 6.513 symboles suppl.	2.148%, soit 22 symboles suppl., $Pr_{fail} = 5.9 * 10^{-5}$
$k = 1024$, $N_1 = 7$	0.238%, soit 2.437 symboles suppl.	1.465%, soit 15 symboles suppl., $Pr_{fail} = 8.2 * 10^{-5}$
$k = 4096$, $N_1 = 5$	0.516%, soit 21.135 symboles suppl.	1.050%, soit 43 symboles suppl., $Pr_{fail} = 7.9 * 10^{-5}$
$k = 4096$, $N_1 = 7$	0.120%, soit 4.915 symboles suppl.	0.464%, soit 19 symboles suppl., $Pr_{fail} = 7.1 * 10^{-5}$
$k = 8192$, $N_1 = 5$	0.496%, soit 40.632 symboles suppl.	0,842%, soit 69 symboles suppl., $Pr_{fail} = 7,5 * 10^{-5}$
$k = 8192$, $N_1 = 7$	0.101%, soit 8.258 symboles suppl.	0,281%, soit 23 symboles suppl., $Pr_{fail} = 9,3 * 10^{-5}$
CR = 1/2		
$k = 180$, $N_1 = 7$	1.120%, soit 2.016 symboles suppl.	8.889%, soit 16 symboles suppl., $Pr_{fail} = 7.7 * 10^{-5}$
$k = 256$, $N_1 = 7$	0.852%, soit 2.181 symboles suppl.	5.860%, soit 15 symboles suppl., $Pr_{fail} = 8.9 * 10^{-5}$
$k = 256$, $N_1 = 9$	0.677%, soit 1.733 symboles suppl.	5.477%, soit 14 symboles suppl., $Pr_{fail} = 7.1 * 10^{-5}$
$k = 1024$, $N_1 = 5$	1.228%, soit 12.575 symboles suppl.	3.125%, soit 32 symboles suppl., $Pr_{fail} = 7.2 * 10^{-5}$
$k = 1024$, $N_1 = 7$	0.378%, soit 3.871 symboles suppl.	1.660%, soit 17 symboles suppl., $Pr_{fail} = 9.6 * 10^{-5}$
$k = 4096$, $N_1 = 5$	1.690%, soit 69.222 symboles suppl.	2.637%, soit 108 symboles suppl., $Pr_{fail} = 9.4 * 10^{-5}$
$k = 4096$, $N_1 = 7$	0.463%, soit 18.964 symboles suppl.	0.977%, soit 40 symboles suppl., $Pr_{fail} = 8.5 * 10^{-5}$

Table 5.1: Capacités de correction des codes LDPC-Staircase en décodage ML, en fonction de $\{CR, k, N_1\}$, avec $G = 1$.

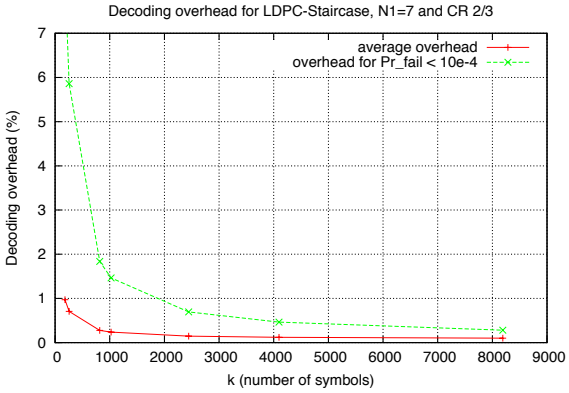
5.2.3 *Discussion*

Ces résultats montrent qu'il est possible d'atteindre assez facilement des overheads moyens très faibles, significativement en deçà de 1%, pour les valeurs usuelles de k et des code rates supérieurs ou égaux à 2/3, lorsque N_1 vaut 5. En revanche, réduire le pied de courbe, c'est à dire réduire la probabilité d'avoir un échec de décodage pour un overhead donné, peut nécessiter d'augmenter la valeur de N_1. Le choix $N_1 = 7$ est ainsi recommandé si la métrique "overhead pour une probabilité d'échec de décodage inférieure à 10^{-4} est le critère premier[7]. Nous voyons figure 5.2 (limitée au cas $N_1 = 7$) que l'overhead nécessaire pour cet objectif de décodage est la plupart du temps inférieur à 1%, ce qui est excellent.

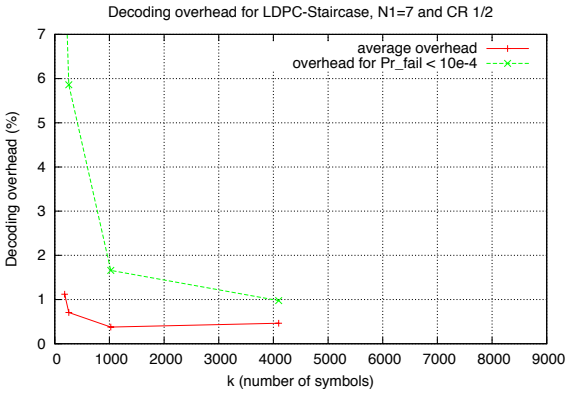
[7] La valeur 10^{-4} correspond à un échec qui se produit en moyenne une fois sur 10 000 décodages. Est-ce élevé ? Tout dépend du nombre de décodages effectués par heure, c'est à dire du champ d'application... Dans beaucoup de cas on fixera un objectif de nombre d'échecs de décodage par heure, et les conditions de transmission permettront de fixer l'objectif de borne maximale de la probabilité d'échec de décodage, et non l'inverse.



(a) $CR = 0.9$



(b) $CR = 2/3$



(c) $CR = 1/2$

Figure 5.2: Capacités de correction des codes LDPC-Staircase en décodage ML, en fonction de CR, avec $N_1 = 7$ et $G = 1$.

Si augmenter N_1 est un excellent moyen d'améliorer les capacités de correction en décodage ML, en revanche cela va aussi :

- dégrader le décodage IT préliminaire, ce qui augmentera la complexité du système résultant donné au décodeur ML;
- augmenter les complexités d'encodage et décodage, et en particulier le nombre d'opérations XOR;

Cependant nous montrerons au chapitre 6 qu'un codec convenablement implémenté supporte aisément ce surcroît de complexité. En pratique on utilisera **la valeur de N_1 la plus faible qui réponde aux besoins de performances de décodage du champ d'application cible**.

5.3 *Comparaison avec des codes Reed-Solomon sur $GF(2^8)$*

Les codes Reed-Solomon sont des codes MDS et de ce fait ont des performances de décodage idéales. Cependant des contraintes de complexité algorithmique et pratiques limitent le type de corps fini sur

lequel ils travaillent à $GF(2^8)$ en général. Ceci induit une limite directe sur les paramètres dimension et longueur : $k < n \leq 255$.

L'aspect pratique est essentiel : avec $GF(2^8)$, chaque élément du corps est codé sur 8 bits. Les accès mémoire se font donc sur la base des octets, ce qui est aisé. Par contre, avec $GF(2^4)$, une valeur encore utilisable, un élément est stocké sur 4 bits, et chaque octet d'un symbole contient deux éléments. Un codec optimisé tachera donc d'effectuer les traitement sur deux éléments à la fois (c'est ce que nous faisons dans le codec Reed-Solomon (RS) du projet OpenFEC.org, ce qui permet à ce codec de disposer de vitesses d'encodage/décodage légèrement supérieures à celle de la version $GF(2^8)$. La différence est cependant faible, et sans cette optimisation il devient nettement plus lent. Utiliser les corps finis avec des tailles d'éléments autres rend les opérations encore plus complexes.

Ainsi qu'il est expliqué dans la section 3.2.1 ("problème du collecteur de coupons"), il devient nécessaire de découper tout objet volumineux en plusieurs blocs d'encodage ce qui limite les capacités de correction atteignables. Ce phénomène est d'ailleurs d'autant plus important que l'objet est important[8].

[8] Ces résultats sont valables avec tout code MDS dont les dimensions et longueurs maximum sont fortement limitées.

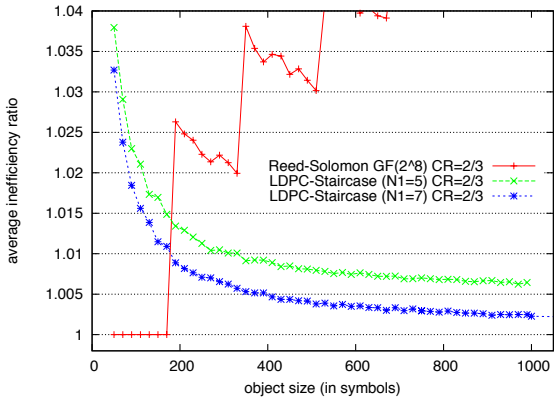


Figure 5.3: Capacités de correction des codes LDPC-Staircase (ML) et Reed-Solomon, avec ($k = 1024$, $CR = 2/3$, $G = 1$).

La figure 5.3 illustre ce phénomène. Nous voyons clairement la supériorité des codes Reed-Solomon pour les petits blocs. Puis, au delà de $k = 170$ symboles[9], un découpage de l'objet en deux blocs a lieu, puis trois, etc., dégradant dès lors de plus en plus les capacités de correction. Le même phénomène se produit à chaque multiple de 170. A l'inverse, les codes LDPC-Staircase deviennent rapidement plus efficaces que les codes Reed-Solomon (dès une taille d'objet de 200 symboles), puis la tendance s'amplifie.

[9] Il est à noter que $k = 170$ est la limite supérieure avec $CR = 2/3$, puisque $170 * 2/3 = 255$, valeur maximale sur le corps fini $GF(2^8)$.

Ces simples tests nous montrent que le critère usage a autant d'importance que le code à proprement parler, et que des codes non MDS peuvent aisément se révéler plus performants en termes de capacités de correction d'effacements que des codes dotés de performances pourtant idéales.

Vautour fauve dans la brume, Haute Savoie, 2011.

6
Performances des codes LDPC-Staircase : complexité/vitesse et besoins mémoire

Résumé :
Ce chapitre est consacré aux performances des codes LDPC-Staircase du point de vue de sa complexité algorithmique, de sa vitesse (qui découle pour partie de la complexité algorithmique), et des besoins mémoire crêtes, que ce soit à l'encodage ou au décodage. Ces travaux reposent sur d'importants développements logiciels, initiés en 2003, et à l'optimisation continuelle des codecs logiciels qui en sont le fruit, certains étant distribués au sein du projet OpenFEC.org.

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux communs menés avec C. Neumann, M. Cunche, J. Detchart, F. Mattoussi, V. Savin.*

6.1 Introduction

En plus de capacités de correction d'effacements des codes AL-FEC, leurs performances en termes de complexité algorithmique, vitesse, besoins mémoire crêtes peuvent être décisifs dans le choix d'une solution de protection, pour un champ d'application donné. Ces métriques de performance, objet de ce chapitre, doivent être évaluées à la fois à l'encodage et au décodage. Actuellement, les services de type 3GPP-MBMS (et leurs équivalents), gros utilisateur de codes AL-FEC [45], se focalisent sur des terminaux mobiles (smartphones) qui reçoivent les flux d'information et effectuent le décodage AL-FEC. Ainsi les codes Raptor/RaptorQ [91] ont été optimisés afin de privilégier le décodeur, au détriment de l'encodeur[1]. Nous nous intéresserons pour notre part aux deux aspects, encodage et décodage, puisque d'autres champs d'applications nécessitent qu'un équipement embarqué soit à la source de flux protégés par un encodage AL-FEC. On pourra citer des caméras de surveillance, ou des robots d'exploration, des sondes spatiales, ou même des modèles de diffusion de type pair à pair sur smartphone.

Nous accorderons une grande importance aux performances obte-

[1] Ainsi effectuer un encodage Raptor/RaptorQ nécessite de résoudre un système linéaire afin de produire les symboles intermédiaires, là où les codes LDPC-Staircase vont directement générer les symboles d'encodage. Ceci se traduit par un surcroît de complexité inévitable.

nues sur un smartphone représentatif de 2012, le Samsung Galaxy SII (GT-I9100P), sous un système Android 2.3.4. Nous compléterons avec les performances obtenues sur une ancienne machine de bureau.

6.2 *Et si on parlait de l'algorithme de décodage ?*

Les performances que nous étudions ici sont intrinsèquement liées aux algorithmes de décodage utilisés. Nous avons déjà introduit les notions de décodage IT et ML dans la section 4.6, ainsi que la possibilité de débuter en IT et finir en ML si nécessaire. Nous allons plus loin dans cette section en introduisant un raffinement du décodage IT ainsi que la technique SGE utilisée pour le ML.

6.2.1 *Le décodeur itératif avec élagage*

Le décodeur itératif est un décodeur rapide, de complexité linéaire (voir la section 4.6). En revanche, lorsqu'il est implémenté de façon classique, il est "gourmand" en cela que chaque symbole susceptible d'être décodé le sera, qu'il s'agisse d'un symbole source ou d'un symbole de redondance. Or un certain nombre de ces décodages seront au final inutiles, puisqu'il ne servira à rien de décoder un symbole de redondance si ce dernier ne permet pas de décoder un symbole source par la suite. Il est donc possible d'optimiser largement la vitesse du décodeur itératif en procédant en deux étapes : dans un premier temps nous créons un graphe représentant la liste des opérations XOR à effectuer sur les symboles, et une fois que nous avons la certitude de pouvoir finir le décodage, nous parcourons cet arbre, retirons toutes les branches ne se trouvant pas sur un chemin aboutissant au décodage d'un symbole source, pour enfin n'effectuer que les opérations XOR utiles. Cette optimisation permet des gains significatifs, que ce soit en termes de vitesse (réduction du nombre de XOR) ou de consommation mémoire (réduction du nombre de buffers contenant les symboles de redondance décodés).

6.2.2 *Le décodeur ML : l'approche SGE*

Deux articles, proches l'un de l'autre, donnent les fondements de l'approche SGE qui est fort bien adaptée à la résolution de systèmes linéaires creux, tels que ceux que l'on trouve lors du décodage d'un code LDPC. Il s'agit des travaux de B. A. LaMacchia et A. M. Odlyzko d'une part [41], et de C. Pomerance et J. W. Smith d'autre part [71].

La technique comporte deux phases. La première est la suivante :

Reduction of Huge, Sparse Matrices over Finite Fields Via Created Catastrophes

Carl Pomerance and J. W. Smith

CONTENTS

We present a heuristic method for the reduction modulo 2 of a large, sparse bit matrix to a smaller, dense bit matrix that can then be solved by conventional means, such as Gaussian elimination. This method worked effectively for us in reducing matrices as large as 100,000 × 100,000 to much smaller, but denser square matrices.

1. INTRODUCTION

The last stage of several algorithms to factor integers, notably the quadratic sieve [Pomerance et al. 1988] and the number field sieve [Buhler et al.], involves the reduction of a huge, sparse bit matrix. (By a *bit matrix* we mean a matrix over the field with two elements: every element is 0 or 1 and arithmetic is modulo 2.) Although the matrices involved can be truly huge—100,000 rows and columns is no longer unusual—they may have as few as twenty 1's per row. Another feature of these matrices is that the first few columns tend to be denser than the rest: typically, about half of all of the 1's appear in the first $\sqrt{M}$ columns, out of a total of M columns.

In this paper we describe a heuristic method for obtaining linear dependencies among the rows of such matrices. In addition, we present data from experimental runs with randomly generated square matrices of size 50,000 and 100,000.

Odlyzko [1985] introduced a simple, two-step algorithm for dealing with these matrices. First, a certain fraction of the columns (the heaviest ones, which are mostly on the left side of the matrix) are declared "inactive", and are temporarily removed from the sparse data structure holding the matrix. What is left is a very sparse matrix with fewer columns than rows. Odlyzko then follows the *Markowitz rule* for row reduction in the active

The authors' research was supported in part by NSF grants.

0316-6450/92 $0.50 per page

Soit le système constitué des équations du code LDPC-Staircase,

```
dans lequel tous les symboles connus sont sommées dans les
termes constants. Ces symboles connus sont ceux qui ont été
reçus ainsi que ceux qui ont été décodés dans la phase de
décodage IT). Toutes les colonnes de la matrice sont déclarées
"actives" dans un premier temps (le vecteur des termes constants
est considéré à part). Le poids d'une ligne de la matrice est
définie comme étant le nombre de colonnes actives intervenant
dans cette ligne. De même, le poids d'une colonne est défini
comme étant le nombre de lignes faisant intervenir le symbole
associé à la colonne.

while (le nombre de colonnes actives > 0) {

    Etape 0 :
      On déclare inactives certaines colonnes de la matrice,
      choisies convenablement (plusieurs approches sont possibles,
      les auteurs suggérant de sélectionner un faible pourcentage
      des colonnes de poids maximum).
      Ces colonnes (et les symboles correspondants) forment un sous
      système qui sera décodé ultérieurement. Pour l'instant nous
      faisons comme si la valeur de ces symboles était connue.
      Cette étape a pour effet de diminuer le poids de chaque ligne
      où intervenaient ces symboles.

    Etapes 1, 2 et 3 : inutiles dans notre cas.

    Etape 4 : /* c'est en fait un décodage IT... */
    while (il existe une ligne de poids 1) {

        Eliminer la ligne de poids 1;
        Eliminer la seule colonne active correspondante puisque ce
        symbole sera connu une fois que tous les symboles inactivés
        de cette ligne seront connus (phase 2);
        Eliminer également ce symbole de toutes les lignes où il
        intervient, en mémorisant que ce symbole devra être ajouté
        au terme constant une fois connu (phase 2);
    }
}
```

A l'issue de cette phase, nous avons un deuxième système constitué des colonnes inactivées et des lignes correspondantes.

La seconde phase consiste alors à décoder avec une approche Pivot de Gauss classique ce sous système. La valeur des symboles inactivés dans la première phase est donc trouvée, en d'autre termes ces symboles sont décodés.

Enfin on réintègre les valeurs de ces symboles décodés dans la seconde phase dans le terme constant des équations où ils ont été éliminés. On s'arrêtera lorsque la totalité des symboles source effacés sont

Figure 6.1: Brian A. LaMacchia (http://www.farcaster.com/)

Figure 6.2: Carl Pomerance (http://www.math.dartmouth.edu/~carlp/)

connus (reçus ou décodés).

6.2.3 *Qu'en est-il d'un ML par pivot de Gauss sur la matrice H_{id} ?*

Classiquement un décodage ML se fait en partant de la matrice de parité "simplifiée", où l'on a sommé tous les symboles connus dans les termes constants des équations. Dès lors les variables du système sont les symboles source et redondance effacés. Une alternative existe qui permet de réduire le nombre d'inconnues : il s'agit de travailler non pas sur la matrice de parité mais sur la matrice H_{id} (section 4.3), en ne conservant que les équations de H_{id} concernant les symboles de redondance connus. Le passage de l'un à l'autre est trivial, du fait de la structure escalier, et les capacités de correction rigoureusement identiques. Moyennant quoi le système à résoudre par pivot de Gauss peut inclure un nombre significativement moindre de variables (ce qui est surtout vrai dans le cas d'un code rate faible). En revanche le système devient plus dense (voir les figures 4.6). La question est donc de savoir si travailler sur H_{id} est bénéfique ou non.

6.2.4 *Quid de la décomposition LU ? Est-ce adapté ?*

Soit une matrice A de taille $N \times N$ pouvant être décomposée en : $A = LU$, où L ("Lower") et U ("Upper") sont respectivement triangulaires inférieure et supérieure. Alors la résolution du système $Ax = b$, où x est le vecteur inconnu, peut également se faire de la façon suivante ([3], chap. 2.3) :

$$A.x = (L.U).x = L.(U.x) = b$$

L'idée est alors de résoudre dans un premier temps :

$$L.y = b$$

ce qui se fait aisément avec des substitutions en avant (même technique que pour l'encodage d'un code LDPC-Staircase). Une fois trouvé le vecteur y, il reste à résoudre :

$$U.x = y$$

ce qui se fait là aussi aisément au moyen de substitutions arrières (même technique que pour la deuxième phase de décodage ML). Ces deux étapes sont rapides, surtout dans le cas de matrices binaires : elles conduisent à un total de $2 * (N^2/2) = N^2$ opérations XOR sur des symboles, sans nécessiter de manipulation autre sur les matrices. Cependant la création de la décomposition LU a un coût : $N^3/3$, de complexité comparable à une élimination de Gauss.

Dès lors il ressort que faire une décomposition LU préalable ne peut apporter de bénéfices que lorsque la matrice A est fixe et utilisée pour

plusieurs jeux de vecteurs b. Dans notre cas, le vecteur b est constitué de la somme XOR des symboles d'encodage reçus de chaque équation. Cette somme dépend donc du canal, même si le code est fixé. La matrice A est constituée du sous ensemble de H correspondant aux symboles inconnus après réception. Cette matrice dépend également du canal, même si le code est fixé. Il ne s'agit donc pas d'une solution à notre problème.

6.3 *Et tout le reste...*

L'algorithme de décodage est essentiel dans l'obtention de vitesses élevées. Il ne faut cependant pas négliger le reste, et en particulier :

- les opérations XOR entre symboles;
- la représentation des matrices;

6.3.1 *Les opérations XOR entre symboles*

Les opérations XOR peuvent se faire sur différentes tailles de données : bit à bit, octet par octet, sur des mots de 32-bits, 64-bits, ou sur des vecteurs de 128-bits avec certaines extensions du jeu d'instruction de type Single Instruction Multiple Data (SIMD) (par ex. Intel/SSE ou ARM/NEON). Sommer deux symboles de plusieurs centaines d'octets devient alors efficace, puisque travailler sur 128-bits au lieu d'un octet divise par 16 le nombre d'opérations XOR effectuées. Il s'agira cependant de vérifier le bien fondé de ces optimisations, les détails d'architecture du processeur cible pouvant aller à l'encontre de ce principe général, en particulier pour les accès mémoire.

Exemple de routine assembleur ARM/NEON de XOR de deux symboles :

Nos tests pour une cible ARM Cortex-A9, disposant de l'extension NEON pour les traitements SIMD, ont montré que l'optimum est obtenu de la façon suivante (figure 6.3).
Cette routine effectue des transferts de données depuis la mémoire vers deux "quad-registers" (Q) de taille 16 octets chacun, soit par blocs de 32 octets (256-bits) (`VLD1.64`*). A noter que le deuxième* `VLD1.64` *effectue une post-incrémentation de* `R1` *du nombre d'octets lu (*`!` *final). Ensuite chaque registre du premier symbole est XORé avec le registre correspondant du deuxième symbole au moyen de* `VEOR`*. Ces XOR sont donc effectués par bloc de 128 bits. Enfin le résultat contenu dans (Q0,Q1) est stocké en mémoire avec le* `VST1.64` *final, le registre* `R0` *étant post-incrémenté à son tour.*
A noter que la lecture/écriture de blocs de huit registres Q est possible via les instructions `VLDM/VSTM`*, soit pour un bloc de taille 128 octets (1024 bits).*

```
# Effectue to = to ^ from;
# r0 contient l'adresse du symbole to
# r1 contient l'adresse du symbole from
# r2 contient le nombre de mots de 32 octets
# (la taille de symboles est multiple de 32)
Loop:
    PLD             [R0,#64]
    PLD             [R1,#64]
    VLD1.64         {Q0,Q1}, [R0]
    VLD1.64         {Q2,Q3}, [R1]!
    VEOR            Q0,Q2
    VEOR            Q1,Q3
    SUBS            R2,R2,#1
    VST1.64         {Q0,Q1}, [R0]!
    BGT             Loop
```

Figure 6.3: Routine en assembleur ARM/NEON effectuant : to = to ^ from;

Cependant les performances sont alors en retrait, ceci étant probablement lié au fait que l'unité de calcul (utilisée pour le XOR) est bloquée en attente de la fin des accès mémoire (lors des `VLD` *et* `VST`*).*

La routine ci-dessus travaille avec deux symboles. Lorsque le même symbole doit être ajouté à plusieurs symboles destination, on aura intérêt à utiliser une routine dédiée qui ne lit ce symbole qu'une seule fois, et XOR ces données alors dans les registres du processeur à tous les symboles destination. La routine réciproque, où l'on ajoute plusieurs symboles source au même symbole destination, sera également d'un grand secours.

6.3.2 *La représentation des matrices*

La représentation en mémoire des matrices (ou, ce qui est équivalent, du système linéaire) est capitale. Nous utilisons des variantes des structures `mod2sparse` et `mod2dense` du simulateur LDPC de R. Neal (section 4.8). La représentation éparse stocke chaque entrée non nulle de la matrice binaire au sein d'une structure qui donne un accès direct, via des pointeurs gauche/droite/haut/bas, aux entrées non nulles adjacentes de la matrice[2]. La représentation dense stocke pour sa part chaque ligne de la matrice au sein d'un vecteur binaire, les bits à "0" ou "1" correspondant aux entrées nulles ou non de la matrice.

Chaque représentation est utile et la vitesse maximale sera obtenue :

- avec une représentation creuse de la matrice lorsque l'algorithmique nécessite de fréquemment chercher la prochaine entrée non nulle d'une ligne ou d'une colonne. C'est typiquement ce qui se passe lors du décodage IT;
- avec une représentation dense de la matrice lorsque l'algorithmique nécessite de fréquemment sommer deux lignes. C'est typiquement ce qui se passe lors du décodage par pivot de Gauss, lorsque l'on ajoute la ligne du pivot à toutes les autres lignes du dessous où apparaît la variable servant de pivot;

Le décodage **démarre donc par une représentation creuse** de la matrice, utilisée pour le décodage IT et la première phase du décodage SGE. Puis une **bascule vers la représentation dense** est faite pour décoder par élimination de Gauss le sous système de l'approche SGE.

[2] D'autres représentations de matrices creuses existent. C'est le cas de l'approche "Indexed Storage" ([3], chap. 2.7), assez peu intuitive. Cependant cette approche est optimisée au problème de multiplications matrice-vecteurs, ce qui est assez éloignée de nos objectifs. De plus elle est utilisée pour des matrices quelconques, et non pour des matrices binaires qui permettent d'autres optimisations.

6.4 *Vitesses de décodage et encodage*

Nous donnons maintenant une idée des vitesses obtenues en décodage, sur différentes machines cibles. Nous retenons la vitesse comme indi-

cateur principal de complexité et non le nombre d'opérations sur les symboles (opérations XOR entre symboles avec un code binaire)[3].

6.4.1 *Vitesse comparée entre différentes techniques de décodage*

Tout d'abord, afin de répondre à la question de la section 6.2.3, nous avons implémenté et optimisé les différents algorithmes de décodages. Sur un MacBook Pro, Core i7, 2.2GHz, $k = 8442$, $CR = 2/3$, $N_1 = 7$, des symboles de taille 1288 octets, et à la limite de décodage ML (overhead de 9 symboles), nous observons alors :

- décodage pivot de Gauss sur H : débit **251 Mbps**
- décodage pivot de Gauss sur H_{id} : débit **316 Mbps**
- décodage SGE sur H : débit **1055 Mbps**[4]

La différence est nette et l'approche SGE est clairement la solution à retenir, apportant un **gain d'un facteur 4 en vitesse de décodage** sur un pivot de Gauss classique. Nous ne considérerons donc que SGE dans la suite.

6.4.2 *Conditions de test*

Ces mesures utilisent le codec LDPC-Staircase avancé (v2.2.0, début décembre 2012). Nous prenons $CR = 2/3$, $N_1 = 7$, des symboles de 1024 octets et $G = 1$. Nous considérons deux situations :

- bon canal, avec un taux de pertes de 5% seulement;
- mauvais canal, à la limite des capacités de corrections de nos codes, sachant qu'un code idéal peut décoder jusqu'à 33,33% dans ce cas. (i.e. perdre un symbole de plus empêche tout décodage);

Nous considérons une machine de bureau ancienne (achat fin 2006) dotée d'un processeur Intel Xeon 5120/1.86GHz et sous Linux/64 bits. La figure 6.4-(a) montre les vitesses obtenues, suivant que l'on effectue un décodage IT (bon canal) ou ML (mauvais canal, à la limite des capacités de correction du code).

Nous considérons également un smartphone représentatif, puisqu'un nombre important de champs d'applications visent des terminaux mobiles (section 6.5). Nous avons donc réalisé des tests sur le Samsung Galaxy SII (GT-I9100P), sous Android 2.3.4. Ce terminal est doté d'un processeur ARM Cortex-A9, double coeur, cadencé à 1,2 GHz, et 1 GO de RAM. Dans nos tests, un seul coeur est utilisé à 100% (le deuxième est ignoré), le téléphone étant à sa fréquence nominale de 1,2 GHz.

[3] Nous avons pu en effet constater que diminuer ce nombre d'opérations ne garantit pas nécessairement un gain en vitesse. Ce fut le cas lorsqu'une optimisation sur la phase de "backward substitution" lors du pivot de Gauss nous a permis à réduire de 71000 le nombre de XOR (sur un total de 680000), alors que la vitesse de décodage était en retrait. Une deuxième optimisation, moins importante en nombre d'opérations XOR ("seulement" 44000 opérations évitées), s'est avérée plus intéressante et a été retenue. Ce comportement est typiquement lié à des aspects tels que la complexité algorithmique requise pour aboutir à cette réduction, ou bien le comportement des différents niveaux de caches mémoire, qui sont ignorés si l'on se contente de comptabiliser les XOR.

[4] ce résultat inclue d'autres optimisations que le simple passage à SGE. Cependant les gains viennent essentiellement de l'approche SGE.

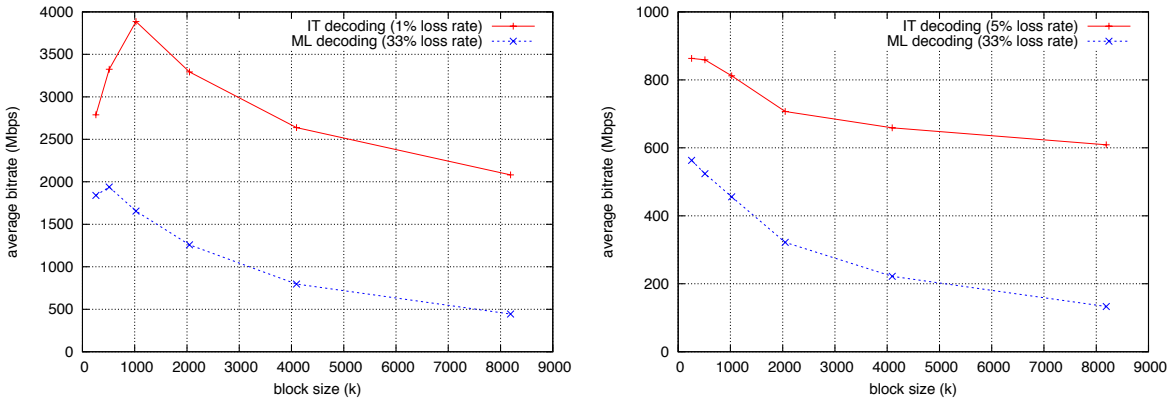
Le mode opératoire dans le cas des tests smartphone/Android consiste à travailler en root sur le terminal, via un shell déporté, et à passer le terminal en mode "performance" via la commande :

```
echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_governor
```

Nous vérifions ensuite que la fréquence d'horloge est bien de 1,2 GHz :

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling\_cur\_freq
```

6.4.3 Vitesse de décodage



(a) machine de bureau

(b) Samsung Galaxy SII

Figure 6.4: Vitesse du codec LDPC-Staircase, en décodage IT (bon canal) et ML (mauvais canal) ($CR = 2/3$, $N_1 = 7$, symboles de 1024 octets, et $G = 1$), sur une machine de bureau et un smartphone.

Les résultats des tests sont donnés dans les figures 6.4.

Ces résultats montrent que les codes LDPC-Staircase sont très rapides, y compris sur des terminaux légers, et en tous cas bien au dessus des débits de réception pratiques. Un corollaire est que la charge CPU induite par le décodage de ces codes est faible.

6.4.4 Vitesse d'encodage

Nous nous limitons ici au cas du smartphone. Les vitesses sont excellentes et stables avec k. Encoder n'est donc jamais un soucis, y compris pour les grandes valeurs de k.

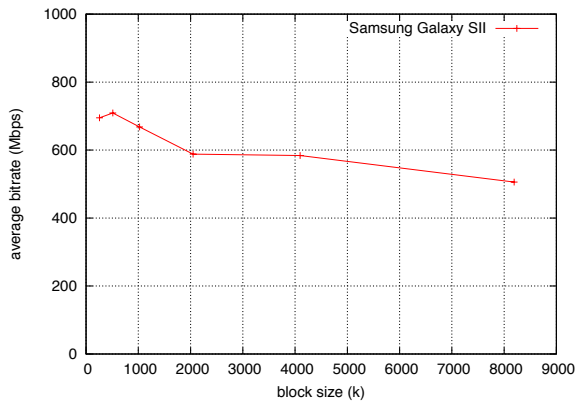


Figure 6.5: Vitesse d'encodage en fonction de k, Samsung Galaxy SII ($CR = 2/3$, $N_1 = 7$, symboles de 1024 octets, et $G = 1$).

6.5 *Performances comparées obtenues durant la compétition 3GPP-eMBMS*

Le groupe SA4 du 3GPP de fin 2011 à fin janvier 2013 a mis en place une compétition en vue de sélectionner les futurs codes AL-FEC de la pile de communication des smartphones. De nombreux scénarios ont pour cela été testés, que ce soit pour des transmissions de type "téléchargement de fichiers" (en mode diffusion avec FLUTE/ALC), ou pour des transmissions de type "streaming" (en mode DASH dans lequel le flux vidéo est découpé en tronçons de quelques secondes, transmis en séquence là encore au moyen de FLUTE/ALC).

→ **Important :** *Ces scénarios ont été sélectionnés pour être* **représentatifs des usages effectifs des services MBMS**. *En revanche ils ne couvrent pas la totalité des configurations possibles des codes AL-FEC. Ainsi aucun test ne nécessite de descendre significativement en dessous d'un code rate 2/3 ! De plus ce sont des tests in-vivo, où les codes AL-FEC sont testés en situation réelle, au sein de l'application FLUTE/ALC. Ils ne correspondent donc pas au performances pure des codecs AL-FEC mais incluent également des temps de traitement autres. C'est un choix assumé du groupe de travail SA4 du 3GPP. Il n'est donc pas possible de comparer directement les valeurs avec celles de la section 6.4*

Trois propositions étaient en lice :

- notre proposition basée sur l'association Reed-Solomon + LDPC-Staircase, et notée RS+LDPC;
- les codes RaptorQ de Qualcomm, notés 6330 d'après le RFC les définissant [49];
- les codes SuperCharged de Broadcom [94].

Nous ne montrons ici que les résultats des tests "téléchargement" dans lesquels nous utilisons les codes LDPC-Staircase. Trois contenus devaient être transmis :

- une vidéo HD de taille 1,8 GB : nous avons utilisé 174 blocs de taille $k = 8422$ symboles, chacun de taille 1288 octets;
- une vidéo SD de taille 128 MB : nous avons utilisé 13 blocs de taille $k = 8016$ symboles, chacun de taille 1288 octets;
- un clip vidéo de taille 3 MB : nous avons utilisé un bloc de taille $k = 2443$ symboles, chacun de taille 1288 octets;

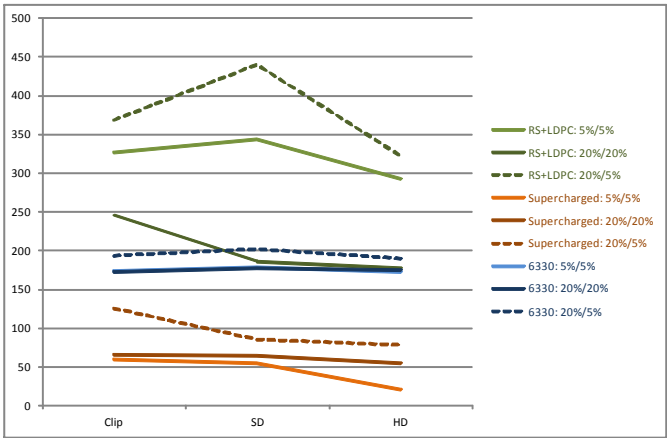


Figure 6.6: Vitesse de décodage (en Mbps) des 3 propositions (la notre, notée RS+LDPC, les codes RaptorQ, notés 6330, et les codes Supercharged), en mode "téléchargement de fichier" (Samsung Galaxy SII). Source Huawei Technologies France, Tdoc S4-130061, TSG SA472 meeting, Janvier 2013.

Speed (MB/s)	RS+LDPC			Supercharged (set 1)			6330 (set **3**)		
	Clip	SD	HD	Clip	SD	HD	Clip	SD	HD
5%/5%	326,8	343,3	292,4	59,5	19,0	21,3	173,8	178,1	172,9
20%/20%	245,9	185,4	177,8	65,7	25,0	55,2	172,4	177,3	174,9
20%/5%	368,5	439,7	322,0	124,9	81,3	77,7	193,6	201,4	189,6
Memory (MB)	4,8-6,2	13,8-18,6	25,3-30,3	8,0-9,3	173,3-192,8	195,7-222,9	6,7-7,0	21,0-21,9	22,4-23,4

Figure 6.7: Détails des performances des 3 propositions (la notre, notée RS+LDPC, les codes RaptorQ, notés 6330, et les codes Supercharged), en mode "téléchargement de fichier" (Samsung Galaxy SII). Source Huawei Technologies France, Tdoc S4-130061, TSG SA472 meeting, Janvier 2013.

Les figures 6.6 et 6.7 sont extraites du document final de Huawei Technologies France [28], seul participant indépendant ayant réalisé des vérifications croisées des trois propositions. Les résultats présentés sont basés sur les dernières versions des logiciels lui ayant été fournies début janvier 2013, avant le vote final.

On voit que, pour les scénarios considérés, représentatifs des usages effectifs des services MBMS :

- les codes LDPC-Staircase sont toujours au moins aussi rapides que les codes RaptorQ;
- les codes LDPC-Staircase sont nettement plus rapides lorsque le code rate est élevé;

- les codes LDPC-Staircase sont nettement plus rapides lorsque l'on a un canal bien meilleur que ce qui est anticipé[5];
- les codes RaptorQ ont pour leur part une vitesse très stable au travers des différents scénarios, du fait de leur conception;

Concernant les consommations mémoire, nous constatons qu'il n'y a pas de différence majeure entre LDPC-Staircase et RaptorQ : certains scénarios sont en faveur de l'un, d'autre en faveur de l'autre. Dans tous les cas, les besoins mémoire de ces deux codecs sont largement contenus, y compris dans le cas le plus exigeant des vidéos HD.

[5] Les services extended Multimedia Broadcast and Multicast Services (eMBMS) sont pour des flux transmis en mode diffusion, destinés à un grand nombre de terminaux qui ont des taux de pertes potentiellement très différents les uns des autres.

Ces résultats confirment que les codes LDPC-Staircase sont très rapides et peu gourmand en mémoire, y compris sur des terminaux légers. Ces performances ont contribué à ce que notre proposition RS+LDPC obtienne un nombre final de votes légèrement supérieur à RaptorQ/Qualcomm, avec 31 votes contre 30 (et 5 votes pour la proposition SuperCharged/Broadcomm).

6.6 *Pour plus d'informations...*

Nos publications relatives au thème :

- "RS + LDPC-Staircase Codes for the Erasure Channel: Standards, Usage and Performance", V. Roca, M. Cunche, C. Thienot, J. Detchart, et J. Lacan, [77]
  → *Cette publication détaille un certain nombre d'optimisations mises en oeuvre ainsi que les résultats de la compétition du 3GPP qui a vu notre proposition RS+LDPC-Staircase arriver en tête.*

D'autres travaux sur le thème LDPC et complexité algorithmique :

- "Analysis of Quasi-Cyclic LDPC codes under ML decoding over the erasure channel", M. Cunche, V. Savin et V. Roca", 2010, [21]
  → *Cette publication introduit des codes LDPC quasi-cycliques, qui après permutations de lignes/colonnes, peut être mis sous une forme quasi-bandes, ce qui permet de contrôler la complexité algorithmique du décodage ML.*
- "Performance Analysis of a High-Performance Real-Time Application with Several AL-FEC Schemes", K. Matsuzono, J. Detchart, M. Cunche, V. Roca et H. Asaeda", 2010, [55]
  → *Cette publication discute des performances de différents codes (dont LDPC-Staircase) pour un usage "petit bloc", afin de protéger des flux temps réels (voir chapitre 11).*
- "Erasure Codes with a Banded Structure for Hybrid Iterative-ML Decoding", A. Soro and M. Cunche and J. Lacan and V. Roca", 2009

[92]
→ *Cette publication discute de la possibilité de construire des codes LDPC dont la matrice de parité est creuse (pour le décodage IT) et dont la matrice génératrice présente une structure bande afin de contrôler la complexité algorithmique du décodage ML.*

Couple de pygargues, Canada, 2007.

7
Les codes GLDPC-Staircase

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux essentiellement menés avec F. Mattoussi et B. Sayadi durant ces dernières années, et auparavant avec M. Cunche et V. Savin.*

Résumé :
Ce chapitre présente les codes GLDPC-Staircase, de la famille des codes GLDPC, qui constituent une extension des codes LDPC-Staircase au moyen de codes Reed-Solomon avec matrice de Hankel. Les codes GLDPC-Staircase apportent plusieurs avantages pratiques en termes de capacités de correction, de performances de correction indépendantes du paramètre k, de faible code rate, de génération incrémentale de la redondance, tout en étant une évolution incrémentale et rétro-compatible avec les codes LDPC-Staircase.

7.1 Vers des codes plus performants

Nous avons détaillé dans les chapitres précédents les excellentes performances des codes LDPC-Staircase. De nombreuses évolutions sont cependant possibles.

7.1.1 Codes Quasi-Cycliques et "codes bande"

Ainsi nous avons proposé une évolution vers des codes Quasi-Cyclique (QC), ou QC-LDPC-Staircase [21] (figure 7.1). Ces codes sont inspirés par [99] (pour la construction QC) et [97] (pour les performances des codes bande) et ils permettent, du fait de leur structure interne, de contrôler la complexité de décodage ML tout en bénéficiant de performances de décodage améliorées[1]. Cependant nous ne détaillerons pas ce travail dans ce document.

[1] Une manipulation (permutation ligne/colonne) du système à résoudre permet en effet de se ramener à une structure pseudo-bande, dont la largeur est contrôlable aisément lors de la construction du code. La complexité du décodage par pivot de Gauss est ainsi maîtrisée (complexité en $O(k\sqrt{(k)})$ opérations lignes/colonnes), et les performances de correction proches de l'idéal.

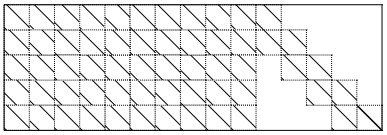


Figure 7.1: Matrice de parité d'un code QC-LDPC-Staircase (le bloc en bas à droite est une matrice escalier).

7.1.2 Une synthèse des techniques modernes : les codes WireHair

Les codes WireHair de Christopher A. Taylor, développeur réputé dans le monde du jeu, sont très intéressants, d'autant qu'ils sont associés à une implantation open-source disponible sur :
`http://wirehairfec.com`
Mélangeant matrice de parité majoritairement creuse avec une sous

matrice binaire dense, coefficients binaires et non binaires, une structure très astucieuse pour encoder efficacement un code binaire dense, et décodage de type SGE, c'est indéniablement une solution à hautes performances qui devrait bientôt compter... Plus d'informations sur la construction de ces codes (et bien d'autres choses) sont disponibles sur le blog de l'auteur [100].

Figure 7.2: Christopher A. Taylor, http://en.wikipedia.org/wiki/Chris_Taylor_(game_designer)

7.1.3 *La généralisation GLDPC*

Nous présentons dans ce chapitre une évolution des codes LDPC-Staircase utilisant une construction LDPC Généralisée, ou GLDPC. Cette évolution est motivée, entre autres choses, par le besoin d'avoir une famille de codes unique et performante à la fois pour les situations où la taille des blocs est importante (plusieurs milliers de symboles, typiquement lors de transfert de gros fichiers) et pour les situations où l'on doit protéger de petits blocs (quelques symboles, typiquement lors de la transmission de flux temps réel bas débit).

7.2 *Les codes GLDPC*

Les codes GLDPC constituent une extension des codes LDPC binaires dans laquelle les relations de parité, aussi appelées Single Parity Check (SPC) (chaque relation permet de recouvrer une seule perte de symboles), sont remplacées par des codes linéaires blocs en général dotés de meilleures capacités de détection/correction[2]. Nous appellerons **"codes composants"** ces codes de remplacement des SPC.

[2] Historiquement les codes GLDPC ont été proposés pour des canaux à pertes, d'où l'importance de pouvoir détecter et corriger des erreurs.

Cette proposition d'extension des codes LDPC a été proposée pour la première fois par R.M. Tanner dans [98]. Les codes composants (appelés "sub-codes" dans l'article) sont alors des codes linéaires binaires génériques, tandis que la matrice de parité LDPC est laissée inchangée.

Figure 7.3: R. Michael Tanner

Plus tard, d'autres méthodes de construction et d'autres codes composants ont été proposées, en particulier par Boutros et al. [11] et Lentmaier et al. [43], basées par exemple sur des codes de Hamming ou Bose, Ray-Chaudhuri, Hocquenghem code (BCH).

7.3 *Les codes GLDPC-Staircase*

Les codes GLDPC-Staircase [57] font partie de la famille des codes GLDPC : partant d'un code LDPC-Staircase donné, nous substituons des codes Reed-Solomon (RS) construits à partir de matrices de Hankel (section 7.4) aux codes SPC.

Ce passage est illustré par la figure 7.4 qui représente les graphes bipartites des codes LDPC-Staircase et GLDPC-Staircase : là où avec LDPC un "noeud de contrôle" reliait un ou deux noeuds de redon-

dance à des noeuds sources, avec GLDPC nous ajoutons en sus des symboles de redondance supplémentaires (les "noeuds variables" oranges du haut, dans cet exemple en nombre $E = 2$ par noeud de contrôle) du fait de l'usage de codes composants RS. Soit $1 + E$ le nombre total de symboles de redondance par "noeud de contrôle" du graphe bipartite, ou ce qui est équivalent, par ligne de la matrice de parité H_L.

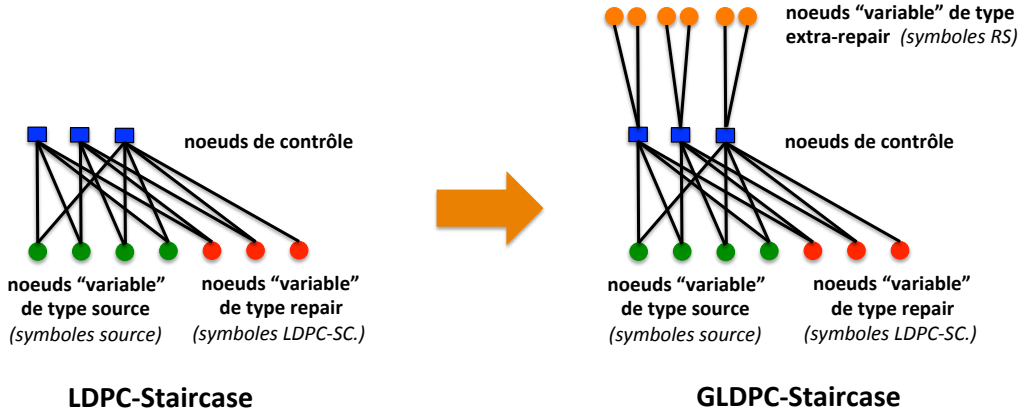


Figure 7.4: Graphes bipartites montrant le passage du code LDPC-Staircase(7, 4) à un code GLDPC-Staircase (13, 4) avec E=2 symboles "extra-repair" par noeud de contrôle.

Nous nous limitons dans la suite au cas où E (appelé nombre d'extra symboles de redondance) est le même pour toute ligne de H_L. C'est en effet la distribution qui nous offre les meilleures capacités de correction en décodage ML pour ces codes[3]. Soient les notations :

[3] Par exemple une distribution uniforme entre deux bornes, meilleure dans le cas d'un décodage IT+RS, est légèrement en retrait en décodage ML [22][57]).

Nom	Description
r_G	code rate du code GLDPC-Staircase, tel que vu par l'utilisateur
r_L	code rate interne du code LDPC-Staircase
H_L	matrice de parité LDPC-Staircase
E	nombre d'extra symboles de redondance pour un code composant donné

De la construction des codes GLDPC-Staircase il découle un lien direct entre les différents paramètres :

$$r_G = \frac{r_L}{1 + (1 - r_L)E} \tag{7.1}$$

Deux cas particuliers sont à noter :

- *le cas où $E = 0$ correspond à un code LDPC-Staircase;*
- *le cas où $r_L = 1$ et $E \geq 1$ correspond à un code RS;*

Ces cas particuliers peuvent être importants lorsque l'on désire avoir un comportement MDS avec de petits objets ou blocs sources (on pourra choisir alors $r_L = 1$ et $E \geq 1$). Ce peut également être un moyen de bénéficier de la faible complexité algorithmique des codes LDPC-Staircase lorsque le champ d'application l'exige (on choisira alors $E = 0$).

Propriété de dualité du premier symbole de redondance :

Les codes composants Reed-Solomon utilisés sont tels que leur premier symbole de redondance est égal à la somme XOR des symboles source de ce code. Dit autrement, pour la ligne d'indice $i \geq 1$ de la matrice H_L (NB : la première ligne a l'indice $i = 0$), le symbole de redondance au sens LDPC-Staircase, s_{k+i}, est égal à la somme XOR des symboles source de cette ligne et du symbole s_{k+i-1}. Ce même symbole est également le premier symbole de redondance créé par encodage RS prenant comme symboles sources (au sens RS) les symboles sources de cette ligne et le symbole s_{k+i-1}[4].

Nous verrons que cette propriété, qui vient de la construction par matrice de Hankel des codes RS considérés (section 7.4) a l'énorme avantage d'améliorer l'efficacité du décodage IT+RS.

[4] NB-1 : le cas particulier de la ligne d'indice $i = 0$ se traite de la même façon, la seule différence étant qu'elle n'admet pas de précédant symbole de redondance.
NB-2 : la propriété de dualité ne s'applique qu'au premier symbole de redondance. Les extra symboles de redondance n'ont de sens que par rapport aux codes RS.

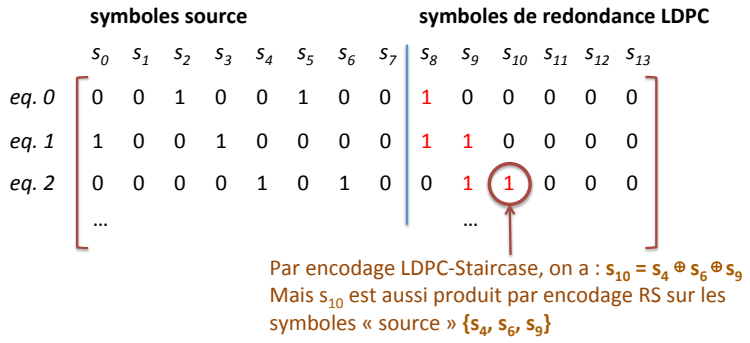


Figure 7.5: Dualité du premier symbole de redondance, code GLDPC-Staircase(14, 8).

Exemple :
Sur la figure 7.5, si l'on considère la troisième ligne de H_L, le premier symbole de redondance, s_{10}, est à la fois le résultat d'un encodage LDPC-Staircase (somme XOR des symboles source s_4, s_6 et du précédant symbole de redondance s_9) et le résultat d'un encodage RS prenant comme symboles sources les symboles $\{s_4, s_6, s_9\}$. A ce symbole s_{10}, E symboles de

redondance supplémentaires pourront être ajoutés en continuant l'encodage RS. Suivant le corps fini utilisé, ce nombre E peut être très élevé. En pratique $E = 1$, 2 ou 3 sont souvent utilisés.

Encodage :

L'encodage se fait traditionnellement, en débutant par les symboles de redondance LDPC-Staircase de façon incrémentale. Puis, si $E \geq 1$, un certain nombre d'extra symboles de redondance peuvent être produits. Il n'est pas nécessaire de les produire tous en pratique, ces derniers pouvant être générés à la volée, avant transmission. On pourra par exemple produire un extra symbole pour un sous ensemble choisi aléatoirement des relations définies par H_L.

La stratégie suivante d'encodage et de transmission des symboles est en général appropriée [58] :

- tout d'abord l'émetteur transmet séquentiellement les symboles sources[5];
- il transmet ensuite dans un ordre aléatoire les extra symboles de redondance définis par le code RS, et enfin dans un ordre aléatoire les symboles de redondance LDPC-Staircase. L'inverse est possible mais dans [58] nous montrons que transmettre les extra symboles avant permet au décodeur IT+RS de mieux fonctionner.

[5] Dans le cas d'une instance FECFRAME protégeant un flux streamé (chapitre 11), cela signifie que l'encodage FEC n'ajoute pas de délai d'encodage supplémentaire : chaque symbole source peut être transmis de suite, l'encodage n'intervenant qu'une fois le bloc constitué.

Décodage :

Le décodage suit une approche hybride, dans la même idée qu'avec les codes LDPC-Staircase, avec comme différence la présence éventuelle des extra symboles de redondances. Plus précisément :

- le récepteur effectue à la volée un décodage IT+RS. Cela signifie que chaque nouveau symbole source ou de redondance LDPC-Staircase reçu ou décodé est donné au décodeur IT, et chaque nouveau extra symbole de redondance reçu est donné au décodeur RS concerné. Les deux types de décodeurs fonctionnent en parallèle, les codes RS pouvant aider à décoder des symboles source ou de redondance LDPC-Staircase supplémentaires[6]. Si le décodage IT+RS aboutit, le récepteur s'arrête.
- sinon, si le récepteur sait qu'aucun autre symbole ne sera reçu, ou s'il estime qu'il est opportun de finir le décodage (il sait par exemple que celui ci a une très forte probabilité de réussir étant donné le nombre de symboles disponibles), une approche ML est alors utilisée. Un "décodage ML binaire" ne considérant que les

[6] A noter que l'inverse n'est pas intéressant : décoder un extra symbole n'a pas d'intérêt pratique.

équations binaires est tenté si c'est pertinent (par exemple si le nombre d'équations est suffisant). Si ce "décodage ML binaire" n'aboutit pas, un décodage ML considérant la totalité des équations sur $GF(2^8)$ est tenté.

7.4 *Les codes Reed-Solomon basés sur les matrices de Hankel*

Intéressons nous aux codes composants, et en particulier à la construction non classique basée sur les matrices de Hankel au lieu et place des matrice Vandermonde ou Cauchy.

Figure 7.6: Hermann Hankel

En algèbre linéaire, une matrice de Hankel, du nom du mathématicien Hermann Hankel, est une matrice carrée dont les valeurs sont constantes le long des diagonales ascendantes, c'est-à-dire dont les indices vérifient la relation $a_{i,j} = a_{i-1,j+1}$ *(source Wikipedia).*

L'usage de ces matrice pour construire les codes Reed-Solomon a un double avantage dans notre cas :

- il permet d'obtenir la propriété de dualité du premier symbole de redondance;
- il permet de réduire significativement le temps d'initialisation de la matrice génératrice, et donc les vitesses d'encodage/décodage si cette matrice génératrice doit être produite à la volée.

[84][56] décrivent la technique de construction de codes Reed-Solomon basés sur une matrice de Hankel. Plus précisément, la matrice triangulaire supérieure T_q est définie ainsi :

$$T_q = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 1 & b_1 & b_2 & b_3 & \cdots & b_{q-3} & b_{q-2} & \\ 1 & b_2 & b_3 & . & \cdots & b_{q-2} & & \\ 1 & . & . & \cdots & & & & \\ . & . & . & \cdots & & & & \\ . & . & b_{q-2} & & & & & \\ 1 & b_{q-2} & & & & & & \\ 1 & & & & & & & \end{pmatrix}$$

où les coefficients sont égaux à $b_i = \frac{1}{1-y^i}$, $1 \leq i \leq q-1$, et où y est un élément primitif arbitraire de $GF(q)$ et y^i est calculé sur $GF(q)$. Comme T_q ne contient pas de sous matrice carrée singulière [84], nous pouvons en extraire une sous matrice $A(k, n-k)$ et construire la matrice génératrice du code Reed-Solomon systématique ainsi :

$$G = [I_k | A_{k,n-k}]$$

Dès lors, étant donné le vecteur s contenant les symboles source, le vecteur des symboles d'encodage est obtenu par : $e = s \times G$.

Il est à noter que T_q est un cas particulier particulier de matrice de Hankel. Ce cas nous intéresse puisqu'il nous permet de satisfaire à la propriété de dualité du premier symbole de redondance, la première colonne étant constitués d'éléments tous égaux à 1[7].

Nous voyons que produire la matrice génératrice d'un code Reed-Solomon MDS et systématique, avec la propriété d'avoir une première colonne de $A_{k,n-k}$ égale à 1, est quasi immédiat et ne demande que très peu de calculs comparé à la méthode basée sur matrice de Vandermonde (le bénéfice est de deux ordres de grandeurs). Cette famille de code Reed-Solomon est donc hautement recommandée, indépendamment des codes GLDPC-Staircase.

Pour plus d'informations, voir [56][8].

[7] La propriété de dualité du premier symbole de redondance est satisfaite puisque la première colonne de la sous matrice $A_{k,n-k}$ de G ne contient que des éléments de $GF(2^8)$ égaux à 1. La multiplication du vecteur s (contenant les symboles source) par cette colonne de G équivaut à effectuer la somme au sens XOR des symboles sources.

[8] Cet article montre plus précisément que l'on observe avec des codes RS(250, 50) une réduction d'un facteur 95,8 du nombre d'opérations XOR et accès à des tables entre la version Hankel et la version Vandermonde, et un facteur 247,1 avec un code RS(250, 150). Ces gains sont confirmés, bien que moindres, si l'on considère les vitesses de création.

7.5 *Résultats de performance en capacités de correction*

Nous nous intéressons dans cette section aux capacités de correction des codes GLDPC-Staircase dans leur globalité, les codes composant RS étant pour leur part MDS.

7.5.1 *Méthodologie d'évaluation*

Pour cela deux approches ont été suivies :

- une étude théorique, permettant d'évaluer les performances asymptotiques au moyen soit d'une approche EXtrinsic Information Transfer (EXIT) chart, soit Density Evolution (DE). Ces aspects sont détaillés dans [59][60];
- des études par simulation, utilisant les outils de simulation disponibles dans le projet OpenFEC (chapitre 12.4.3).

Nous allons montrer que les performances sont exceptionnelles, y compris sur de très petits blocs (k faible). Ainsi il n'est pas nécessaire d'avoir plusieurs symboles par paquet afin d'augmenter (artificiellement) la taille effective du bloc (voir la discussion sur G de la section 5.2). Autrement dit il n'y a pas de paramètre G avec les codes GLDPC-Staircase.

Pour évaluer les probabilités d'échec de décodage, l'approche est identique à celle de la section 5.2 : chaque test élémentaire correspond à la fois à un code différent et à un séquencement de paquets différent. La probabilité d'échec de décodage doit être vue comme montrant des performances moyennes sur l'ensemble des codes possibles pour un quadruplet $\{n, k, N_1, r_L\}$ donné.

7.5.2 *Les paramètres N_1 et r_L*

Nous avons montré qu'utiliser $N_1 = 5$ offre un compromis intéressant entre capacités de décodage IT+RS (qui vont être dégradées si N_1 est trop élevé) et ML (qui à l'inverse s'améliorent si N_1 augmente).

De la même façon, nous nous limitons au cas où $r_L = 2/3$, ce qui signifie qu'ajouter un extra symbole de redondance ($E = 1$) par équation permet d'atteindre un code rate global $r_G = 1/2$.

7.5.3 *Résultats*

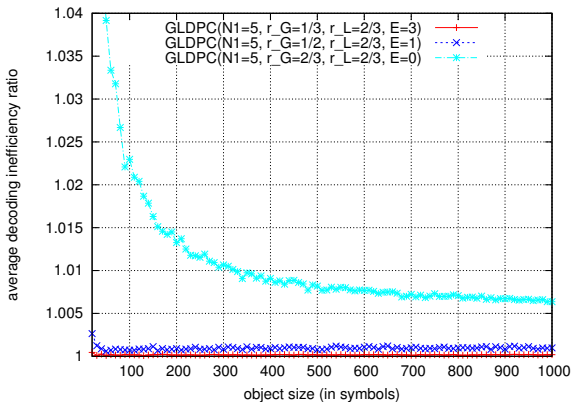
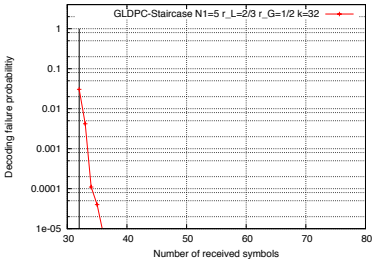


Figure 7.7: Capacités de correction moyens des codes GLDPC-Staircase (cas où $E = 0$, $N_1 = 5$ et non 7, bleu clair) et GLDPC-Staircase avec $E = 1$ ($r_G = 1/2$) et $E = 3$ ($r_G = 1/3$).

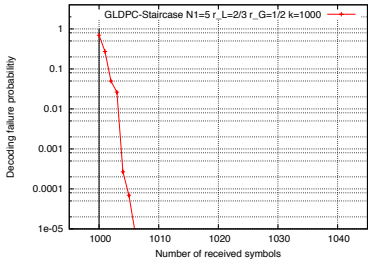
Les codes GLDPC-Staircase, du fait qu'ils incluent des équations avec coefficients sur $GF(2^8)$, ont d'excellentes capacités de correction, ceci pour toutes les valeurs de k. La figure 7.7 montre les capacités de correction moyennes pour ces codes, en fonction de E. Le cas où $E = 0$ correspond au cas des codes LDPC-Staircase (mais avec $N_1 = 5$ et non la valeur 7 plus communément utilisée). Sans surprise, les capacités de correction se dégradent si la dimension du code diminue. A l'inverse, le simple fait d'avoir $E = 1$ permet d'augmenter considérablement les capacités de correction, y compris pour les codes de dimension très faible.

Ces performances de correction se confirment lorsque l'on considère la probabilité d'échec de décodage en fonction de l'overhead, quelle que soit la dimension du bloc. Ainsi un overhead de 3 symboles pour $k = 32$ ou 5 symboles pour $k = 1000$ suffit pour descendre en dessous d'une probabilité de 10^{-4}.

Il est à noter qu'une configuration où $r_L = 1$ permet de revenir au cas particulier d'un code RS (section 7.3), donc avec des capacités de corrections idéales. Ce peut être utile pour des codes de dimension très faible.



(a) $k = 32$



(b) $k = 1000$

Figure 7.8: Pr_echec = (overhead) en ML des codes GLDPC-Staircase pour respectivement $k = 32$ et 1000 symboles ($N_1 = 5$, $r_L = 2/3$ et $r_G = 1/2$.

7.6 *Pour plus d'informations...*

Nos publications relatives au thème :

- "Impacts of the Packet Scheduling on the Performance of Certain Erasure Codes: Methodology and Application to GLDPC-Staircase", F. Mattoussi, V. Roca et B. Sayadi, [58]
  → *Cet article introduit une méthodologie pour une évaluation systématique de performances pour différents types d'ordonnancements de paquets et de canaux à mémoire, puis applique cette approche aux codes GLDPC-Staircase.*

- "Good coupling between LDPC-Staircase and Reed-Solomon for the design of GLDPC codes for the Erasure Channel", F. Mattoussi, B. Sayadi et V. Roca, [60]

- "Design of Small Rate, Close to Ideal, GLDPC-Staircase AL-FEC Codes for the Erasure Channel", F. Mattoussi, V. Roca, et B. Sayadi, [57]
  → *Cet article introduit les différentes techniques utilisées et détaille les performances obtenues par les codes GLDPC-Staircase.*

- "Optimization with EXIT functions of GLDPC-Staircase codes for the BEC", F. Mattoussi, V. Savin, V. Roca, et B. Sayadi, [59]
  → *Cet article introduit des modélisations par EXIT chart et DE des codes GLDPC-Staircase.*

- "Complexity comparison of the use of Vandermonde versus Hankel matrices to build systematic MDS Reed-Solomon codes", F. Mattoussi, V. Roca, et B. Sayadi", [56]
  → *Cet article détaille les codes Reed-Solomon systématiques construits à partir de matrices de Hankel puis montre que la création de leur matrice génératrice est plusieurs ordres de grandeur plus rapide que celle des codes basés sur les matrices de Vandermonde.*

- "Low-rate coding using incremental redundancy for GLDPC codes", M. Cunche, V. Savin, V. Roca, G. Kraidy, A. Soro et J. Lacan, [22]
  → *Dans cet article, nous introduisons pour la première fois les principes de construction des codes GLDPC-Staircase et en évaluons les performances lors d'un décodage IT (uniquement).*

Scène de vie de famille chez les blaireaux, Isère, avril 2013

Part II

Schémas FEC : quand la signalisation rejoint les codes

8
Utilisation des codes AL-FEC : schéma FEC et signalisation

Résumé :
Ce chapitre discute de la notion de "schéma FEC" (ou "FEC Scheme") introduite par les groupes RMT et FECFRAME de l'IETF. Leur objectif est de fournir d'une part une spécification exhaustive et non ambiguë des codes AL-FEC, et d'autre part une spécification de la signalisation associée, indispensable à leur usage au sein de protocoles.

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux menés au sein de l'IETF et des groupes de travail Reliable Multicast Transport (RMT) et FECFRAME. Les propositions de schémas FEC ont également impliqué M. Cunche, J. Lacan, K. Matsuzono, A. Bouabdallah, D. Furodet, S. et J. Peltotalo.*

8.1 Introduction

Nous avons vu dans les chapitres précédents plusieurs exemples de codes AL-FEC ainsi que les performances que l'on peut en attendre. Nous nous intéressons maintenant à l'usage des codes AL-FEC au sein des protocoles définis par les groupes RMT et FECFRAME de l'IETF. Pour cela il faut bien évidemment disposer d'une spécification exhaustive et non ambiguë de ces codes : comment produire des symboles de redondance durant l'encodage et comment les exploiter durant le décodage ? Il faut également décrire de façon exhaustive et non ambiguë la signalisation permettant l'usage de ces codes dans un protocole donné : comment synchroniser encodeur et décodeur et comment indiquer quel symbole un paquet contient ?[1] Nous insisterons tout particulièrement sur cet aspect de signalisation, essentiel en pratique.

[1] D'autant qu'il est parfois utile d'avoir plusieurs symboles par paquet (section 5.1)...

8.2 La notion de schéma FEC

La notion de schéma FEC est décrite au sein du RFC 5052 [105]. Ce document définit à la fois :

- **les codes AL-FEC**, à savoir la création du code, l'encodage (production de symboles de redondance), et le décodage (exploitation des

symboles d'encodage reçus);

- **les algorithmes de dérivation de certains paramètres**, tels que les algorithmes relatifs à la dimension (k) et la longueur du code (n);
- **la signalisation** indispensable à l'usage de ces codes, qui permet aux encodeurs et décodeurs de se synchroniser.

Grâce à un schéma FEC, l'usage des codes est possible au sein des protocoles définis par les groupes IETF RMT et FECFRAME (entre autres). Cependant des différences majeures existent entre les deux familles de protocoles (par exemple la façon dont les blocs de l'objet sont définis) et des schémas spécifiques sont donc nécessaires.

Exemple de différence entre un usage RMT et FECFRAME :
Le RFC 5170 [81] définit un schéma pour les codes LDPC-Staircase adapté aux protocoles ALC et NORM du groupe RMT. Dans ce contexte, pour un objet donné, les blocs sont issus d'un découpage le plus régulier possible de l'objet (la différence entre blocs de taille maximum et minimum est d'au plus 1 symbole). C'est l'objet du "block partitioning algorithm" décrit en section 9.1 du RFC 5052.

Le RFC 6816 [75] définit un schéma pour ces mêmes codes (le RFC 6816 hérite de la définition des codes du RFC 5170) mais adapte les algorithmes de dérivation de paramètres ainsi que la la signalisation aux spécificités de l'architecture FECFRAME. Dans ce contexte, pour une session donnée, l'objet pourra être constitué de "la totalité du flux TV streamé". Si ce flux est capturé en temps réel, l'émetteur n'en connaît pas la taille avant que l'émission ne soit achevée. Le découpage en blocs se fait donc sur une autre base, à partir d'un temps d'attente pour la constitution d'un bloc qui correspond également à la latence maximum de décodage.
Nous reviendrons sur ces aspects dans la section 8.4.

Vous le voulez totalement ou partiellement spécifié, votre schéma FEC ?

Les schémas peuvent être totalement spécifiés (**ou "Fully Specified"**) et ils font alors référence à une seule famille de codes AL-FEC. C'est le cas général. Le schéma est alors totalement identifié par le FEC Encoding IDentifier (FEC Encoding ID). [2]

Les schémas peuvent être sous spécifiés (**ou "Under Specified"**) et ils regroupent alors plusieurs codes AL-FEC qui partagent les mêmes algorithmes et signalisation. La seule fourniture du FEC Encoding ID

[2] Par exemple le RFC 5170 spécifie un schéma totalement spécifié pour les codes LDPC-Staircase (FEC Encoding ID 3). Le RFC 5510 spécifie pour sa part les schémas totalement spécifiés FEC Encoding ID 2 et 5 qui s'appliquent respectivement aux codes Reed-Solomon codes sur $GF(2^m)$, où $m \in 2;16$, et $GF(2^8)$ (cas particulier).

est alors insuffisante pour utiliser ce schéma et il faut y adjoindre l'identifiant d'instance ou FEC Instance IDentifier (FEC Instance ID)[3].

A titre d'illustration, la figure 8.1 montre les tables Internet Assigned Numbers Authority (IANA) enregistrant les schémas FEC du groupe de travail RMT, selon qu'ils sont totalement ou partiellement spécifiés :

`http://www.iana.org/assignments/rmt-fec-parameters/rmt-fec-parameters.xml`

[3] Par exemple le RFC 5445 décrit le schéma sous spécifié FEC Encoding ID 129 regroupant les codes petits blocs systématiques. En tant que tel, ce schéma ne peut pas être utilisé dans les protocoles. Le RFC 5510 décrit pour sa part le schéma (FEC Encoding ID 129, FEC Instance ID 0) pour les codes Reed-Solomon sur $GF(2^8)$, qui lui précise tous les détails pratiques et peut être utilisé.

ietf:rmt:fec:encoding - Fully-Specified FEC schemes (0-127)

Registration Procedure(s)
IETF Consensus
Reference
[RFC5052]

Value	Description	Reference
0	Compact No-Code	[RFC5445]
1	Raptor	[RFC5053]
2	Reed-Solomon Codes over GF(2^^m)	[RFC5510]
3	LDPC Staircase Codes	[RFC5170]
4	LDPC Triangle Codes	[RFC5170]
5	Reed-Solomon Codes over GF(2^^8)	[RFC5510]
6	RaptorQ Code	[RFC6330]
7-127	Unassigned	

ietf:rmt:fec:encoding - Under-Specified FEC schemes (128-255)

Registration Procedure(s)
IETF Consensus
Reference
[RFC5052]

Value	Description	Reference
128	Small Block, Large Block and Expandable FEC Codes	[RFC5445]
129	Small Block Systematic FEC Codes	[RFC5445]
130	Compact FEC	[RFC5445]
131-255	Unassigned	

Figure 8.1: Enregistrement IANA des schémas FEC totalement et partiellement spécifiés.

Qu'est-ce que le schéma FEC No-Code ?

La figure 8.1 montre un schéma FEC particulier : No-Code. Comme suggéré, il s'agit d'un schéma qui n'est pas associé à un code AL-FEC particulier. Il n'y a donc pas de symbole de redondance suite à "l'encodage No-Code" et ce schéma ne sera d'aucune utilité pour corriger des effacements. Mais il reste essentiel, ceci pour trois raisons :

- Tout d'abord il est essentiel en tant que dénominateur commun, afin de garantir une interopérabilité. Ainsi toute implantation de FLUTE/ALC doit supporter ce schéma.

- Ensuite, des objets de taille très faible (par exemple quelques dizaines d'octets), tenant dans un unique paquet ALC, ne nécessitent pas d'encodage FEC. Une simple duplication de ce paquet permettra d'apporter la fiabilité cherchée[4].

- Enfin, il peut être utile d'avoir un moyen d'identifier chaque symbole source d'un objet, indépendamment de tout encodage FEC (c'est le cas de notre approche GOE). C'est aussi ce que permet le schéma No-Code qui effectue un découpage en bloc(s) suivi d'un découpage de chaque bloc en symbole(s).

[4] Nous verrons avec l'approche GOE (chapitre 9) qu'il sera souvent pertinent de protéger globalement plusieurs objets, par exemple de petite taille, afin d'améliorer la robustesse des transmissions.

8.3 Gros plan sur la signalisation

L'encodeur AL-FEC et le décodeur AL-FEC sont exécutés sur deux entités distinctes, par exemple un serveur et un smartphone, et reçoivent des paquets via le canal de communication. De ce fait les paramètres utilisés à l'encodage d'un bloc doivent être communiqués au décodeur. De même, un paquet reçu par un récepteur doit comporter des informations permettant d'identifier précisément le ou les symboles qu'il contient. Pour cela, deux types de signalisation sont distinguées :

- les informations globales à l'objet, et
- les informations d'identification relatives à un paquet.

Distinguons tout d'abord le cas d'un usage au sein d'un protocole du groupe RMT de celui du groupe FECFRAME.

8.3.1 Le cas de RMT

Informations globales à l'objet :

Dans le contexte RMT, les informations globales à l'objet sont appelées *FEC Object Transmission Information (FEC OTI)* . Ces données sont partagées par les instances émission et réception et **permettent la synchronisation des encodeur et décodeur AL-FEC**.

Exemple de FEC OTI :
Nous prenons l'exemple du Schéma LDPC-Staircase (RFC 5170). Le FEC OTI comprend :

FEC Encoding ID : Il s'agit là de l'identifiant de schéma AL-FEC.

Taille d'objet transféré (L) : Dans le cas où l'objet soumis par l'utilisateur subit une transformation (par ex. une compression gzip), il s'agit de la taille après transformation de l'objet, tel que transféré, et non de la taille d'origine;

Taille de symboles (E) : Il s'agit de la taille de tout symbole, source ou redondance. Dans le cas d'un objet qui n'a pas une taille multiple de E, le dernier symbole source transmis sera plus petit. En revanche, tous les symboles de redondance sont de taille E, et le dernier symbole source devra être "bourré avec o" lors de son utilisation à l'encodage ou décodage.

Nombre maximum de symboles source par bloc (B) : Il est clair que les codes peuvent être limités en taille maximum de bloc du fait de leur principes de fonctionnement[5]. A cela un codec peut à son tour imposer des limites supplémentaires, par exemple liées à la complexité algorithmique lors du décodage, ou à la taille (en bits) d'une variable utilisée en interne pour

[5] Par exemple Reed-Solomon sur GF(2^8) qui sera qualifié de code "petit bloc". C'est moins vrai avec ces codes LDPC dits de type "grand bloc" qui acceptent une valeur B élevée.

stocker un identifiant[6]*. Ce paramètre permet de spécifier ce type de contraintes pratiques.*

[6] Ainsi des indices de ligne/colonne sur 16 bits permettront d'avoir au maximum 2^{16} symboles après encodage.

Nombre maximum de symboles d'encodage (max_n) : Les codes LDPC-Staircase ayant un code rate prédéfini, ce dernier est communiqué via ce paramètre max_n. Le ratio B/max_n donnera alors le code rate utilisé par l'encodeur.

G : Les codes LDPC étant des codes assymptotiquement bons, il peut être utile d'augmenter artificiellement le nombre de symboles en considérant des valeurs E plus faibles que ce que permet la taille maximum de paquet (Path Maximum Transmission Unit (PMTU)), moins les divers en-têtes. G indique donc le nombre de symboles se trouvant dans un paquet donné. Par défaut, $G = 1$ (un seul symbole par paquet).

$N_1 m3$: (ou N_1 moins 3) Le nombre de "1s" par colonne de la matrice de parité, N_1 (section 4.3), est donné par : $N_1 = N_1 m3 + 3$.

PRNG seed : Puisque la construction de la matrice de parité dépend à la fois d'un algorithme de remplissage et d'un PRNG, cette graine est transmise au décodeur.

Nous verrons qu'avec FECFRAME, le format sera différent.

Les FEC OTI peuvent être transmises de multiples façons. La solution privilégiée consiste à les transmettre sous forme eXtensible Markup Language (XML) au sein d'une "instance de File Delivery Table (FDT)" de FLUTE (c'est à dire d'une table décrivant le contenu du carrousel de transmission à un moment donné, voir le chapitre 10).

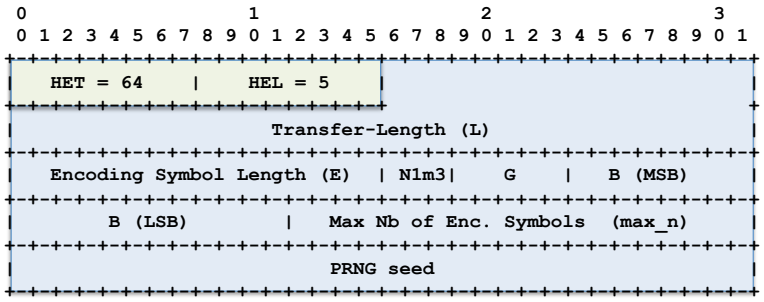


Figure 8.2: Format binaire du FEC OTI du RFC 5170. Le champ Header Extension Type (HET) indique une extension d'en-tête ALC/LCT de type EXT_FTI, tandis que le champ Length (HEL) donne sa longueur, ici 5 mots de 32-bits.

Les FEC OTI d'un objet peuvent aussi être transmises sous forme binaire via une extension d'en-tête ALC spécifique, `EXT_FTI`, au sein de paquets ALC transmis contenant des données de cet objet. Au sein de la figure 8.2, le champ `HET` (Header Extension Type) indique une extension d'en-tête ALC/LCT de type `EXT_FTI`, tandis que le champ `HEL` donne la longueur, en mots de 32-bits, de cette même extension d'en-tête (ici 5 mots).

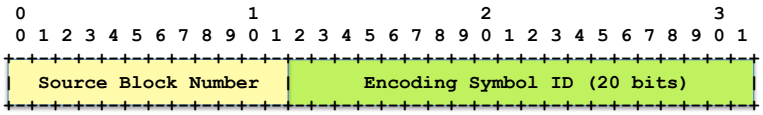


Figure 8.3: Format binaire du FPI du RFC 5170.

Informations relatives à un paquet :

Les **informations d'identification** relative à un paquet sont appelées *FEC Payload IDentifier (FPI)* . Ces données associent un identifiant de symbole au sein du bloc, Encoding Symbol Identifier (ESI), et un identifiant de bloc au sein de l'objet, Source Block Number (SBN), à chaque paquet.

Lorsque $G = 1$ (cas général), ce FPI identifie l'unique symbole du paquet. Lorsque $G > 1$ (cas général), ce FPI identifie le premier symbole du paquet, les autres identifiants étant déduit de ce premier FPI par un algorithme précis[7]. Muni de plus des informations de l'entête ALC, le récepteur sait dès lors exactement à quel objet de quelle session ce symbole appartient, la position précise de ce symbole dans l'objet (identifiant de bloc et position au sein du bloc), ainsi que sa nature (symbole source ou de redondance).

[7] Lorsque $G > 1$, les paquets contiennent soit uniquement des symboles sources, soit uniquement des symboles de redondance. Dans le premier cas, les symboles sources sont séquentiels, depuis ESI jusqu'à $ESI + G - 1$. Dans le deuxième cas, les symboles de redondance sont tout d'abord permutés de façon aléatoire, en suivant un algorithme précis, puis l'émetteur prend G symboles successifs dans cette permutation. Si le nombre de symboles de redondance n'est pas multiple de G, l'encodeur complète avec les premiers symboles de cette permutation de façon à avoir exactement G symboles.

8.3.2 *Le cas de FECFRAME*

Informations globales à l'objet :

Dans le contexte FECFRAME, les informations globales à l'objet sont appelées *FEC Framework Configuration Information (FFCI)* . Ici également, ces données sont partagées par les instances émission et réception et **permettent la synchronisation des encodeur et décodeur AL-FEC**.

Exemple de FFCI :
Nous prenons là encore l'exemple du Schéma LDPC-Staircase (RFC 6816).
Le FEC OTI comprend :

FEC Encoding ID : Il s'agit là de l'identifiant de schéma AL-FEC.

Bit "taille stricte" (S) et taille de symboles (E) : Le bit S indique, s'il vaut S=1, que le paramètre E indique la taille effective des symboles durant toute la session, et sinon, si S=0, que E indique la taille maximum, la taille effective pouvant être temporairement choisie plus faible pour un bloc donné.

$N_1 m3$: Le nombre de "1s" par colonne de la matrice de parité, N_1 (section 4.3), est donné par : $N_1 = N_1 m3 + 3$.

PRNG seed : Puisque la construction de la matrice de parité dépend à la fois d'un algorithme de remplissage et d'un PRNG, cette graine est transmise au décodeur.

Nous constatons que le format est bien différent de celui du cas RMT. En particulier les paramètres k et n ne sont pas communiqués dans la FEC OTI, mais dans la FPI car le découpage en blocs dépend aussi et surtout du flux applicatif et des éventuelles contraintes temps réelles (cf. section 8.4).

Les FEC OTI peuvent être transmises de multiples façons, en général au sein d'une description de la session au format Session Description Protocol (SDP) ou XML. Cette description est alors transmise aux récepteurs via une technique quelconque (via Real-Time Streaming Protocol (RTSP), Session Initiation Protocol (SIP) ou autre). Les paramètres FEC OTI sont donc spécifiés une fois pour toute pour la durée de la session. C'est là une distinction importante avec les cas d'usage RMT où chaque objet transmis est décrit dans un FEC OTI spécifique.

Exemple d'encodage SDP de FFCI :
Un exemple de description FFCI pour les codes LDPC-Staircase, au sein d'une description SDP, est la suivante :
a=fec-repair-flow: encoding-id=7; fssi=seed:1234,E:1400,S:0,n1m3:4
Ici 7 est le FEC Encoding ID[8]*. Le champ FEC-Scheme-Specific Information (FSSI) contient les informations spécifiques à ce schéma, en format textuel cette fois ci. Attention, le champ "encoding-id" plus les informations contenues dans le champ FSSI constituent le FEC OTI.*

[8] Un nouvel espace d'adressage FEC Encoding ID a été créé pour les besoins de FECFRAME ce qui explique qu'il soit différent de celui utilisé dans le contexte RMT.

Informations relatives à un paquet :

Les informations d'identification relatives à un paquet sont là aussi appelées *FPI* . Elles permettent tout d'abord **d'identifier les symboles reçus**, à savoir identifier le symbole au sein du bloc (via l'ESI) et le bloc au sein du flux (via le SBN). Mais contrairement aux schémas du RMT, la constitution du bloc source peut évoluer dynamiquement en fonction du flux applicatif, ce qui rend nécessaire de le communiquer pour chaque bloc (là où cette information était factorisée dans le FEC OTI dans RMT). Ainsi le FPI permet également de transmettre des **paramètres relatifs au découpage en blocs du flux de données applicatif et à l'encodage AL-FEC réalisé**. Il s'agit là aussi d'une différence importante par rapport au cas RMT liée au mode de flux de données.

Ainsi, dans le cas particulier des codes LDPC-Staircase, ces données associent un identifiant de symbole, ESI, un identifiant de bloc, SBN,

mais aussi k et n à chaque paquet de redondance (figure 8.4). Un format plus simple (ne donnant pas le paramètre n) est associé aux paquets source.

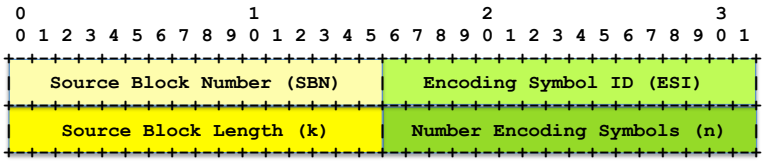


Figure 8.4: Format binaire du FPI du RFC 6816 dans le cas d'un paquet de redondance.

8.4 *Découpage d'un objet en blocs*

Le découpage en blocs est un processus indispensable pour deux raisons :

- les codes AL-FEC ont des limites sur la taille maximale d'un bloc qui peuvent empêcher un encodage en une seule fois de la totalité de l'objet (voir le cas de RMT);
- les données peuvent ne pas être disponible en totalité alors qu'il est impossible d'attendre plus (voir le cas de FECFRAME).

Nous allons voir que la façon de procéder diffère largement suivant le champ d'application.

8.4.1 *Le cas de RMT*

Ici l'émetteur dispose d'emblée de la totalité de l'objet une fois celui-ci communiqué par l'utilisateur. Mais avant de pouvoir réaliser un encodage, il faut découper cet objet en blocs de telle façons que ces blocs soient en deçà de la taille maximale des codes AL-FEC utilisé, et de telle façon que le récepteur soit en mesure de retrouver le découpage exact effectué par l'émetteur, ceci sans alourdir la signalisation[9].

Pour cela [105], section 9.1, définit un algorithme que tout schéma FEC du groupe RMT doit suivre. Partant des paramètres nombre maximum de symboles source par bloc, B, taille d'objet transféré, L, et taille de symboles, E, cet algorithme fournit le nombre total de symboles source, T, le nombre total de blocs, N, le nombre de blocs de taille maximale, I, et la taille des blocs de taille maximale A_{large} et minimale A_{small}.

[9] Il serait par exemple peu efficace de communiquer la liste des tailles (nombre de symboles sources) de chaque bloc d'un gros objet.

Soient B, L et E. Alors :
$T = ceil[L/E]$;
$N = ceil[T/B]$;
$A_{large} = ceil[T/N]$;
$A_{small} = floor[T/N]$;
$I = T - A_{small} * N$;

Par construction A_{large} et A_{small} sont soit égaux, soit diffèrent de 1 symbole seulement : tous les blocs sont garantis d'être de taille quasi égale[10].

[10] Cet algorithme permet de gérer les situations telles que, par exemple, $T = B + 1$. Une approche naïve conduirait alors à avoir deux blocs, le premier de taille B symboles source et le deuxième de taille 1. La protection de ce bloc minuscule serait alors largement problématique ! Avec l'algorithme officiel, on a $N = 2$ blocs, l'un de taille $A_{large} = ceil((B+1)/2)$, l'autre de taille $A_{small} = floor((B+1)/2)$.

8.4.2 *Le cas de FECFRAME*

Si dans les approches RMT le composant ALC (ou NORM suivant le cas) dispose d'emblée de la totalité des données à protéger, ici FECFRAME travaille en "mode flux de données". Ainsi les Application Data Unit (ADU)s que soumet l'application (par exemple un serveur de streaming `VLC` ou `ffmpeg`) arrivent au fur et à mesure et l'instance FECFRAME doit en regrouper un nombre adéquat afin de créer un bloc source et de procéder à l'encodage AL-FEC.

Le découpage en blocs d'un flux applicatif n'étant pas imposé par [104] (à l'inverse du RFC 5052 pour RMT), nous prendrons l'exemple du schéma LDPC-Staircase pour FECFRAME [75]. L'instance FECFRAME accumule dans un premier temps les ADU. Trois causes limitent le nombre d'ADU :

- le code et le codec utilisés qui peuvent induire des limites propres (par exemple liées à la complexité algorithmique);
- le schéma FEC utilisé qui a lui aussi ses contraintes (par exemple la taille du champ ESI permettant de stocker les identifiants de symboles d'encodage, pour un code rate donné);
- les caractéristiques temps réelles éventuelles du flux qui induisent un délai de décodage maximum, qui lui même se transpose en une taille de bloc maximale.

Supposons que code, codec et schéma conduisent conjointement à une taille maximum de max_k ADUs par bloc. Soit l la latence de décodage maximum désirée, et d la durée couverte par chaque ADU d'un flux à débit constant. Nous cherchons la taille maximum de bloc max_B.

Soit l, d et max_k. Alors :
$max_{rt} = floor(l/d)$;
$max_B = min(max_k, max_{rt})$;

Nous savons alors que l'émetteur ne devra jamais accumuler plus de max_B ADU dans un bloc source. Si à un moment donné le débit instantané du flux applicatif décroît significativement, alors le nombre de symboles dans le bloc pourra lui aussi diminuer significativement.

8.5 *Pour plus d'informations...*

Nos publications relatives au thème :

- **[RFC 5510]** "Reed Solomon Error Correction Scheme", J. Lacan, V. Roca, J. Peltotalo, S. Peltotalo, 2009, [40]
  → *C'est le standard de référence pour les codes correcteurs Reed-Solomon, pour un usage au sein des protocoles FLUTE/ALC.*
- **[RFC 5170]** "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes", V. Roca, C. Neumann, D. Furodet, 2008, [81]
  → *C'est le standard de référence des codes correcteurs LDPC-Staircase, qui en spécifie tous les détails.*
- **[RFC 6865]** "Simple Reed-Solomon Forward Error Correction (FEC) Schemes for FECFRAME", V. Roca, M. Cunche, J. Lacan, A. Bouabdallah, K. Matsuzono, 2013, [76]
  → *Ce document explique comment utiliser les codes Reed-Solomon (RFC 5510) dans un contexte streaming.*
- **[RFC 6816]** "LDPC-Staircase Forward Error Correction (FEC) Schemes for FECFRAME", V. Roca, M. Cunche, J. Lacan, 2012, [75]
  → *Ce document explique comment utiliser les codes LDPC-Staircase (RFC 5170) dans un contexte streaming.*

Bain de soleil du lézard vert, sur son pierrier, Hautes Alpes, 2011.

9
Les schémas FEC GOE, ou comment apporter protection d'erreur inégale ou globale

Résumé :
Ce chapitre explique comment aller au delà d'un encodage AL-FEC classique, en apportant soit une protection inégale au sein d'objets, soit une protection globale à un ensemble d'objets, tout ceci sans toucher au codes AL-FEC. Nous comparons également notre proposition, GOE, avec l'approche de Qualcomm, et montrons la nette supériorité de GOE.

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux menés avec A. Roumy, B. Sayadi et R. Imad.*

9.1 Objectifs de l'approche GOE

Si un objet est structuré et contient des données de différents niveaux de priorité, il est naturel de vouloir offrir un niveau de protection différencié par lequel les données les plus importantes bénéficieront d'une meilleure protection. Ce service, appelé Unequal Erasure Protection (UEP), est le premier service que les schémas FEC de type GOE offrent.

Ensuite, lorsque l'on doit transférer un certain nombre d'objets de diverses tailles (en particulier de très petite taille), chacun d'eux devant être reçu par les récepteurs, protéger individuellement chaque objet est clairement inefficace[1]. Procéder par la création préliminaire d'une archive peut être une solution dans certaines situations, mais elle nécessite des manipulations supplémentaires (archivage coté émetteur et dés archivage coté récepteur) qui ne sont pas toujours souhaitables. Les schémas FEC de type GOE offrent en revanche un service de protection global d'un ensemble d'objets hautement efficace.

[1] Il s'agit là encore d'une manifestation du problème du collecteur de coupons, déjà discuté à la section 3.2.1. Pour s'en convaincre intuitivement, il suffit d'imaginer un scénario où l'on doit transmettre une centaine d'objets de taille un octet chaque. Les protéger indépendamment, seule solution avec les schémas AL-FEC discutés en section 8, sera peu efficace car on travaille alors nécessairement sur chaque objet de façon indépendante.

9.2 Trois familles de solutions

Concentrons nous sur l'objectif de protection inégale, UEP. Trois familles de solutions sont possibles :

- utiliser des codes FEC spécifiques;
- utiliser une approche de paquetisation spécifique;
- utiliser un mécanisme de signalisation spécifique;

Nous passons successivement en revue ces trois approches.

9.2.1 *Les approches FEC spécifiques à l'UEP*

Dans cette approche la protection inégale est rendue possible par l'utilisation de codes AL-FEC spécifiques, construits spécialement pour offrir un service de protection inégale. C'est le cas de Dependency-Aware Unequal Erasure Protection (DA-UEP) [9] dont la matrice génératrice est tout spécialement construite, en exploitant au mieux les dépendances existant naturellement au sein du flot de données.

Un exemple typique consiste en la protection d'un Group Of Pictures (GOP) d'un flux vidéo : recevoir une trame **P** n'a aucun intérêt si la trame **I** dont elle dépend n'est pas disponible. C'est cette interdépendance qui est codée en dur au sein du code.

Bien qu'efficace, cette approche est restrictive et sera utilisée pour des champs d'application spécifiques, où le flux est connu et relativement stable dans ses caractéristiques.

9.2.2 *Les approches de paquetisation spécifiques à l'UEP*

Dans cette deuxième classe de solutions nous trouvons l'approche PET [5][10], ainsi que l'approche "UOD pour RaptorQ" [50], récente, qui en est une réalisation pratique[2]. L'idée est de tirer partie d'une paquetisation appropriée des symboles d'encodage afin de fournir un service de protection inégale (PET et UOD), voir de protection globale d'un ensemble d'objets (UOD seulement).

[2] A noter l'approche de Urbanke et Winner [103], équivalente mais limitée à un encodage Reed-Solomon dans sa présentation.

La figure 9.1 illustre cette approche. Elle montre deux classes de priorités distinctes : les deux symboles hautes priorité ("objet" O_1) constituent les données prioritaires du flux, et les sept symboles basse priorité ("objet" O_2) sont constitués du reste du flux[3]. Chaque niveau de priorité bénéficie d'un encodage AL-FEC adapté, avec la contrainte que chaque objet i dispose du même nombre de symboles **d'encodage**, n_{PET}. Fournir une protection inégale, c'est à dire un code rate $CR_i = k_i/n_{PET}$ différent, nécessite alors de jouer sur k_i. La solution consiste à jouer sur la taille s_i des symboles pour l'objet i, seul paramètre libre, sous la contrainte que la somme de toutes les tailles de symboles soit inférieure ou égale à la taille maximum d'un paquet, l : $\sum_{i=0}^{d} s_i \leq l$, où d est le nombre d'objets. Ainsi un objet prioritaire sera découpé en symboles de taille potentiellement importante afin que le nombre de symboles source soit faible, et de ce fait le code rate faible. A l'inverse,

[3] La notion d'objets est floue dans la terminologie UOD. Nous considérerons qu'il s'agit d'un sous ensemble de données de même niveau de priorité dans cette section. Si un objet source est constitué de données de priorité inégale, chaque objet regroupera des données de même priorité.

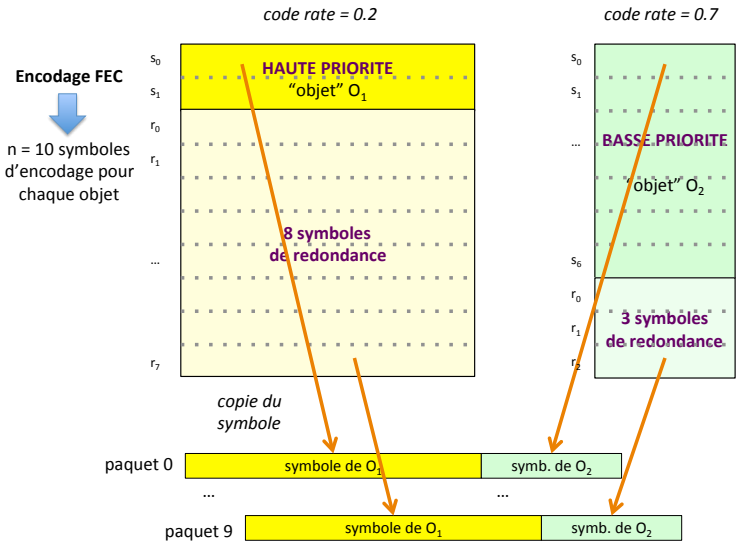


Figure 9.1: Principes de l'approche UOD pour RaptorQ.

un objet moins prioritaire sera découpé en symboles de taille potentiellement plus faible afin d'en augmenter le nombre et de ce fait avoir un code rate élevé.

La création d'un paquet consiste alors à sélectionner un symbole d'encodage de chaque groupe, puis de les concaténer afin de constituer le paquet. Chaque paquet donné est donc constitué d'un ensemble de symboles d'encodage, un par objet source. La réception d'un paquet contribue alors exactement à la réception d'un symbole pour chaque classe de priorité.

UOD permet également une protection groupée d'un ensemble d'objets sources. Pour cela [50] considère que chaque objet contribue pour un symbole d'encodage dans le paquet. Le code rate cible est ici le même pour tous, du fait qu'il s'agit de protéger l'ensemble des objets de façon équivalente. Recevoir un paquet permet dès lors de contribuer à chaque objet à hauteur d'un symbole. L'objectif est rempli.

Si l'on y regarde de plus près, l'approche a des limites sérieuses :

- manque de flexibilité : la taille de chaque symbole est une conséquence des autres paramètres : taille et nombre des objets, taille maximum de paquet, et codes rate cibles. Les codes RaptorQ imposent eux aussi une granularité de 4 octets, ce qui limite également les choix possibles. Toutes ces contraintes conduisent à des arrondis dans la détermination des paramètres, ces arrondis étant à l'origine d'inefficacités. Nous exhibons un tel exemple dans `http://www.ietf.org/proceedings/81/slides/rmt-2.pdf`.

- complexité : indépendamment de la complexité de l'approche (il suffit de lire [5] pour s'en convaincre), le paramétrage de la so-

lution nécessite un grand nombre d'étapes qui, bien que n'étant pas décrites dans [50], n'en sont pas moins essentielles. De plus, les symboles étant de taille inférieure au maximum permis par le PMTU, cela signifie que les dimensions des codes est souvent supérieure à ce qui serait permis par une approche plus classique (par exemple GOE, section 9.3). Les systèmes linéaires à résoudre sont dès lors plus grands, ce qui a un impact direct sur la complexité algorithmique de l'approche.

- manque de stabilité : lors de l'usage de UOD pour protéger un ensemble d'objets, ajouter un objet supplémentaire de taille un octet peut faire effondrer le système, passant d'une situation où tous les objectifs sont parfaitement remplis (remplissage des paquets, code rate cibles) à une situation où plus rien ne va, le tout petit objet se trouvant exagérément protégé et le paquet insuffisamment remplis.

Analyse critique: *Le gros avantage des solutions PET/UOD est le déterminisme des résultats. Le prix à payer est par contre élevé. Ceci nous amène à dire que s'il s'agit d'une excellente idée sur le papier, en revanche on ne saurait recommander UOD pour des réalisations pratiques dans le cas général.*

9.2.3 *Les approches de signalisation FEC spécifiques à l'UEP*

Dans cette troisième classe de solution la protection inégale ainsi que la protection globale d'un ensemble d'objets sont assurés par des mécanismes de signalisation. L'idée est de pouvoir dissocier objets sources et objets que l'on qualifiera de "généralisés" sur lesquels se fait l'encodage FEC. Dès lors que l'émetteur est en mesure de dissocier les deux notions, tout devient possible puisqu'un "objet généralisé" peut être constitué d'un sous ensemble d'un objet source (ce qui permet d'apporter une protection inégale au sein d'un objet donné), ou à l'inverse être constitué d'un ensemble d'objets source. Deux objets généralisés peuvent se chevaucher et donc protéger tous deux la même zone de données, ou bien être disjoints, la partie non couverte des données sources n'étant pas protégée, ou être associés à des code rates différents de façon à offrir une protection différenciée. L'émetteur dispose d'une totale liberté qui lui permet d'offrir le service le plus adapté aux besoins.

Aussi attractive soit elle, *cette solution n'est viable que si elle est associée à un mécanisme de signalisation simple et efficace*. C'est grâce à cette signalisation que l'émetteur pourra informer les récepteurs de l'exacte façon dont sont constitués le ou les objets généralisés.

Nous montrons dans ce chapitre que l'approche GOE représente actuellement le meilleur compromis qui soit en terme de performances, simplicité, et compatibilité avec l'existant.

9.3 Gros plan sur les schémas FEC GOE

Présentons plus en détails l'approche GOE. Elle est détaillée dans [82][83] et évaluée dans [86].

9.3.1 Les trois étapes

L'application d'un schéma FEC GOE se fait en trois étapes :

- **Etape 1 - traitement des objets sources** : L'émetteur choisit tout d'abord une taille de symboles qui devra être appliquée à tous les objets de la session[4]. Chaque objet source est alors "encodé" au moyen du schéma FEC No-Code (voir la section 8.2). Il ressort une suite de symboles d'encodage constitués des symboles sources auxquels le "pseudo-encodage" No-Code a ajouté les informations FPI. Chaque symbole de la session est dès lors totalement identifié par le triplet Transport Object Identifier (TOI), SBN, ESI. Les paquets ALC ou NORM contenant ces symboles peuvent dès lors être transmis sur le réseau.

  Cette étape utilise un schéma connu, supporté par tout récepteur ALC, et n'est en aucun cas spécifique à l'approche GOE. Par son biais une compatibilité ascendante est également possible.

- **Etape 2 - création des objets généralisés en fonction du service désiré** : La deuxième étape consiste en la définition des objets généralisés, chacun d'eux étant identifié par un TOI spécifique, distinct des TOI identifiant les objets sources. La définition des objets généralisés est essentielle et conditionne totalement le service fourni.

[4] Cette taille unique (au lieu d'être définie objet par objet, comme c'est le cas dans les schémas FEC classiques) permet de simplifier la signalisation puisque des objets généralisés peuvent être à cheval entre plusieurs objets source.

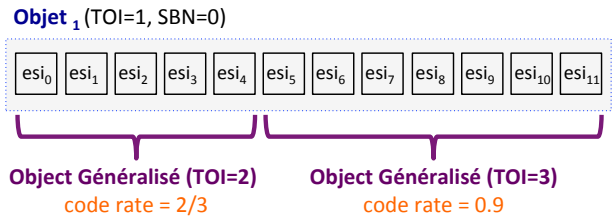


Figure 9.2: Définition des objets généralisés en vue de fournir un **service UEP** dans une approche FEC GOE.

La figure 9.2 montre comment utiliser une approche FEC GOE afin de fournir un service de protection inégale. Les données prioritaires

sont associées à un encodage AL-FEC avec un code rate faible, alors que les données les moins prioritaires bénéficient d'un encodage AL-FEC avec un code rate plus élevé (voire aucun encodage).

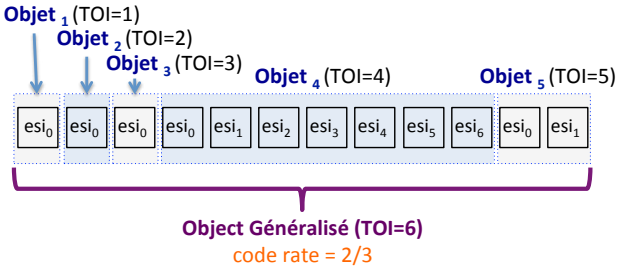


Figure 9.3: Définition d'un objet généralisé en vue de fournir un **service de protection globale** dans une approche FEC GOE.

La figure 9.3 montre comment utiliser une approche FEC GOE afin de fournir une protection globale d'un groupe de cinq fichiers, de taille variable : trois objets de taille un symbole source, un objet de taille deux symboles et un objet de taille sept symboles. Un unique objet généralisé est créé qui englobe l'ensemble des cinq objets source. La protection apportée à chacun des objets source est identique puisque leur sort est commun (l'objet généralisé est soit décodé dans sa globalité soit non décodé).

- **Etape 3 - création et transmission des données de redondance** : La dernière étape consiste en l'encodage AL-FEC de chaque objet généralisé. Suivant les besoins, ceci peut se faire au moyen d'un schéma FEC GOE de type LDPC-Staircase [83], d'un schéma FEC GOE de type Reed-Solomon [82], ou autre. Les symboles source étant déjà "encodés" au moyen du schéma "No-Code", seuls les symboles de redondance sont considérés et transmis en sortie de l'encodage.

Notons que l'approche GOE n'impose aucun ordonnancement des paquets source ou redondance. Liberté totale est laissée à l'émetteur qui peut en profiter pour ajuster sa stratégie en fonction des objectifs suivis. Cet aspect sera détaillé au sein de la section 9.3.4.

9.3.2 Signalisation de niveau objets

La composition de chaque objet source est transmise via le FEC OTI correspondant au schéma "No-Code". De même, la composition de chaque objet généralisé est transmise via le FEC OTI correspondant au schéma FEC GOE utilisé. Ces informations FEC OTI permettent à l'encodeur et au décodeur de partager les informations relatives à chaque objet. Dans le cas d'un objet source, nous trouverons :

- l'identifiant du schéma FEC utilisé,

- les informations de taille d'objet et de symboles.

Dans le cas d'un objet généralisé, nous trouverons de surcroît :

- l'identification précise du premier symbole (ou "Initial Source Symbol") de l'objet généralisé, via le triplet {`ISS_TOI`, `ISS_SBN`, `ISS_ESI`};
- la taille de l'objet généralisé, en nombre de symboles;
- et enfin des paramètres spécifiques aux codes AL-FEC utilisés (dimension et/ou longueur maximum, ou graine du PRNG);

Muni de l'ensemble de ces informations, un récepteur pourra déduire la correspondance précise entre objets sources et objets généralisés ainsi que les paramètres des codes AL-FEC utilisés.

9.3.3 *Signalisation de niveau symboles*

Il n'y a pas de différence au niveau des informations FPI entre un schéma AL-FEC donné et sa version GOE. Ainsi chaque symbole source est transmis au sein d'un paquet source dédié, avec une signalisation de type "No-Code" qui, par le biais des informations FPI, identifie précisément le symbole[5]. De même, chaque symbole de redondance est transmis au sein d'un paquet de redondance dédié, avec la signalisation du schéma FEC GOE utilisé qui, par le biais des informations FPI, identifie précisément le symbole.

[5] Rappelons que les informations FPI sont généralement constituées des identifiants SBN et ESI, le TOI étant pour sa part communiqué via l'entête du protocole ALC/LCT ou NORM.

9.3.4 *Ordonnancement des symboles : permutation aléatoire et Δ-permutation*

Considérons maintenant l'ordonnancement des paquets au niveau de l'émetteur. L'approche GOE n'impose aucune solution, l'émetteur étant totalement libre de choisir la stratégie la plus appropriée. Nous considérons dans l'évaluation deux approches :

- **permutation aléatoire :** la transmission intervient suivant un ordre totalement aléatoire. Un avantage de cette solution est de rendre La transmission robuste face à tout type de modèle de pertes, y compris des pertes en rafales;
- **Δ-permutation :** la transmission intervient dans un ordre partiellement aléatoire, contrôlé par le paramètre Δ. Il est à noter que cette notion de Δ-permutation est similaire à ce que nous avions introduit dans [79] pour le cas d'une application de transfert de fichier, lorsqu'il s'agissait de lisser dans le temps la charge CPU liée au décodage des différents blocs.

  L'approche est la suivante. Soit n_{GOE} le nombre de symboles (ou paquets puisque nous nous limitons au cas où un paquet contient un

seul symbole). Nous permutons chaque paquet d'indice $i * \Delta$ avec un autre paquet choisi aléatoirement parmi les n_{GOE} choix possibles. Le mélange est d'autant plus important que l'on choisit une valeur Δ faible. Le cas extrême correspond à $\Delta = 1$, auquel cas nous retombons sur une permutation totalement aléatoire. Nous verrons par la suite que $\Delta = 3$ est un choix relativement pertinent lorsque l'on veut minimiser le délai de décodage (le mélange n'étant pas total, si les objets transmis en premier sont les plus prioritaires, ils auront une probabilité élevée d'être décodés en premier) tout en se ménageant une certaine robustesse face aux pertes en rafales.

9.4 *Evaluation de performances des approches PET/UOD et GOE*

Nous donnons maintenant des éléments de performance des approches PET/UOD et GOE. Nous nous limitons ici à un résumé, les détails étant disponibles dans [86]. Ces éléments de performance sont obtenus à la fois par simulation et par modélisation.

Paramètres d'entrée	
m, taille de l'objet original	100KB
d, nombre d'objets (ou niveaux priorités)	5
α_i, taille de l'objet i	[10KB, 10KB, 20KB, 30KB, 30KB]
ρ_i, code rate cible pour l'objet i	[0.5, 0.6, 0.65, 0.8, 0.95]
l, taille de paquets cible	250 B
Paramètres de sortie dans le cas UOD/PET	
β_i, nombre de symboles source	[279, 335, 363, 447, 531]
s_i, taille de symboles pour l'objet i	[36 B, 30 B, 56 B, 68 B, 57 B]
taille de paquets effective	247 B
nombre de symboles d'encodage	558
code rate effectif	[0.5, 0.6, 0.651, 0.801, 0.952]
n_{PET}, nombre total de paquets	558
Paramètres de sortie dans le cas GOE	
k_i, nombre de symboles source pour l'objet i	[40, 40, 80, 120, 120]
taille de symboles (tout objet confondu)	250 B
n_i, nombre de symboles d'encodage pour l'objet i	[80, 67, 124, 150, 127]
taille de paquets effective	250 B
code rate effectif	[0.5, 0.597, 0.645, 0.8, 0.945]
n_{GOE}, nombre total de paquets	548

Table 9.1: Paramètres expérimentaux, pour l'étude UEP.

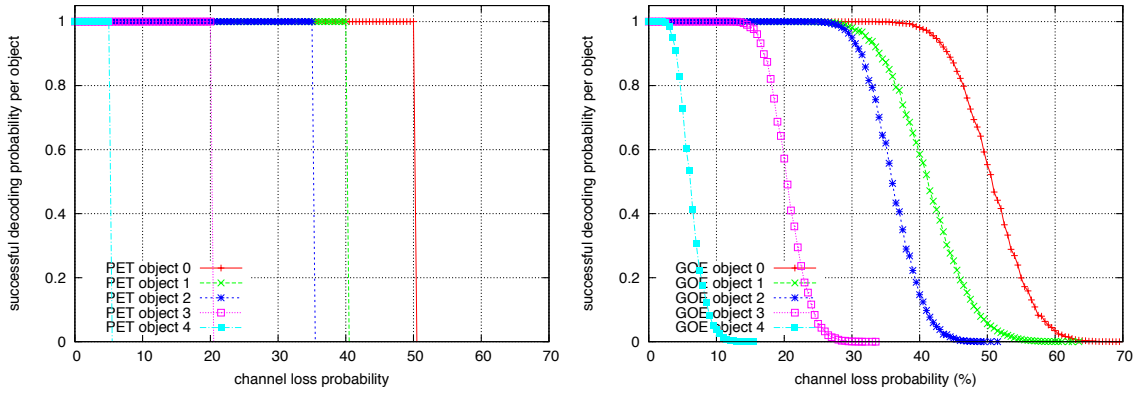
Pour les simulations, nous considérons la situation suivante, extraite de [5] et résumée dans la table 9.1. Un objet source, de taille totale 100 kilo-octets, incluant 5 niveaux de priorités, est considéré. Cet objet source est donc découpé en 5 "objets généralisés" (GOE) ou "objets UOD". La même protection FEC est apportée dans chaque approche UEP, fonction du niveau de priorité : d'un code rate cible 1/2 jusqu'à 0.95. Nous constatons que le code rate effectif obtenu est inférieur ou égal à la cible avec l'approche GOE, alors qu'il est supérieur ou égal

avec l'approche PET/UOD[6]. Ainsi l'approche PET/UOD peut sous protéger les données alors que l'approche GOE peut les surprotéger. Sinon l'examen de cette table nous montre que PET/UOD ne parvient pas à remplir parfaitement les paquets (la taille effective est inférieure de 3 octets)[7].

Analyse critique: *Ces différents soucis de respect des objectifs avec PET/UOD ne sont pas anecdotiques mais significatifs des multiples contraintes qui pèsent sur l'approche : remplir un paquet dont la taille résulte d'une recherche de PMTU au moyen de multiples symboles dont PET/UOD ne contrôle ni le nombre (c'est le nombre de niveaux de priorités) ni la taille (c'est la conséquence du code rate cible et de la taille de chaque objet, avec dans le cas UOD la contrainte supplémentaire d'être multiple de 4 octets) induit nécessairement des erreurs d'arrondis. Rien de tel n'existe avec GOE.*

Regardons maintenant les performances obtenues. Nous nous limitons aux métriques principales que sont le nombre d'objets décodés et le délai nécessaire pour chacun d'eux. Nous faisons l'hypothèse d'un code AL-FEC idéal, capable de décoder parfaitement dès que k symboles quelconques sont reçus. Cette hypothèse permet de comparer les approches PET/UOD et GOE et non les codes associés.

9.4.1 *Performances en terme de nombre d'objets décodés*



(a) PET (b) GOE

Figure 9.4: Nombre d'objets décodés en fonction du taux de pertes avec les approches PET et GOE.

Les figures 9.4 montrent la probabilité que les cinq objets de la session PET (gauche) et GOE (droite)soient décodés lorsque le taux de pertes observé augmente. L'objet le plus prioritaire (rouge), qui a fait l'objet d'un encodage avec un code rate 1/2, est bien évidemment celui qui tolère le plus fort taux de pertes. Ces deux courbes illustrent parfaitement la différence majeure entre les approches : le comportement

[6] La différence entre code rate cible et effectif n'est pas importante dans cet exemple. Il existe cependant des situations où cette différence devient significative avec une approche PET/UOD. Nous exhibons un tel exemple dans `http://www.ietf.org/proceedings/81/slides/rmt-2.pdf`.

[7] Ce problème serait plus important si l'on considérait UOD qui requiert de surcroît que les tailles de symboles soient multiples de 4, du fait des contraintes liées à l'usage des codes RaptorQ.

de PET est *totalement déterministe* et la probabilité de décodage passe immédiatement de 1 à 0 dès que le taux de pertes maximum toléré par la protection FEC est atteint. A l'inverse, l'approche GOE présente une transition "douce" à mesure que l'on s'approche de la valeur seuil. Cette caractéristique n'est pas en soit un soucis, ainsi que nous allons maintenant le montrer.

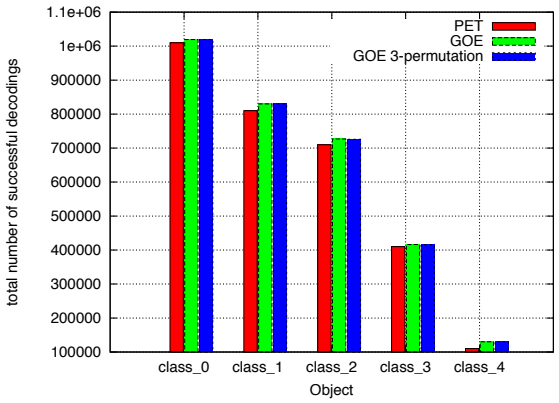


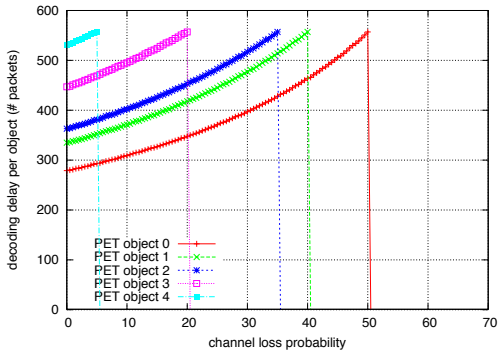
Figure 9.5: Comparaison du nombre total d'objets décodés pour tous les taux de pertes considérés avec les différentes approches UEP.

Déterminer la meilleure approche nécessite de savoir si le nombre de cas n'ayant pas pu être décodés avec GOE avant d'atteindre le seuil est compensé ou non par le nombre de cas ayant pu être décodés après avoir dépassé ce seuil. La figure 9.5 compare donc le nombre total d'objets décodés, sur toutes les simulations effectuées, pour toutes les valeurs de taux de pertes considérées (intervalle $[0;70]$). Nous constatons sur cet exemple des résultats similaires avec un léger avantage pour GOE, que ce soit dans sa version permutation aléatoire ou 3-permutation.

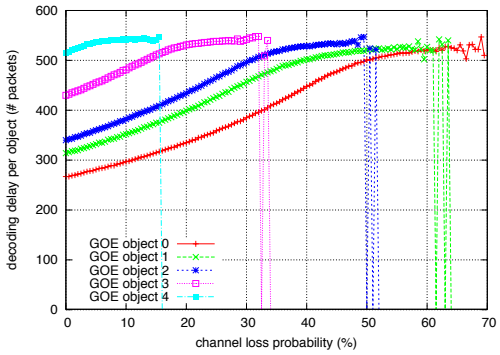
Les performances de capacités de correction sont similaires entre les deux approches. Nous l'avons montré par simulation sur un exemple particulier (avec un léger avantage pour l'approche GOE), nous le démontrons théoriquement dans [86].

9.4.2 *Performances en terme de délai de décodage*

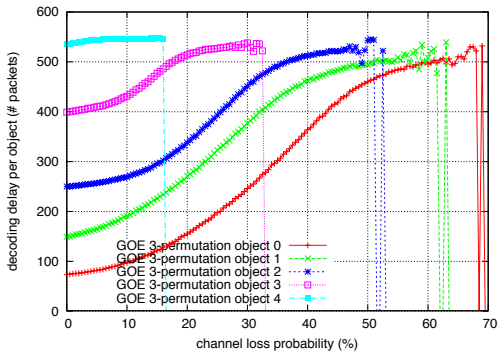
Considérons maintenant la métrique "délai de décodage", qui indique à partir de combien de symboles reçus le récepteur peut décoder un objet donné. Cette métrique est tout particulièrement importante lorsque l'on applique ces approches UEP à des flux temps réels, les objets les plus prioritaires devant être décodés en premier.



(a) PET



(b) GOE



(c) GOE, 3-permutation

Figure 9.6: Nombre d'objets décodés en fonction du taux de pertes avec les approches PET et GOE.

Avec GOE, l'ordonnancement le plus favorable pour le délai de décodage lorsque le taux de pertes est nul consiste à transmettre k_0 symboles source pour l'objet 0 (le plus prioritaire), puis k_1 symboles source pour l'objet 1, et ainsi de suite. En présence de pertes et donc avec un encodage FEC, on peut choisir de transmettre n_0 symboles d'encodage pour l'objet 0, puis n_1 symboles d'encodage pour l'objet 1, et ainsi de suite, ce qui là encore minimisera le délai de décodage pour l'objet 1 (au détriment des autres objets). Ceci correspond à une absence de permutation. Ajouter une permutation augmentera inéluctablement le délai de décodage de l'objet le plus prioritaire, tout en rendant le service plus robuste face à des pertes en rafales. Cependant, et sans surprise, l'approche GOE 3-permutation conservera un avantage certain sur GOE doté d'une permutation totalement aléatoire, ainsi que sur PET. C'est ce que nous montrons ci-dessous.

Nous constatons sur la figure 9.6 un léger avantage de GOE avec une permutation aléatoire sur PET, et un avantage très net pour GOE 3-permutation[8].

La table 9.2 précise les résultats obtenus. Pour un taux de pertes de 25%, nous constatons que l'usage de GOE (resp. GOE 3-permutation) permet un gain relatif $\Delta\% = (359 - 371)/371 = -3.2\%$ (resp. $\Delta\% =$

[8] Les courbes de la figure 9.6 montrent les délais moyens obtenus pour des décodages réussis. Ainsi les situations d'échec ne sont pas considérées, ce qui explique que les courbes GOE s'étendent plus loin vers la droite que celle de PET. Ce phénomène est un effet de bord de la façon de calculer la métrique et ne doit pas être interprété comme un avantage de GOE. Les oscillations observées pour GOE pour les valeurs du taux de pertes élevées sont également expliquées par cette façon de calculer la métrique : lorsque, sur les 1000 tests effectués pour chaque valeur de taux de perte donnée, aucun test n'est parvenu à décoder un objet, la valeur "délai" est alors nulle.

−46.6% (!)) sur PET pour l'objet le plus prioritaire. Si l'on considère l'objet 2, le dernier objet pouvant être décodé avec l'approche PET, le gain relatif obtenu est de $\Delta\% = -3.5\%$ pour GOE (resp. $\Delta\% = -18.6\%$ pour GOE 3-permutation) sur PET. Dans le cas d'un taux de pertes nul, les gains permis avec GOE 3-permutation sont considérables pour les objets les plus prioritaires : $\Delta\% = -73.5\%$ pour l'objet 0.

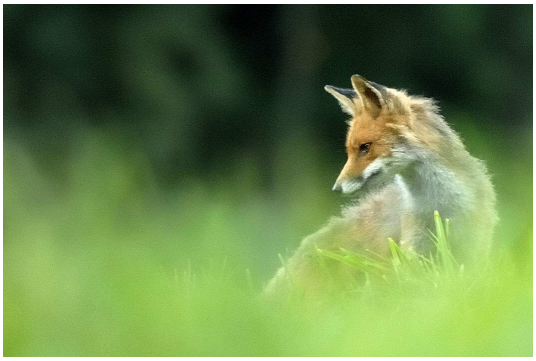
Technique	*Pertes*	*Délai*(O_0)	*Délai*(O_1)	*Délai*(O_2)	*Délai*(O_3)	*Délai*(O_4)
PET	0%	279	335	363	447	531
GOE	0%	267	314	340	430	515
GOE 3-p.	0%	74	149	250	399	535
PET	25%	371	445	483	0	0
GOE	25%	359	427	466	538	0
GOE 3-p.	25%	198	323	393	527	0
PET	50%	557	0	0	0	0
GOE	50%	501	521	0	0	0
GOE 3-p.	50%	460	494	543	0	0

Table 9.2: Délai de décodage, pour chaque objet, pour les différentes techniques UEP.

9.5 *Pour plus d'informations...*

Nos publications relatives au thème :

- "Memory Consumption Analysis for the GOE and PET Unequal Erasure Protection Schemes", A. Roumy, V. Roca, B. Sayadi, et R. Imad, 2012, [85]
- "Unequal Erasure Protection and Object Bundle Protection with the Generalized Object Encoding Approach", A. Roumy, V. Roca, B. Sayadi, et R. Imad, 2011, [86]
  → *Ce rapport de recherche fournit les fondements d'une évaluation de performance comparée des deux approches, à la fois par une modélisation mathématique des deux solution et par voie de des simulation.*
- "The Generalized Object Encoding (GOE) Approach for the Forward Erasure Correction (FEC) Protection of Objects and its Application to Reed- Solomon Codes over GF(2x)", V. Roca, A. Roumy, B. Sayadi, [82]
  → *Ce document de travail spécifie les fondements de l'approche GOE et la signalisation associée, puis l'applique aux codes Reed-Solomon.*
- "The Generalized Object Encoding (GOE) LDPC-Staircase FEC Scheme", V. Roca, A. Roumy, B. Sayadi, [83]
  → *Ce document de travail applique l'approche GOE aux codes LDPC-Staircase.*



Renard, Isère, 2008.

Part III

Protocoles de diffusion robuste de l'information

10
Les approches FLUTE/ALC, FCAST/ALC et FCAST/NORM pour une diffusion robuste d'objets

Résumé :
Ce chapitre décrit le protocole de transport "multicast fiable" ALC/LCT défini par le groupe RMT, ainsi que les applications FLUTE et FCAST qui lui sont associées.

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux menés au sein du groupe RMT de l'IETF, en particulier avec l'aide de C. Neumann. Ils ont également été enrichis par les diverses contributions et discussions qui ont eu lieu au sein de ce groupe, par nous ou par d'autres.*

10.1 Les travaux précurseurs et le groupe RMT

La question des protocoles multicast fiables a fait l'objet de nombreux travaux académiques. Nous pouvons citer :

- les **approches purement bout en bout et équilibrées**, chaque récepteur ayant un rôle identique, avec mécanisme d'acquittements positifs (ACK) / négatifs (NACK) : on peut citer Scalable Reliable Multicast (SRM)[27], ou auparavant Xpress Transport Protocol (XTP)[24], qui prenait en charge soit connexions point-à-point et multipoints. Le soucis principal dans ces approches est de faire face à l'explosion de la source face aux demandes de retransmission (NACK). La solution passe alors par des mécanismes de suppression d'acquittements et de retransmission locales [15][1] (associés bien sûr à des codes AL-FEC [65]);

- une variante des approches bout à bout avec mise en place d'une **hiérarchie de noeuds d'assistance** (les "Designated Receivers", DR), comme cela est fait avec Reliable Multicast Transport Protocol (RMTP) [68]. Les acquittements et retransmissions sont alors pris en charge par ces DR, évitant par là même de remonter à la source;

- les approches faisant intervenir des **équipements réseaux** (routeurs) afin prendre en charge acquittements et retransmissions. La solution Pragmatic General Multicast (PGM) [93], poussée par CISCO,

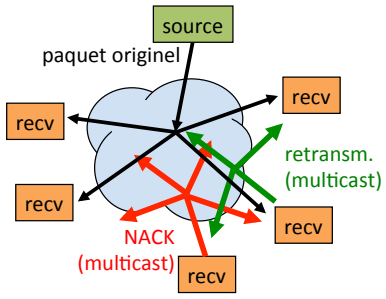


Figure 10.1: Mécanisme de retransmission avec SRM.

[1] Un récepteur ne transmet pas un NACK immédiatement mais attend une durée aléatoire, fonction du nombre estimé de participants. Si la perte n'a pas été récupérée à l'arrivée à échéance du timer, il transmet un NACK en multicast. Un récepteur proche recevant ce NACK et disposant du paquet perdu effectue alors, toujours après une durée aléatoire, la retransmission demandée, là encore en mode multicast afin de pouvoir servir plusieurs récepteurs ayant observés les mêmes pertes (figure 10.1). Par ce biais on espère minimiser l'implication de la source dans le mécanisme de fiabilisation, et donc de garantir un meilleur passage à l'échelle. Malgré tout la solution a des limites intrinsèques, que ce soit en terme de nombre de récepteurs ou bien en termes de prise en charge de l'hétérogénéité de ces récepteurs.

en est un bon exemple. Si cette approche est efficace, les routeurs étant idéalement placés, aux point de branchement en particulier, il est clair que cela rend plus complexe ces derniers et crée des exceptions dans le flux de traitements, difficilement compatible avec les très hauts débits;

- les approches de **type fontaines**, où la fiabilité est atteinte via des transmissions continues au sein d'un carrousel et usage massif de codes AL-FEC [14]. Cette approche est celle qui a donné naissance au protocole ALC/LCT que nous détaillons dans ce chapitre;

L'IETF a lancé en 1999 le groupe RMT afin de standardiser des protocoles multicast fiables. Initialement, trois propositions ont été identifiées, pour au final n'en conserver que deux, ALC/LCT et NORM.[2]

Nous détaillons dans ce chapitre uniquement le protocole ALC/-LCT et laissons de coté la solution NORM. En effet, l'essentiel de nos travaux a concerné les approches "fontaines", qui par leur apparente simplicité (du moins dans l'idée de base) nous semblait plus à même de résoudre efficacement les problèmes posés. Nous nous intéressons ensuite à la construction d'applications de transfert fiable d'objets au dessus de ALC/LCT, via les approches FLUTE et FCAST.

[2] La troisième proposition, poussée par CISCO, incluait une assistance des routeurs afin de cacher et retransmettre le cas échéant les paquets perdus. Après quelques temps, l'effort de standardisation a été interrompu et la proposition a disparu. Est-ce suite à la prise en compte de réelles difficultés techniques rendant l'approche malaisée ?

10.2 Le protocole de transport ALC/LCT

Le protocole de transport ALC/LCT [52][51] répond à un cahier des charges strict :

- la source doit être en mesure de transmettre un ensemble d'objets à un **nombre quelconque de destinataires** avec la même efficacité;
- le protocole doit fonctionner en **l'absence de tout canal de retour**;
- une **haute fiabilité** de réception doit être possible;
- Les modèles de transmission sont variés, de type **"push"**, où des contenus sont transmis à l'ensemble de récepteurs en attente, ou bien **"à la demande"** ("on demand"), où des contenus sont transmis pendant un laps de temps suffisamment long pour permettre à chaque récepteur potentiel de joindre la session et récupérer les contenus.

Ce cahier des charges a donné lieu à la création d'un protocole de type "fontaine" [14] qui "se contente" de générer des flux de paquets, envoyés en mode diffusion ("multicast" ou "broadcast") en espérant que la plupart des destinataires les recevront. Les paquets à transmettre sont stockés au sein de carrousels, la transmission ayant lieu en

boucle, aussi longtemps que la source estime qu'il est souhaitable de le faire, ce qui permet de travailler en mode "push" ou "à la demande".

Un usage massif de codes AL-FEC est fait, afin :

- de fiabiliser les transmissions (par exemple suite à des pertes dues à des phénomènes d'évanouissement lents ou rapides pour une transmission sans fil, ou des problèmes de congestion de routeurs Internet). Il est à noter que cela peut ne pas suffire puisqu'en l'absence de flux de retour il n'est pas possible de connaître l'état de chaque récepteur;
- d'améliorer le service (les récepteurs étant libres d'arriver et partir à leur guise, ils recevront un sous ensemble des paquets transmis durant la session dont la durée est alors nettement supérieure au temps minimum de réception);

La figure 10.2 illustre ce principe avec un scénario de diffusion continue d'une bibliothèque de contenus multimédia vers une flotte de véhicules.

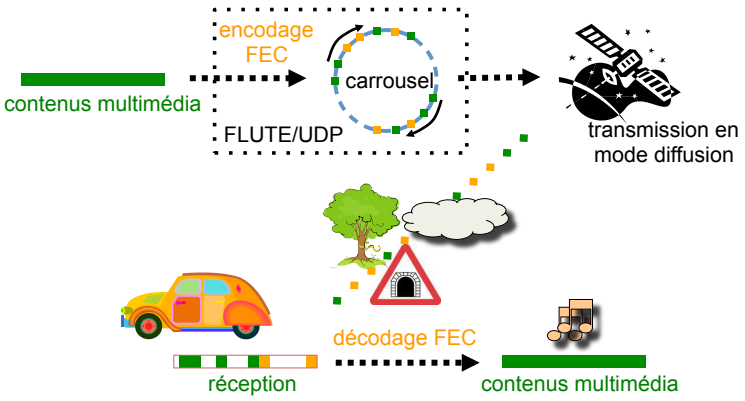


Figure 10.2: Exemple de transmission de contenus multimédias à des véhicules dernier cri au moyen de FLUTE/ALC.

10.2.1 ALC/LCT plus en détails

Formellement parlant, ALC est un protocole de transport qui spécialise plusieurs briques élémentaires, en particulier LCT qui définit les concepts de base ainsi que les entêtes de paquets. Pour simplifier, nous parlerons de ALC pour désigner l'ensemble.

ALC est un protocole de transport relativement simple, dans la mesure où il n'y a pas de mécanisme explicite d'ouverture ou fermeture de session (cela n'aurait pas grand sens puisqu'il n'y a pas de voie de retour, à la différence de TCP par exemple). Il est juste possible de signaler qu'une fin de session ALC est imminente en positionnant le flag `A` dans les derniers paquets transmis. Il est aussi possible de spéci-

fier le temps approximatif restant dans la session ALC via l'extension d'en-tête EXT_TIME (voir ci-dessous).

Au sein de la session, tout objet est identifié de façon unique via son TOI. Il est possible là encore de signaler qu'une fin de transmission de paquets pour un objet donné est imminente, en positionnant le flag B dans les derniers paquets transmis pour cet objet.

Format d'en-tête :

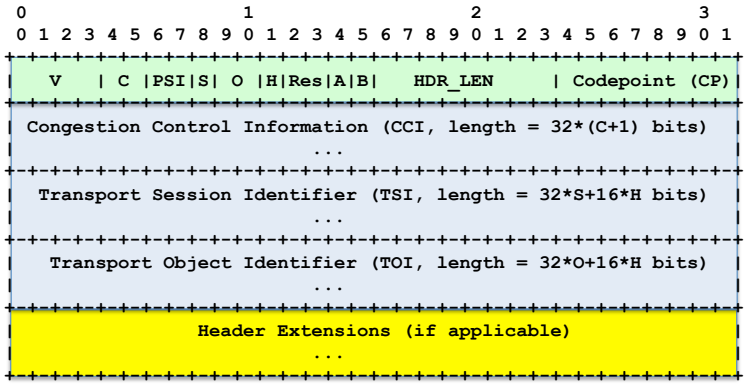


Figure 10.3: En-tête LCT.

L'en-tête (figure 10.3) est constitué d'une partie fixe (32 bits), de trois champs de taille variable, potentiellement suivis d'une ou plusieurs extensions d'en-tête. Ce design a pour objectif d'apporter un maximum de flexibilité au protocole, puisqu'il est possible :

- *d'adapter la taille de certains champs aux besoins :* ainsi le champ TOI qui identifie l'objet auquel appartiennent les données de ce paquet a une taille qui varie de 0 bit (pour un paquet de signalisation pure), à 16 bits, 32 bits, et jusqu'à 112 bits (14 octets) en fonction du nombre d'objets d'une session et de la façon d'attribuer ces identifiants (voir ci-dessous);
- *d'ajouter de nouveaux mécanismes :* par exemple l'extension EXT_AUTH ajoute un service d'authentification/intégrité via l'une des techniques que nous avons spécifiées pour cela [73][78]. FLUTE utilise également cette possibilité en définissant deux extensions, EXT_FDT et EXT_CENC (section 10.3);
- *d'ajouter des informations supplémentaires lorsque c'est pertinent :* par exemple l'extension EXT_TIME permet de transmettre des informations relatives à la durée restante de la session ou de la transmission de l'objet. D'un usage plus important, l'extension EXT_FTI (section 8.3.1) permet de transmettre les FEC OTI d'un objet, telle la File Delivery Table Instance (FDT Instance) de FLUTE;

Gros plan sur l'en-tête ALC/LCT *(figure 10.3) :*

Partie fixe :

- *le flag A indique une fin imminente de la session ALC. C'est un flag qui est présent dans le ou les derniers paquets transmis pour une session;*
- *le flag B indique une fin imminente de la transmission de paquets pour un objet. C'est un flag qui est présent dans le ou les derniers paquets transmis pour un objet;*
- *le champ Codepoint en général est utilisé pour transporter le FEC Encoding ID utilisé pour l'objet auquel est associé ce paquet. Il est à noter que d'autres usages de ce champ sont possibles;*

Puisque rien ne garantit qu'un récepteur recevra les paquets contenant les flags A ou B (ce récepteur peut être déconnecté à ce moment là, ou ne plus avoir de connectivité), il faut être prudent. En général un autre mécanisme de signalisation, de niveau session, est là pour fournir ce service.

Champs de taille variable :

- *le champ CCI est utilisé dans le cas où l'usage d'un protocole de contrôle de congestion est requis. Sur Internet, le protocole Wave and Equation Based Rate Control (WEBRC) [47] doit être utilisé à cet effet bien qu'il ait un niveau de maturité plus faible (Expérimental) et que peu de retours d'expérience soient disponibles. A noter les travaux de V. Lucas sur le sujet [53]. En pratique, du fait de la complexité du protocole WEBRC, nous avons préféré Receiver Driven Layered Congestion-Control (RLCC) et Fair Layered Increase/Decrease with Static Layering (FLID-SL) dans notre implantation d'ALC*[3]*. Cependant, la plupart des déploiements d'ALC se font sur des canaux totalement maîtrisés, de débit constant, pour lesquels aucun contrôle de congestion n'est requis. Dans ce cas, ce champ reste à zéro*[4]*;*
- *le champ TSI associé à l'adresse IP source identifie la session de façon unique;*
- *le champ TOI identifie l'objet auquel appartiennent les données du paquet ALC. Ce champ est délibérément de taille variable, potentiellement jusqu'à 112 bits (14 octets) afin d'être alloué soit de façon séquentielle, soit par tirage aléatoire, soit par hachage d'un élément de l'objet;*

[3] Le point fort de WEBRC vient de la prise en compte de la latence Internet Group Management Protocol (IGMP)/Multicast Listener Discovery (MLD) lors d'un désabonnement à un groupe multicast, ce qui réduit d'autant la réaction face à une situation de congestion. L'usage de transmissions "par vagues" permet dès lors de s'affranchir de ce soucis en reportant le problème de latence à l'adhésion à de nouveaux groupes afin de maintenir un débit de réception constant. C'est aussi la principale cause de complexité de ce protocole. Il est à noter que l'usage systématique d'une approche de suivi explicite des récepteurs intéressés par un flux multicast (par exemple [6]) permettrait de réduire grandement de cette latence.

[4] Il n'est pas possible de retirer le champ CCI, même si aucun contrôle de congestion n'est requis.

10.3 L'application FLUTE

L'application FLUTE est construite pour travailler en relation avec ALC/-LCT (figure 10.5). Elle apporte les mécanismes supplémentaires requis pour construire le service de transmissions de "fichiers" (au sens large).

En particulier, FLUTE permet de décrire dynamiquement les fichiers transmis au sein de la session. Pour cela des informations (méta-données) sont fournies par FLUTE pour chaque objet ALC transmis à ce moment là, ce qui permet à un récepteur, après en avoir pris connaissance, de choisir un sous ensemble de fichiers qui l'intéressent parmi ceux transmis et d'ignorer les autres. Le récepteur a également en sa possession les méta-données nécessaires pour traiter les fichiers reçus, tels que le nom et le répertoire où écrire le fichier, son type, voir l'application associée.

Figure 10.4: Une superbe flûte de facteur (J.L. Boudreau), de type soprano renaissance LA:440.

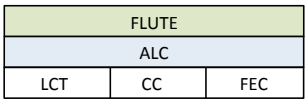


Figure 10.5: Pile protocolaire pour l'application FLUTE.

10.3.1 La File Delivery Table, ou FDT

A la base de FLUTE se trouve la notion de FDT, ou "table de livraison de fichiers". Il s'agit d'un objet ALC spécifique, impérativement identifié par le `TOI` 0, qui est destiné à contenir la liste des fichiers de la session et leurs méta-données, le tout sous forme XML. Comme la session est potentiellement dynamique, de nouveaux fichiers pouvant être ajoutés et d'autres retirés, cette table doit pouvoir évoluer dans le temps, d'où la notion d'instance de FDT, ou FDT Instance. Ainsi un deuxième identifiant est introduit, le "FDT Instance ID", qui permet d'identifier chaque instance de façon unique[5] Tout élément `<File />` de la table doit contenir au minimum un attribut TOI, qui est l'identifiant de base au niveau ALC, et un attribut `Content-Location`, qui est l'identifiant de base au niveau FLUTE. Il est également indispensable qu'une date limite d'usage soit associée à chaque instance. C'est le but du champ `Expires` qui fournit une date (au format Network Time Protocol (NTP)) au delà de laquelle il est indispensable de ne plus tenir compte de cette instance.

FLUTE ajoute deux extensions d'en-tête à LCT :

- `EXT_FDT` qui permet d'indiquer la version de FLUTE (1 pour le RFC 3926, et 2 pour le RFC 6726), ainsi que le `FDT Instance ID`;
- `EXT_CENC` qui permet d'indiquer l'encodage (par exemple via `gzip`) qui pourrait être utilisé pour la FDT Instance.

[5] Une réutilisation de l'identifiant d'instance FDT est possible à condition que le précédant ait expiré.

Exemple :
L'exemple de la figure 10.6 montre une session FLUTE constituée de deux objets ALC à ce moment là, contenant une page Hyper-Text Markup Language (HTML) pour l'un, un fichier `mp3` *pour l'autre. Ces objets sont*

respectivement associés aux TOI 1 et 2, les `Content-Location` *sont bien évidemment spécifiés (obligatoire) et un hashé Message Digest 5 (MD5) est fourni pour le deuxième objet, entre autres choses.*

Figure 10.6: Exemple d'instance de FDT.

```
<?xml version="1.0" encoding="UTF-8"?>
        <FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:fl="http://www.example.com/flute"
            xsi:schemaLocation="http://www.example.com/flute-fdt.xsd"
            Expires="2890842807">
            <File    Content-Location="www.example.com/menu/tracklist.html"
                TOI="1"
                Content-Type="text/html"/>
            <File    Content-Location="www.example.com/tracks/track1.mp3"
                TOI="2"
                Content-Length="6100"
                Content-Type="audio/mp3"
                Content-Encoding="gzip"
                Content-MD5="Eth76GlkJU45sghK"/>
        </FDT-Instance>
```

10.3.2 Limites

Une caractéristique fondamentale de FLUTE est que les instances de FDT sont transmises indépendemment des objets. De ceci découlent à la fois les avantages et inconvénients de FLUTE. C'est un avantage car un récepteur disposant des méta-données d'un objet peut décider ou non s'il est intéressé. Si cela ne l'empêchera pas de recevoir les paquets ALC associé à un objet non retenu, il pourra s'en défaire dès le traitement de l'en-tête ALC, voyant que le TOI est celui d'un objet qui ne l'intéresse pas. Les traitements sont donc limités.

En revanche, traiter un objet et les paquets associés ne peut se faire que si une au moins des FDT Instance le décrivant a été reçue et traitée. Il y a une dépendance forte entre objets et table FDT. Puisque les FDT Instance sont transmises comme de simples objets, au même titre que les autres objets de la session, il faut mettre en place une stratégie permettant de garantir, autant que possible, la bonne réception d'une FDT Instance avant le début de réception des objets qu'elle décrit[6].

Enfin une dernière limite de FLUTE est qu'il est intrinsèquement lié à ALC du fait de choix de conception. FLUTE ne pourra pas fonctionner avec NORM, l'autre protocole de transport multicast fiable basé pour sa part sur des échanges bidirectionnels et des acquittements positifs/négatifs.

[6] En fonction du champ d'application précis et afin de faire face à cette limite, on peut choisir de transmettre périodiquement la dernière FDT Instance, mais cela induit un certain overhead de transmission. On peut également choisir de protéger une FDT Instance de taille élevée, transmise au sein de plusieurs paquets, avec un encodage AL-FEC. On peut enfin choisir de bufferiser un certain nombre de paquets d'un TOI pour lequel on n'a pas encore de description. On s'expose alors à des attaques par saturation (l'attaquant génère un grand nombre de paquets pour des TOI n'existant peut être pas), que seul peut limiter l'instauration d'une borne maximum à ce nombre de paquets/objets en attente.

10.4 L'application FCAST

Pour faire face à ces limites et permettre un usage indifférent avec ALC et NORM, nous avons défini une solution allégée, FCAST, qui

est basée sur une idée initialement détaillée en 2000 dans l'un des tous premiers documents décrivant ALC [46]. L'idée est la suivante : *tout objet est transmis avec ses méta-données, ces dernières y étant attachées au sein d'un "objet composite" ("compound object")* (figure 10.7).

Bénéfices et limites :

L'avantage immédiat est que les inefficacités de FLUTE liées à la séparation entre méta-données et objets sont supprimées : avec FCAST les données qui sont naturellement liées sont protégées (par encodage FEC), transmises et décodées ensembles. En revanche, un récepteur ne peut lire les méta-données d'un objet qu'une fois l'objet composite reçu et décodé.

En revanche il faut faire l'hypothèse qu'un objet transmis dans une session intéressera l'ensemble des récepteurs, puisqu'il n'y a pas de moyen de connaître les méta-données avant que l'objet ne soit reçu (au moins les premiers octets)[7].

Cet inconvénient n'est pas rédhibitoire cependant et il reste possible de le contourner au moyen d'une indirection supplémentaire : une première session FCAST de "contrôle" a pour but de transmettre, au sein d'un premier objet, les méta-données des autres objets, ainsi que les TOI associés; puis une deuxième session de "données", que l'on rejoint une fois avoir récupéré les méta-données, permet au récepteur de sélectionner directement les paquets intéressants. Une variante consiste à définir plusieurs sessions "données", une par objet, afin de permettre à un récepteur de ne recevoir que les paquets des objets qui l'intéressent.

[7] Rappelons qu'avec FLUTE, un paquet qui appartient à un objet qui n'intéresse pas un récepteur sera éliminé plus tôt, après traitement de l'en-tête LCT, sans avoir été stocké et sans qu'il y ait de décodage de l'objet.

10.4.1 *Objets composites et de carrousel*

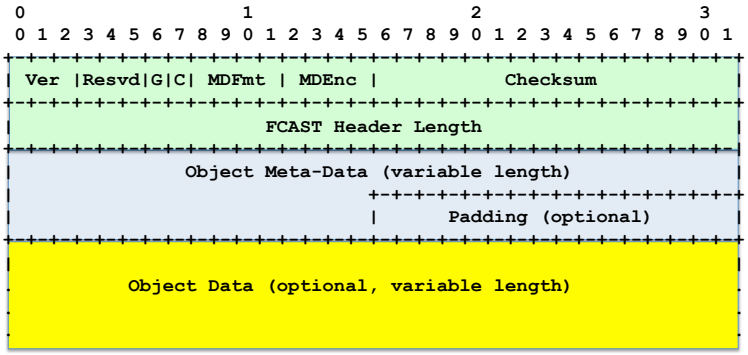


Figure 10.7: Objet composite et en-tête FCAST.

Comme dit précédemment, la notion d'objet composite est centrale à FCAST. La figure 10.7 montre l'en-tête associé à un objet composite :

- La première partie (64 bits) est l'en-tête FCAST;
- La deuxième partie comporte les méta-données de l'objet original;
- La troisième partie est constituée par l'objet original, éventuellement après un encodage au sein de FCAST (par exemple une compression GNU Zip (GZIP)) qui est alors signalée via les méta-données);

Notion de carrousel :

FCAST rend explicite la notion de carrousel de transmission, c'est à dire d'un système de transmission où sont stockés les paquets ALC à transmettre, après encodage AL-FEC le cas échéant, pour les différents objets composites en cours de transmission. Ce carrousel est dynamique, comme pour FLUTE. Le mode "push" pourra faire l'objet d'une unique transmission de ce jeu de paquets, alors que le mode "à la demande" sera fourni au moyen de transmissions cycliques au sein du carrousel, éventuellement après modification de l'ordre.

Un objet spécifique : le descripteur de carrousel :

Le contenu d'une session FCAST, à savoir la liste des TOI des objets en cours de transmission à ce moment là, peut lui même être transmis au sein d'un objet dédié : le "Carousel Instance Descriptor". Cet objet spécifique, qui est lui même un objet composite, contient donc cette liste des TOI. Ce peut être un moyen commode pour un récepteur de déterminer s'il a fini la réception de tous les objets existants et s'il peut quitter la session[8].

[8] Une méta-donnée, "Fcast-CID-Complete: 1", indique qu'aucun nouvel objet ne sera transmis dans le futur.

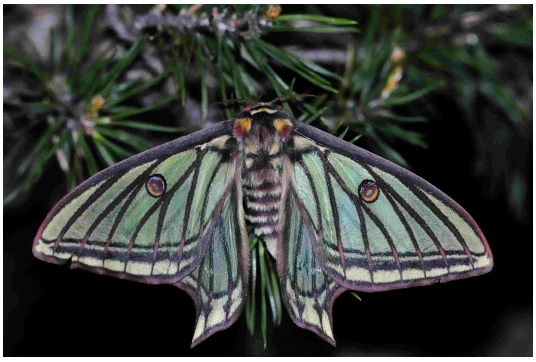
10.5 *Pour plus d'informations...*

Nos publications relatives au thème :

- **[RFC 3926]** "FLUTE - File Delivery over Unidirectional Transport", T. Paila, M. Luby, R. Lehtonen, V. Roca, R. Walsh, [66]
  → *Ce protocole est désormais une des pièces maitresses des standards de diffusion fiable de contenus à destination de smartphones du 3GPP et similaires.*
- **[RFC6726]** "FLUTE - File Delivery over Unidirectional Transport", T. Paila, R. Walsh, M. Luby, V. Roca, R. Lehtonen, [67]
  → *Ce document est une mise à jour du RFC 3926 qui tire partie de l'expérience acquise.*
- **[RFC6968]** "FCAST: Object Delivery for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", V. Roca, B. Adamson, [74]

→ *Ce document décrit une solution alternative à FLUTE, à la fois plus simple dans sa conception et plus large d'application.*

- "Large Scale Content Distribution Protocols", C. Neumann, V. Roca et R. Walsh, 2005, [64]
  → *Cet article introduit les approches RMT et donne quelques éléments d'informations sur les codes AL-FEC associés.*



Isabelle, tu es la plus belle, Hautes Alpes, mai 2011.

11
L'approche FECFRAME pour la diffusion robuste de flux continus de données

Résumé :
Ce chapitre est une introduction à l'architecture FECFRAME de protection de flux continus de données. Le but est de fournir une architecture flexible pour l'utilisation de schémas AL-FEC pour protéger un ou plusieurs flux, souvent caractérisés par des contraintes temps réelles.

Crédits : *Les éléments présentés dans ce chapitre sont issus de travaux menés avec l'aide de M. Cunche, J. Lacan, K. Matsuzono et F. Schuler au sein du groupe FECFRAME de l'IETF. Ils ont également été enrichis par les diverses contributions et discussions qui ont eu lieu au sein de ce groupe, par nous ou par d'autres.*

11.1 Codes AL-FEC et flux temps réel

Les codes AL-FEC ont depuis longue date été appliqués à la protection de flux temps réels [69]. La raison en est simple : les codes FEC en mode bloc n'introduisent que la latence liée au temps couvert par un bloc de symboles source, dont la taille est un paramètre clef qui sera ajusté en fonction du délai maximum permis (voir la discussion de la section 8.4)[1].

Si les travaux de standardisation à l'IETF ont été nombreux sur RTP ainsi que sur les différents formats de contenus supportés, seul [44] définit l'usage de codes FEC dans RTP, au travers d'un code XOR 1-D : à k symboles sources on ajoute un unique symbole de redondance qui est la somme XOR de ces k symboles. Cette protection triviale n'est bien sûre adaptée qu'à des flux bas débit et pour des taux de pertes faibles, sur un canal IID de préférence. C'est ce qui a motivé le travail sur FECFRAME.

11.2 FECFRAME en quelques mots

L'architecture FECFRAME définit des techniques génériques permettant d'insérer des schémas AL-FEC afin de protéger des pertes de transmission, des flux applicatifs consistant chacun en une séquence d'ADUs (autre nom des paquets source générés par une application)[2].

[1] D'autres mécanismes sont utilisés en pratique : un Macro Block (MB) manquant d'une trame vidéo peut être rempli en exploitant la redondance temporelle [102]; des données manquantes peuvent être retransmises si le Round Trip Time (RTT) le permet; l'encodage vidéo peut être conçu pour être robuste face aux pertes (le cas extrême est MJPEG2000, le standard utilisé pour le cinéma numérique, qui n'utilise pas de compression temporelle pour une robustesse maximale : une corruption se limite toujours à une seule trame); ou enfin les techniques "d'inpainting" qui reconstruisent (bientôt en temps réel, c'est l'objectif) les MB perdus en utilisant les informations de voisinage spatiales et temporelles [32].

[2] Par exemple, grâce à FECFRAME, une application de streaming ajoute des paquets de redondance au flux de paquets RTP/UDP transmis. L'application réceptrice peut alors tirer partie des paquets de redondance pour recouvrer des pertes de paquets RTP/UDP sources et passer les ADUs ainsi décodées à l'application, comme si elles avaient été normalement reçues.

Nous trouverons souvent FECFRAME au sein d'une pile RTP/UDP, à la jonction de ces deux protocoles, afin de protéger des flux temps réels. En revanche ceci n'a aucun caractère obligatoire. Rien dans FECFRAME ne restreint la nature des flux applicatifs à RTP, ni au fait que les flux aient des contraintes temps réelles. De même, FECFRAME est indépendant du protocole de transport. Ce peut être UDP ou n'importe quel autre protocole de transport dans la mesure où les frontières entre unités de données sont préservées (Datagram Congestion Control Protocol (DCCP) par exemple [39]).

Enfin une instance FECFRAME peut être utilisée au sein de l'hôte sur lequel est exécuté la (ou les) application(s), ou bien au sein d'une "middlebox" qui par exemple agrège et protège plusieurs flux applicatifs.

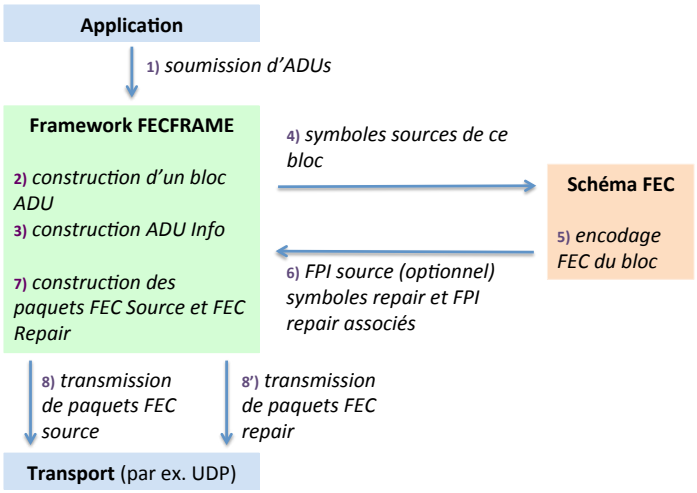


Figure 11.1: L'architecture FECFRAME vue de l'émetteur.

La figure 11.1 montre l'architecture FECFRAME coté émetteur dans le cas d'un unique flux applicatif (le schéma est similaire coté récepteur). On y trouve :

- l'application, qui pourra par exemple générer un flux de paquets au format RTP, chaque paquet constituant une ADU[3]. Si cette figure ne montre qu'une application, il peut y en avoir plusieurs en pratique;
- le framework FECFRAME qui est à l'interface entre application et transport. Une instance de FECFRAME reçoit le ou les flux applicatifs, définit des blocs sources (en tenant compte des contraintes temps réel éventuelles), communique avec le ou schémas AL-FEC, construit et soumet au transport les paquets FEC source et redondance;
- le schéma AL-FEC reçoit du framework un ensemble de symboles sources, issus du bloc source constitué d'ADUs auxquelles on aura

[3] Nous verrons (section 11.3.4) que certains schémas FEC particuliers pour FECFRAME peuvent être restreints à un flux RTP et en tirer profit. Ceci n'est qu'un cas particulier et non la règle générale, d'autant que cette solution induit d'autres limites.

ajouté quelques informations (section 11.3.2), puis effectue l'encodage FEC. Il transmet alors en retour les symboles de redondance créés, ainsi que les information FPI associées aux futurs paquets FEC source et redondance;

- le protocole de transport, qui bien souvent sera UDP. Ce protocole se charge de l'émission des paquets issus de l'instance FECFRAME, en mode datagramme, donc en préservant leurs frontières.

En revanche FECFRAME ne spécifie aucun mécanisme de rétroaction permettant de tenir compte de la qualité du canal de transmission afin d'adapter les paramètres d'encodage (taille de bloc et code rate). Ceci est laissé à d'autres protocoles le cas échéant (section 11.4).

11.3 *Gros plans*

Nous apportons maintenant des éclaircissements sur certains points techniques.

11.3.1 *Utilisation dans l'hôte ou dans un équipement intermédiaire*

Ainsi que nous l'avons mentionné, FECFRAME peut être déployé de deux façons :

- au sein de l'hôte, dans lequel se trouvent les applications (figure 11.2).

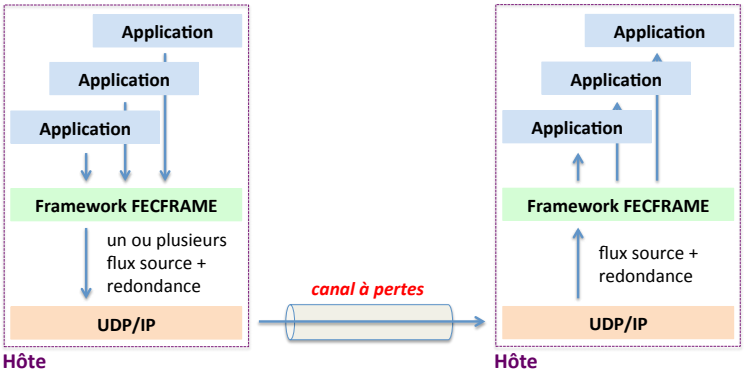


Figure 11.2: Déploiement de FECFRAME dans l'hôte.

Cette première configuration limite les équipements à déployer et garantit qu'aucune perte n'aura lieu entre application et instance FECFRAME. Par contre elle peut nécessiter une modification de l'application afin de confier une ADU à l'instance FECFRAME avant tout appel à la socket UDP (dans certains cas, l'usage de TUN/TAP (`http://en.wikipedia.org/wiki/TUN/TAP`) peut éviter cette contrainte).

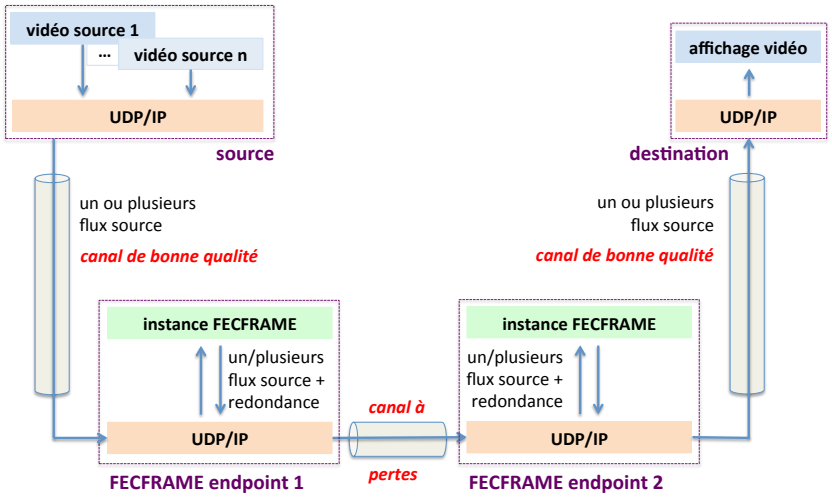


Figure 11.3: Déploiement de FECFRAME dans un équipement intermédiaire.

- au sein d'un équipement intermédiaire ("middlebox") (figure 11.3). Cette deuxième configuration est utile lorsqu'une middlebox agrège plusieurs flux applicatifs (par exemple issus de multiples serveurs de diffusion) afin de les protéger globalement avant de les retransmettre sur un canal à pertes (effectuer une protection globale permettra en général de travailler sur des blocs plus gros, et d'offrir une meilleure protection face à des situations de pertes en rafale). Elle sera également utile pour apporter une protection sans modification des applications (par exemple dont on n'a pas le code source) ou des équipements terminaux (serveur et clients).

  Par contre il est impératif que le canal entre application(s) et l'instance FECFRAME soit de bonne qualité, puisque le trafic n'y est pas protégé[4].

Les figures 11.2 et 11.3 font l'hypothèse d'une communication point à point. En pratique le trafic peut être destiné à un groupe multicast (ASM ou SSM), ou diffusé en mode broadcast. Les exemples ne font pas non plus état d'un quelconque trafic de retour. Celui ci peut exister et prendre la forme d'un flux Real-Time Control Protocol (RTCP) dans le cas où l'application repose sur le protocole RTP.

[4] C'est tout particulièrement vrai dans une approche RTP FEC (section 11.3.4) puisque l'on a des difficultés à signaler la composition exacte du bloc source dès lors qu'il y a des trous dans la numérotation RTP (voir les discussions sur les aspects "operation and management" du [104]).

11.3.2 *Protection de plusieurs flux applicatifs : impacts sur la signalisation*

FECFRAME peut indifféremment être utilisé pour protéger un ou plusieurs flux applicatifs. Dans ce deuxième cas se pose alors le problème de savoir comment une ADU perdue durant la transmission et décodée par l'instance FECFRAME du destinataire peut être associée au bon

flux. Ce problème n'est pas si trivial qu'il y parait car la procédure normale pour associer un paquet reçu à un flux applicatif tire partie des entêtes UDP/IP (par exemple l'adresse IP multicast destination et le port UDP destination). Or l'encodage AL-FEC se faisant avant d'entrer dans la pile UDP/IP, ces informations ne sont pas disponibles à ce moment là, et lorsque l'ADU est décodée aucun lien direct ne pourra être fait.

Un autre soucis, lorsqu'une ADU perdue est décodée par le destinataire, est de connaître la taille d'origine de l'ADU[5].

La solution à ces problèmes nécessite de passer par **l'ajout explicite d'informations à chaque ADU avant l'encodage AL-FEC**. Plus précisément, l'instance FECFRAME ajoute à chaque ADU :

- `FID` (Flow IDentifier) : ce champ de 8 bits identifie le flux applicatif auquel l'ADU appartient[6];
- `L[i]` (Length of i^{th} ADU): ce champ de taille 16 bits précise la taille de la i^{eme} l'ADU. Ceci est rendu nécessaire par le fait que les symboles source qui sont transmis au schéma FEC sont nécessairement de même taille.
- `Pad[i]` : ce champ de taille variable est un bourrage à zéro qui est ajouté en fin d'ADU afin que le tout, `FID; L[i]; ADU; Pad[i]` soit de la taille désirée;

Le tout, `FID; L[i]; ADU; Pad[i]`, est appelé Application Data Unit Information (ADUI) est constitue un symbole source dans le cas des schémas LDPC-Staircase [75] et Reed-Solomon [76] que nous avons définis. Dans certains schémas un découpage de l'ADUI en plusieurs symboles source peut intervenir, mais il semble que cette solution fasse l'objet d'un brevet.

[5] Les ADUs n'ont aucune raison d'être de taille constante. La taille d'une ADU est définie par l'application, hors contrôle, et peut varier largement au fil du temps. Ainsi, dans le cas d'un flux vidéo, la taille des trames qui constituent les ADU variera largement en fonction de leur nature (et du flux avant compression. Seuls les symboles manipulés par le schéma FEC sont de taille constante.

[6] Du fait de la taille du champ FID, le nombre maximum de flux applicatifs pris en charge par une instance FECFRAME est de 256.

11.3.3 Les paquets FEC source et de redondance

Les paquets transmis sur le réseau sont appelés "paquet FEC source" et "paquets FEC de redondance" :

- les paquets source sont, avec la plupart des schémas FEC, constitués de l'ADU (ou d'une fraction d'ADU pour certains schémas FEC) à laquelle on ajoute un "trailer" spécifique, appelé *Explicit Source FEC Payload ID*. Il comportera en général l'identifiant du bloc SBN et l'identifiant de symbole ESI associé, ce qui est indispensable afin de pouvoir l'identifier au sein du bloc source;
- les paquets de redondance sont toujours constitués d'un symbole de redondance auquel on ajoute un entête, appelé *Repair FEC Payload ID*. Il comportera en général l'identifiant du bloc SBN et l'identifiant

de symbole ESI associé, là aussi pour pouvoir l'identifier, auquel on ajoute des informations portant sur la constitution du bloc source[7]. La réception d'un symbole de redondance permet dès lors de reconstituer le bloc source et d'effectuer un décodage si besoin est.

[7] La constitution du bloc source peut varier pour chaque bloc puisque dépendant de paramètres connus au dernier moment. Elle ne peut faire l'objet d'une signalisation globale, valable pour toute la session.

La section 8.3.2 donne de plus amples informations et fournit un exemple.

Nous voyons que la plupart des flux FECFRAME imposent de modifier les paquets sources en y ajoutant un trailer spécifique. Ceci a pour effet direct de casser la compatibilité ascendante dans le cas d'un déploiement de FECFRAME dans l'hôte, un récepteur ne connaissant pas ce protocole étant incapable de tirer partie des paquets source reçus.

De plus, par défaut les paquets de redondance issus d'une instance FECFRAME ne bénéficient pas d'une encapsulation RTP (même si le flux applicatif est un flux RTP). Ceci n'est pas nécessaire dès lors qu'un récepteur est en mesure de distinguer ces paquets de redondance des paquets source. Plusieurs solutions sont possibles pour cela, et en général on transmettra le flux source et le flux de redondance sur des adresses IP multicast différentes, ou à des numéros de port destination différents. Le démultiplexage est alors possible.

11.3.4 *Faut il un format RTP pour les paquets FEC de redondance ? De l'importance de la compatibilité ascendante*

Cependant, il existe des situations où il est essentiel :

- de ne pas modifier les ADUs, qui seront donc strictement identiques aux paquets FEC source;
- de transmettre paquets source et paquets de redondance à la même destination. Le démultiplexage nécessite alors d'examiner le contenu de ces paquets, une fois les entêtes UDP/IP retirés;
- de permettre à un récepteur ne supportant pas le schéma AL-FEC utilisé par l'émetteur de détruire silencieusement les paquets FEC de redondance, pour n'exploiter que les paquets FEC source;

Ces précautions permettent alors de bénéficier d'une **compatibilité ascendante**, le même flux pouvant à la fois être exploité par un récepteur d'ancienne génération (qui se contentera alors des ADU sans aucune possibilité de recouvrer des pertes de paquets) et un récepteur de nouvelle génération (qui pourra exploiter les paquets FEC de redondance).

Ces propriétés peuvent être satisfaites si l'on doit protéger un **unique flux applicatif au format RTP**. L'entête RTP des paquets source sera alors également utilisé par FECFRAME pour son champ `sequence number` qui jouera le rôle d'ESI, et le champ `Payload Type` qui identifiera le flux source. Il n'y a pas dans ce cas d'en-tête "Source FEC Payload

ID". Les paquets FEC de redondance générés seront également au format RTP, avec un champ `Payload Type` spécifique qui les distinguera des paquets sources. Derrière cet entête RTP, on trouvera un entête "Repair FEC Payload ID" qui fournira les informations nécessaires à l'identification de ce paquet de redondance et à la constitution du bloc source[8].

Moyennant ces précautions, un récepteur ne supportant pas FECFRAME sera naturellement amené à ignorer (détruire) les paquets RTP reçus dont le champ `PT` contient un type inconnu, donc les paquets FEC de redondance, et à exploiter les autres paquets RTP correspondant aux ADUs.

[8] Un paquet FEC de redondance sera donc constitué, depuis l'extérieur, des entêtes : IP, UDP, RTP, Repair FPI, le tout suivi du symbole de redondance.

11.4 *Exemple de mise en oeuvre avec retour d'informations*

L'architecture FECFRAME est conçue pour n'imposer aucun mécanisme de rétroaction qui permettrait de tenir compte de la qualité du canal de transmission afin d'adapter en temps réel les paramètres d'encodage FEC, en particulier la taille de bloc et le code rate. Si malgré tout une adaptation est nécessaire, nous retrouverons donc le composant FECFRAME au sein d'un ensemble plus vaste.

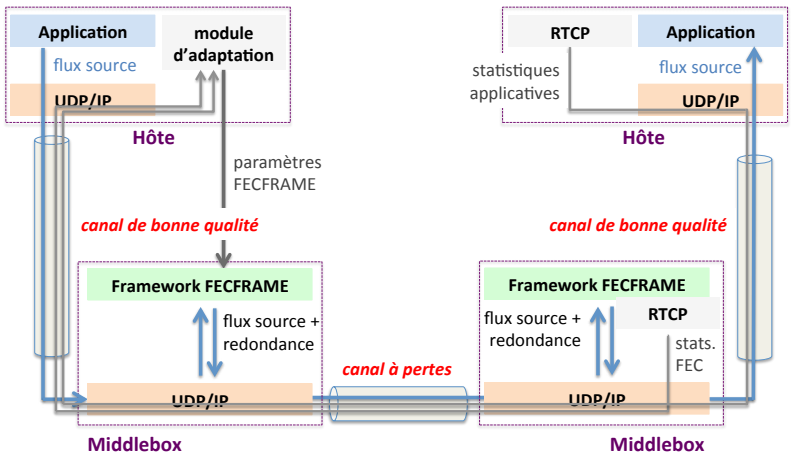


Figure 11.4: Exemple de déploiement de FECFRAME en mode tunnel avec flux de retour et adaptation.

Prenons l'exemple d'un tunnel FECFRAME protégeant un flux vidéo au format RTP, en mode middlebox (figure 11.4). Un flux de retour RTCP peut être généré par l'application réceptrice. Dans ce cas une imprécision reste puisqu'il s'agit de statistiques coté application réceptrice, après décodage FEC. Aucune statistique sur le canal ne pourra être remonté et le paramétrage de l'extrémité d'encodage du tunnel FECFRAME devra être fait prudemment. Il est également possible (en remplacement ou en complément) de générer un retour RTCP au sein de l'instance FECFRAME de réception. Dans ce cas les statis-

tiques pourront inclure des informations avant et après décodage FEC, plus précises, permettant donc un meilleur paramétrage de l'encodeur FECFRAME.

Un module d'adaptation est présent à la source. Il reçoit les informations de retour des entités de réception et en déduit le paramétrage le plus adapté. Il peut être implanté sur la machine source, au niveau de l'application générant le flux RTP (par exemple si l'on utilise des paquets RTCP de l'application réceptrice). Il peut être implanté sur la machine intermédiaire où est implanté l'extrémité d'encodage du tunnel FECFRAME (par exemple si l'on utilise des paquets RTCP du décodeur FECFRAME). Ce module d'adaptation analyse ces statistiques et, ayant connaissance du flux applicatif, agit sur la configuration de l'instance FECFRAME, et le cas échéant sur le flux source (par exemple en changeant l'encodage vidéo). L'intelligence est donc externe à FECFRAME, ce dernier se contentant d'effectuer la création des blocs source et leur encodage/décodage.

11.5 *Pour plus d'informations...*

Nos publications relatives au thème :

- **[RFC 6363]** "Forward Error Correction (FEC) Framework", M. Watson, A. Begen et V. Roca", 2011 [104]
  → *C'est le standard de référence pour l'architecture FECFRAME, qui décrit comment protéger des flux continus au moyen de codes AL-FEC.*

- "Performance Analysis of a High-Performance Real-Time Application with Several AL-FEC Schemes", K. Matsuzono, J. Detchart, M. Cunche, V. Roca et H. Asaeda", 2010, [55]
  → *Cette publication discute des performances de différents codes pour un usage au sein de FECFRAME.*

Silhouette fantomatique du pygargue à travers la frondaison, Canada, 2012.

Part IV

Et pour finir. . .

12
Conclusions, discussions et perspectives

Résumé :
Ce chapitre dresse un bilan des travaux que j'ai menés, en association avec mes collègues et étudiants, autour des codes AL-FEC et des protocoles de protocoles de diffusion robuste, et ouvre sur des perspectives pour des travaux futurs.

12.1 A propos des codes AL-FEC

Nous avons vu dans ce document que concevoir des bons codes AL-FEC, utiles – c'est à dire en phase avec les besoins – et utilisables – c'est à dire d'exploitation libre – est complexe. La difficulté vient essentiellement du caractère multicritères du problème. Avoir une solution qui pêche sur un des critères peut être fatal, pas tant du point de vue scientifique (un article pourra se focaliser sur un aspect et être publié dans la meilleure conférence du domaine), mais lorsque l'on veut aller plus loin et faire que les efforts produits conduisent à des solutions commerciales, intégrées dans des standards, et donc capables de rivaliser face à la concurrence.

12.1.1 Un travail qui est nécessairement global...

↪ Une bonne solution AL-FEC requiert à part égale de bons codes et un bon codec.

Le travail doit impérativement se faire conjointement aux niveaux codes et codec. Ainsi tous les scénarios envisagés lors de la compétition 3GPP-eMBMS (section 6.5) pourraient être adressés avec des codes idéaux (par exemple RS sur GF(2^{16})), fournissant des capacités de correction optimales. Le faire conduirait juste à un système inutilisable si l'on intègre le coût du décodage...

Des exemples moins caricaturaux existent. Ainsi les travaux sur les codes LDPC se sont longtemps limités à un décodage IT, dans la lignée des travaux de 1974 de Zyablov et [108]. Or un décodage ML par pivot de Gauss, longtemps été considéré comme trop complexe du fait

de sa complexité théorique en $O(N^3)$, N étant le nombre de variables du système, est tout à fait réaliste. C'est ce que nous avons montré durant la thèse de M. Cunche [18], et l'usage d'une approche Pivot de Gauss Structuré (section 6.2.2) n'a fait que renforcer cette tendance, s'éloignant plus encore de la complexité théorique en $O(N^3)$.

Sans codec optimisé, aucune évaluation complète n'est possible, et des pistes d'étude pertinentes peuvent être totalement ignorées. De même, faire accepter une proposition par la communauté sera bien plus difficile si l'on ne peut pas démontrer que ces codes peuvent être implémentés efficacement. Or disposer d'un codec optimisé est un travail à part entière, long et difficile.

12.1.2 *... Qui se doit d'intégrer une stratégie de standardisation...*

↪ La standardisation IETF, point de passage obligé.

La standardisation à l'IETF est obligatoire pour deux raisons : (1) c'est une vitrine majeure qui met en avant des technologies clairement identifiées; (2) les standards IETF constituent des solutions abouties, prêtes à l'usage, éventuellement après une légère spécialisation pour s'adapter à un champ d'application spécifique.

Or, travailler sur des codes FEC signifie classiquement se focaliser sur les performances (au sens large) des codes et/ou des codecs. Lorsque l'on travaille sur un standard IETF, de surcroît il faut s'intéresser à l'intégration et à l'usage de ces codes au sein de protocoles et applications divers. Nous avons là une nouvelle dimension qui peut parfois être exploitée pour fournir de nouveaux services. Ainsi notre proposition GOE ajoute des services de protection inégale et de protection globale sur un ensemble d'objets, utilisables avec des codes AL-FEC n'ayant jamais été conçus avec ces objectifs (chapitre 9), en jouant uniquement sur des aspects de signalisation (par exemple pour identifier les symboles devant être associés à chaque classe de priorité). Ceci se fait sans ajouter de complexité prohibitive en terme d'implantation, et sans ajouter non plus de surcharge réseau vis à vis des informations à transmettre au(x) récepteur(s).

12.1.3 *... Et dont l'évaluation est nécessairement multicritères*

↪ Les critères d'évaluation sont multiples, aucun d'eux ne doit être négligé.

Nous avons déjà discuté l'importance des principaux critères de performances techniques : capacités de correction, vitesses d'encodage et de décodage, besoins mémoire crêtes en encodage et décodage. A cela s'ajoutent des critères autres, tout aussi importants, que sont :

- l'adéquation au champ d'application visé : certains codes AL-FEC peuvent être excellents pour certains usages, qui se traduisent par certaines plages de valeurs pour les paramètres clefs, mais pas nécessairement pour les autres usages. Partant de là il est impossible de

dire si ce sont de bons codes ou non dans l'absolu, tout dépendra de la cible;

- la disponibilité d'une implantation de référence, open-source : le projet `http://openfec.org` que j'ai lancé a été un aspect clef dans le succès de nos codes. Or c'est une initiative chrono-phage, qui de surcroît peut nécessiter de développer plusieurs versions de codecs: dans notre cas la version commerciale de certains codes est largement distincte de la version open-source;
- la prise en compte des brevets du domaine : l'appropriation des idées et techniques au profit d'une minorité par le biais de brevets étant une réalité, l'ignorer pourrait compromettre l'exploitation future de nos travaux. En conséquence, nous nous sommes toujours efforcés de rester le plus loin possible des techniques connues comme brevetées, ce qui est très impactant ainsi que discuté dans la section 4.7. Un autre exemple concerne l'usage des codes AL-FEC dans le contexte FECFRAME où nous avons dû mettre en place une solution alternative de façon à ne pas risquer d'enfreindre un brevet;
- la simplicité de la solution : cela reste important, même si ce n'est pas le critère essentiel;

Il en découle que le choix d'une solution AL-FEC résultera d'une mise en balance de différents critères, techniques et non techniques, au moyen d'une alchimie complexe qui dépend totalement du champ d'application prévu. Les performances brutes des codes ne constituent que des critères parmi d'autres.

12.2 *A propos des protocoles de transport*

12.2.1 *ALC/LCT versus NORM*

↪ KISS, "Keep It Simple and Stupid".

Les protocoles de transport constituent la deuxième facette du problème. Après de nombreuses années où les travaux académiques ont proposés de multiples protocoles multicast fiables, le groupe RMT a standardisé deux protocoles, ALC/LCT et NORM (chapitre 10). Ces deux protocoles sont totalement opposés dans leur conception :

- ALC/LCT repose massivement sur l'usage des codes AL-FEC et de techniques de transmission sous forme de carrousel pour atteindre l'objectif de fiabilité. La solution est simple, passe massivement à l'échelle et supporte tout type d'infrastructure sous-jacente puisque ce protocole fonctionne également sur des réseaux unidirectionnels[1].
- à l'inverse, NORM repose sur des mécanismes d'acquittements / demandes de retransmission pour atteindre l'objectif de fiabilité, ce

[1] A noter que les réseaux unidirectionnels sont aussi un élément clef de certains systèmes hautement critiques / confidentiels (militaires par exemple), où le matériel garantit que les échanges ne peuvent se faire que dans un sens. On parle alors de "data diode". FLUTE/ALC est parfaitement adapté à ces situations. `http://en.wikipedia.org/wiki/Unidirectional_network`.

qui apporte une complexité certaine, limite le passage à l'échelle, et requiert la présence d'un chemin de retour.

L'approche ALC/LCT à laquelle j'ai contribué me semble techniquement supérieure pour de nombreux champs d'application pour plusieurs raisons :

- la relative simplicité des spécifications : là où ALC/LCT nécessite 58 pages tout compris [2], il en faut 136 pages pour NORM [3].
- le découpage du problème général en sous problèmes, ou l'application du principe "diviser pour régner" : là où NORM s'inquiète de la réception effective des paquets transmis, ALC/LCT se contente de transmettre. La prise en charge de la réception suite à la fin de la période de transmission peut être effectuée par un "protocole de niveau session" [4], qui laisse aux clients n'étant pas parvenus à finir la réception des objets transmis, l'opportunité de demander des retransmissions, en point-à-point cette fois ci, des symboles manquants. Les deux niveaux, ALC/LCT et protocole de session, sont naturellement complémentaires.

[2] 24 pages pour ALC (RFC 5775) plus 34 pages pour LCT (RFC 5651)

[3] 94 pages pour NORM - Transport Protocol (RFC 5740) et 42 pages pour NACK - Building Blocks (RFC 5401).

[4] Ainsi, dans les déploiements de FLUTE/ALC au sein des réseaux Internet mobiles de type 3GPP, ceci passe par des serveurs de réparation dédiés, cachant le contenu transmis et acceptant des requêtes Hyper-Text Transfer Protocol (HTTP) de retransmission ([4], sections 9.3.5 et 9.3.6). On complète alors un système de diffusion multicast par un système de retransmission point-à-point via le protocole de session universel, HTTP.

12.2.2 *FLUTE versus FCAST*

↪ Une meilleure efficacité de transmission avec le principe architectural ALF, "Application Level Framing".

J'ai participé à la standardisation de l'application FLUTE qui spécialise ALC/LCT pour le cas particulier des sessions de transfert d'objets (fichiers par exemple), initiative lancée par Nokia. Cette spécialisation permet de transporter parallèlement, au sein de la session, les métadonnées associées à chaque objet, afin de permettre à un récepteur d'opérer un filtrage des objets transmis.

Dans un deuxième temps, ayant conscience des limites de FLUTE, j'ai défini et standardisé avec l'aide de B. Adamson l'application FCAST, opérant à la fois sur ALC/LCT et NORM. Ici encore, l'approche FCAST me semble supérieure à FLUTE pour plusieurs raisons :

- un gain en terme d'efficacité de transmission apporté par le respect du principe fondamental Applicatin Level Framing (ALF)[5] : unité de contrôle (ici les métadonnées d'un objet) et unité de transmission (ici le contenu de l'objet) sont identiques. Dès lors, un objet reçu (éventuellement après décodage AL-FEC) sera toujours utilisable avec FCAST. A l'inverse FLUTE impose au préalable d'avoir reçu la FDT qui comporte toutes les métadonnées de l'ensemble des objets de la session, et bien souvent les FEC OTI (informations permettant la synchronisation des encodeurs et décodeurs AL-FEC, voir section 8.3). Dès lors les recommandations de FLUTE portant sur une plus grande priorité à apporter à la transmission des instances de FDT disparaissent avec FCAST.

[5] ALF [16] est un principe architectural relativement universel qui dit que l'efficacité est améliorée si l'unité de données sur laquelle se fait le contrôle est identique à l'unité de transmission.

La seule conséquence est que là où un récepteur a la possibilité de sélectionner à priori les objets qui l'intéressent en consultant la FDT avec FLUTE, il devra décoder tous les objets avec FCAST. En pratique une indirection permet aisément de contourner cette limite : la liste des objets avec leur métadonnées est transmise dans une première session, et les objets en question sont transmis dans une (ou plusieurs) session(s) en parallèle.

- un gain en terme de simplicité, apporté par cette spécialisation (ainsi que la suppression de XML au profit d'un format type HTTP1.1 pour les métadonnées), qui se manifeste (dans de moindres proportions) dans les spécifications[6].

[6] 38 pages pour FCAST qui supporte également NORM, contre 46 pages pour FLUTE (RFC 6726) qui lui se limite à ALC/LCT.

Les travaux récents au sein du 3GPP montrent que la question n'est probablement pas close.

12.2.3 *Et FECFRAME ?*

Nous avons là une architecture simple dans ses principes, très évolutive du fait des nombreuses possibilités de composition de flux source et de redondance (par exemple pour protéger chaque flux individuellement, ou à l'inverse globalement, avec un ou plusieurs flux de redondance), qui pourra être adaptée à de très nombreuses situations. La complexité est plus dans les schémas FEC associés que dans les principes architecturaux, ce qui est là aussi conforme au principe "diviser pour régner".

12.2.4 *Et maintenant ?*

↪ FLUTE "Nouvelle Génération", un protocole universel pour la diffusion fiable de fichiers et pour les flux temps réel.

L'évolution des usages des protocoles de transport pour les flux point à point s'est indéniablement fait vers TCP, pour deux raisons essentielles : d'une part la nécessité de passer des pare-feux et d'autre part la généralisation de HTTP, y compris pour flux temps réels (ce qui n'a rien d'évident) via l'approche Progressive Download et plus récemment DASH).

Les scénarios nécessitant la transmission de volumes importants de données en mode diffusion se sont pour leur part progressivement orientés vers FLUTE/ALC. Pour preuve les scénarios permettant de tester les futurs codes AL-FEC n'incluaient que des flux FLUTE/ALC, y compris pour des flux vidéos transmis de façon progressive (voir la section 6.5). Mais FLUTE/ALC présente des limites, ainsi qu'il a été discuté dans les chapitres précédants et dans cette conclusion. Aussi des travaux ont lieu sur son amélioration possible au sein du 3GPP-SA4, même s'il ne s'agit au moment de la rédaction de ce document que d'une proposition en cours de discussion [34][35][7]. Sans surprise,

[7] Par contre on notera que FECFRAME disparait de facto de la partie utile de la pile protocolaire du 3GPP, même si certaines idées sont reprises dans l'architecture détaillée de [35].

nous retrouvons des idées (et des techniques) qui ont été discutées dans ce document, à savoir :

- la création de "super-objets" associant données source et méta-données (du moins un sous ensemble de celles-ci), dans une optique ALF comparable à celle de FCAST;
- la création d'un niveau d'objets intermédiaires sur lesquels sont effectuées les encodages AL-FEC afin d'apporter protection inégale et protection de groupes d'objets. La solution repose entièrement sur de la signalisation, avec un flux source transmis selon un schéma FEC proche du schéma "No-Code", et un encodage FEC sur un flux distinct. L'optique est donc comparable à celle de notre approche GOE;

Cette évolution récente, qu'elle soit au final retenue ou non, est malgré tout clairement une confirmation de la pertinence de nos contributions.

12.3 *Pour aller plus loin... Variations autour des codes RLC*

Coté codes AL-FEC, nous avons vu que le niveau de performances atteint par les codes LDPC-Staircase est de premier plan : ce fut le cas pour le standard Japonais ISDB-Tmm qui a préféré les codes LDPC-Staircase à RaptorQ en 2011; ce fut également le cas lors de l'appel à technologies pour les futurs codes AL-FEC du service 3GPP-eMBMS où notre proposition a fini première (section 6.5).

Cependant, nous ne prétendons pas avoir la solution ultime. Ainsi que nous l'a aimablement fait remarquer le camp adverse durant la compétition du 3GPP, la solution RS + LDPC-Staircase pêche par :

- la présence de deux codes AL-FEC, ce qui nécessite une sélection du code à utiliser;
- une large plage de paramètres possibles;
- des limites en cas d'usage avec un faible code rate[8];

A toutes ces critiques l'approche GLDPC-Staircase apporte des réponses concrètes. Mais nous pouvons également suivre d'autres pistes. Voici en tous cas quelques éléments de réflexion.

[8] Rappelons qu'après de nombreuses discussions, aucun des scénarios retenus par le 3GPP ne nécessitait d'utiliser un code rate inférieur à 2/3, ce qui est significatif.

12.3.1 *Convolutionnels ou bloc ? N'est-ce pas un peu artificiel ?*

↪ Ne plus être contraint artificiellement par la notion de bloc.

Les codes que nous avons considérés dans ce document sont exclusivement des codes en mode bloc. C'est la solution naturelle lorsque l'on manipule des contenus de taille connue (fichiers), et l'on aura tout intérêt à travailler lorsque l'on manipule de gros contenus de considérer

des codes de type "grand blocs", performants en termes de capacité de correction et de complexité algorithmique lorsque la dimension du code est élevée (10 000 symboles et plus). Inversement, lorsque l'on manipule de petits contenus, ce seront davantage les performances des codes pour de petites dimensions de codes qui seront importantes.

Cependant les codes convolutionnels ont l'avantage de ne pas imposer de notion de bloc, l'encodage ayant lieu sur une fenêtre glissante de symboles source. Les codes convolutionnels offrent donc une flexibilité inégalée, dès lors que les performances en termes de capacités de correction et de complexité algorithmique suivent.

12.3.2 *Quid des codes RLC ?*

↪ Un exemple où des codes triviaux peuvent être de grand secours.

Les codes RLC sont devenus un outil de base dans les applications de type "network coding" où une flexibilité importante est requise afin de pouvoir ré-encoder librement à différents stades d'une transmission. Chaque symbole de redondance d'un code RLC est ici une combinaison linéaire de symboles appartenant à la fenêtre d'encodage (habituellement glissante, comme pour un code convolutionnel), les coefficients non nuls étant choisis sur un corps fini donné :

$$s_j = \sum_{i=0}^{K-1} \alpha_i * s_i$$

où α_i appartient à $GF(2^q)$, $q \geq 1$ donné[9]. A noter que les codes RLC sont, dans le cas où l'on travaille sur $GF(2)$, équivalents à des *codes binaires aléatoires*, et dans le cas où l'on travaille sur $GF(2^q)$, $q > 1$, des *codes non binaires aléatoires*, Ces deux familles de codes sont connues pour avoir de très bonnes (cas $GF(2)$) voir excellentes (cas $GF(2^8)$) capacités de correction.

Ces codes offrent un certain nombre d'avantages :

- ils sont potentiellement efficaces pour de faibles fenêtres d'encodage (on pourra utilement travailler sur $q = 4$ ou 8 pour obtenir de bonnes capacités de correction), ou inversement de grandes fenêtres d'encodage (on pourra alors utilement travailler sur $q = 1$ pour limiter la complexité algorithmique). Leur souplesse est de ce coté là totale;

- ils sont naturellement de type rateless, la production d'un nouveau symbole de redondance étant immédiate dès lors qu'un nouvel ensemble de coefficients est choisi;

- ils ont une simplicité de construction inégalée.

[9] En général on choisira des coefficients α_i sur $GF(2)$, sur $GF(2^4)$, sur $GF(2^8)$. Le cas $GF(2)$ est intéressant dans la mesure où il permet d'effectuer des opérations de multiplication de deux éléments du corps fini au moyen de XOR. Le cas $GF(2^4)$ (resp. $GF(2^8)$) pour sa part nécessite de redescendre au niveau de l'élément de taille 4 bits (resp. 8 bits). Le traitement de paquets est donc plus coûteux puisqu'il s'agira de manipuler indépendamment chaque demi-octet (ou octet) du symbole. Le choix de valeurs autres pour le corps fini est évité, essentiellement pour des raisons de taille de l'élément qui n'est plus aligné sur l'octet (par exemple si q est impaire), ou bien de taille des tables pré-calculées de multiplication de deux éléments du corps fini qui excède rapidement la taille du cache mémoire (par exemple avec $GF(2^{16})$).

Transmission du vecteur d'encodage : Mais tout n'est pas rose! Une première difficulté se trouve coté *signalisation* : comment indiquer à un récepteur comment a été construit un symbole de redondance de façon compacte ? Ceci passe par la transmission du vecteur d'encodage, c'est à dire de l'ensemble des coefficients $\{\alpha_i\}_{0 \leq i < K}$. Dans le cas d'un code RLC binaire, ceci peut se faire au moyen d'un champ de bits. Dans le cas d'un code RLC non binaire, les coefficients peuvent être transmis en séquence (un octet par coefficient dans le cas $q = 8$, deux coefficients par octet dans le cas $q = 4$). Cependant au delà de quelques dizaines de symboles dans la fenêtre d'encodage, transmettre ce champ tel conduit rapidement à un surcoût de transmission important. D'autres techniques existent, mais hormis l'approche PRNG plus graine, les surcoûts de transmission restent élevés[10].

Densité élevée du vecteur d'encodage et du système linéaire associé : Une autre difficulté réside dans la *densité de ces codes*. Par défaut, un code binaire aléatoire aura une densité 1/2. Pour une fenêtre de k symboles, ceci conduit à effectuer $k/2$ opérations XOR pour chaque symbole de redondance. Le cas de codes non binaires aléatoires est similaire. Ceci va bien au delà de la densité des codes LDPC-Staircase par exemple, bien plus creux, ce qui limite le nombre d'opérations XOR requis et rend possible l'usage d'un décodeur IT : ainsi, pour $N_1 = 7$, $CR = 2/3$, un symbole de redondance fait intervenir $max(2; \frac{CR*N_1}{1-CR}) + 1 = 15$ symboles, soit un nombre constant indépendant de k.

[10] Il est bien sûr possible d'effectuer une compression sans perte de ce vecteur dans le cas binaire, avec par exemple une approche de type Run-Length Encoding (RLE) (technique de compression consistant à indiquer le nombre de positions successives avec une certaine valeur, 0 ou 1, http://fr.wikipedia.org/wiki/Run-length_encoding) (c'est sans intérêt dans le cas non binaire les coefficients n'ayant aucune raison d'être égaux). Par contre le vecteur devient de taille variable suivant la répartition des coefficients. Il est aussi possible d'utiliser une combinaison de ces deux techniques, en utilisant la plus appropriée à chaque fois. Mais là encore il n'est pas possible de garantir une taille fixe à ce vecteur, obligeant par là même à restreindre la taille de chaque symbole en considérant le pire cas. [106] propose différentes techniques pour transmettre ce vecteur. Il serait également possible, dans le cas où la séquence de paquet est définie par un PRNG, de ne transmettre que la graine utilisée (en supposant le PRNG connu de part et d'autre), accompagné de l'indice du premier symbole et de la taille de la fenêtre d'encodage. La graine pourrait soit être transmise dans le paquet en tant que tel, soit être constitué de l'identifiant de symbole (ESI).

12.3.3 *Une idée à creuser : les codes RLC structurés*

↪ Des codes RLC "structurés", ou comment revisiter le problème des codes AL-FEC.

La structure totalement uniforme des codes RLC n'est pas une fatalité, et nous proposons d'ajouter une structure à ces codes (le "R" de l'acronyme RLC perd alors un peu de son sens).

Une densité variable : Ainsi une solution peut consister à agir sur la *densité de ces codes*[11]. Disposer de codes avec une densité élevée sera intéressant pour de petites fenêtres d'encodage, et inversement, à mesure que la taille de cette fenêtre augmente, on pourra diminuer cette densité pour contenir la complexité algorithmique.

[11] Par densité nous faisons référence au nombre de coefficients non nuls, soit égaux à 1 avec les codes sur $GF(2)$, soit choisis sur $GF(2^4) - \{0\}$ (resp. $GF(2^8) - \{0\}$) pour les codes sur $GF(2^4)$ (resp. $GF(2^8)$).

Des colonnes de coefficients non binaires : Une autre variante, afin de trouver un compromis entre capacités de correction élevées et complexité algorithmique faible, peut consister à opter pour des *coefficients choisis sur $GF(2)$ pour la majorité des symboles sources, et des coefficients sur $GF(2^4)$ ou $GF(2^8)$ pour les symboles restants*, ces derniers étant en faible nombre. On pourra avoir intérêt à restreindre les coefficients non binaires au même sous ensemble de symboles sources (donc en

colonnes) dans une optique de décodage de type SGE (section 6.2.2). La densité des coefficients non nuls pourra dès lors être réduite tout en conservant d'excellentes performances. La figure 12.1 montre la matrice de parité d'un tel code, certaines colonnes utilisant des coefficients non binaires alors que le reste est largement creux et binaire.

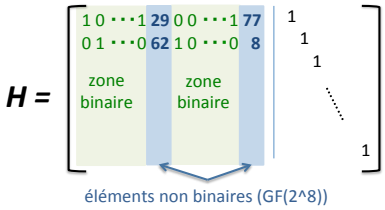


Figure 12.1: Exemple de construction de codes RLC structurés, présentant un nombre restreint de colonnes non-binaires, le reste étant constitué de coefficients binaires.

Des colonnes de coefficients binaires de densité $1/2$ *:* Il est possible d'ajouter des colonnes de coefficients binaires de densité $1/2$. La solution complète comportera alors un structure binaire creuse pour l'essentiel, des colonnes en petit nombres avec des coefficients binaires et une densité $1/2$, et enfin des colonnes en nombres encore plus faibles avec coefficients non binaires. Ceci pourra être utile afin de limiter la complexité algorithmique liée au recours aux coefficients sur $GF(2^8)$.

Ajouter une somme XOR de tous les symboles : Des structures supplémentaires peuvent être appliquées. Par exemple il est possible que le premier symbole de redondance soit la *somme de tous les symboles sources* depuis le début de la session ou dans la fenêtre d'encodage courante. Il est possible que les symboles de redondances soit la somme de ce premier symbole et d'un sous ensemble des autres symboles sources, en respectant par exemple la structure de la figure 12.1.

Prise en compte des contraintes temps réelles : Il peut également être intéressant, lorsque la pertinence d'un symbole source diminue avec le temps (par exemple dans le cas de données "temps réelles"), *d'asservir la probabilité d'inclure ce symbole dans le calcul d'un symbole de redondance avec sa durée de présence dans la fenêtre d'encodage*. Ainsi les symboles les plus anciens seront naturellement et progressivement moins bien protégés que les symboles récents à mesure que le temps passe. Ce peut être une propriété utile lorsque la fenêtre glissante évolue sans considération sur la réception ou la reconstruction des symboles sources du coté du ou des récepteurs. Le retrait d'un symbole source ancien de la fenêtre d'encodage ne conduira pas un récepteur qui n'aurait pas reçu ou décodé ce symbole à mettre de coté toutes les équations où il intervient, ces équations devenant difficiles à décoder, ou à l'inverse, si elles sont finalement décodées, conduisant à reconstruire un symbole ancien qui ne pourra au final être utilisé.

D'autres pistes existent qu'il faudra évaluer. Il faudra conserver à l'esprit la question des mécanismes de signalisation nécessaires pour pouvoir synchroniser émetteur et récepteurs sur la façon dont a été construit chaque symbole de redondance. Cette signalisation peut largement limiter les constructions pratiques de codes RLC structurés.

12.3.4 *Quid des protocoles de transport ?*

↪ Une adaptation nécessaire des protocoles de transport.

Les protocoles ALC/LCT et FECFRAME ont tous été conçus pour fonctionner avec des codes en mode bloc. Les codes RLC pouvant être utilisés soit en mode bloc, soit en mode convolutionnel, il est intéressant d'étudier quels bénéfices éventuels ils peuvent apporter à ces protocoles.

Ainsi utiliser des codes convolutionnels dans le cas du protocole FECFRAME prévu pour fonctionner avec des flux continus semble naturel. En effet, l'usage de codes AL-FEC de type bloc dans FECFRAME n'a d'intérêt que de limiter, par construction, la latence maximale de récupération des pertes. Quels seront les impacts de la largeur k de la fenêtre d'encodage d'un code RLC sur cette latence dans un usage FECFRAME ?

A l'inverse, ALC/LCT est intrinsèquement lié à la notion d'objets, aux frontières bien définies, ce qui rend l'usage de codes AL-FEC de type bloc est naturel[12]. Cependant on peut objecter que :

[12] C'est d'autant plus vrai que l'objet a une taille compatible avec un encodage en une passe, au sein d'un unique bloc.

- le redécoupage en blocs induit directement le problème du collecteur de coupons (section 3.2.1). Ce soucis disparaît avec un code convolutif, qui n'induit pas de notion de bloc. Il est possible toutefois que la largeur de la fenêtre d'encodage doive utilement être adaptée face à la longueur moyenne d'une rafale de perte.

- un deuxième intérêt d'avoir une approche avec fenêtre d'encodage glissante, de largeur B, sera de limiter la complexité algorithmique[96]. Ainsi dans [92] nous introduisons une approche permettant de limiter à $O(kB)$, la complexité algorithmique théorique lors du décodage ML), du fait de la structure bande de la matrice génératrice. Des propriétés proches pourraient être obtenues ici.

Tous ces points sont pour l'instant ouverts, mais ils sont indéniablement porteurs. A suivre...

Coucher de soleil sur les alpages, massif de Belledonne, Isère, octobre 2011.

Annexe : Gros plan sur les performances des codes LDPC-Staircase

Résumé :
Ce chapitre donne des détails sur les performances des codes LDPC-Staircase et complète le chapitre 5. Tous les résultats ont été obtenus au moyen des outils d'évaluations du projet OpenFEC.org.

12.4 *Probabilité d'échec de décodage des codes LDPC-Staircase dans le cas d'un décodage ML*

Nous traçons les probabilités d'échec de décodage en fonction de l'overhead. pour différentes valeurs de N_1, de taille de l'objet (ou de bloc d'encodage, k, puisque chaque objet n'est constitué que d'un seul bloc dans ces tests), et du code rate, CR.

Il est à noter que chaque test correspond à la fois à un code différent et à un séquencement de paquets différent. A l'inverse nous aurions pu figer un code en optant pour un ensemble $\{n, k, N1, graine\ du\ PRNG\}$ donné, et évaluer les performance de ce code. Ce que nous montrons ici doit plutôt être vu comme des performances moyennes sur l'ensemble des codes possibles pour un triplet $\{n, k, N1\}$ donné.

Figer le code durant les tests serait intéressant si l'on disposait d'un oracle indiquant quelle valeur de graine utiliser afin de réduire le pied de courbe, les autres paramètres étant donnés.

Le test consiste alors à déterminer le nombre de symboles minimum permettant d'effectuer le décodage dans ces conditions, c'est à dire le seuil $k * \epsilon_{ML}$. Chaque courbe montre à la fois la courbe $proba\_chec = f(overhead)$, ainsi que l'histogramme associé qui représente la distribution des ϵ_{ML}.

Il est à noter que l'allure générale de la courbe change totalement en fonction de l'échelle horizontale ! Dans toutes les courbes qui suivent, l'abscisse montre exactement 50 symboles. Il est à noter également qu'avoir besoin de 10 symboles supplémentaires dans le cas d'un bloc de taille 256 symboles ne correspond pas au même overhead ϵ_{ML} que dans le cas d'un bloc de taille 1024 symboles[13].

[13] Plus précisément, 10 symboles en plus pour $k = 256$ correspond à $\epsilon_{ML} = 3.91\%$, contre $\epsilon_{ML} = 0.98\%$ si $k = 1024$. C'est à garder à l'esprit.

12.4.1 *Code rate 0.9*

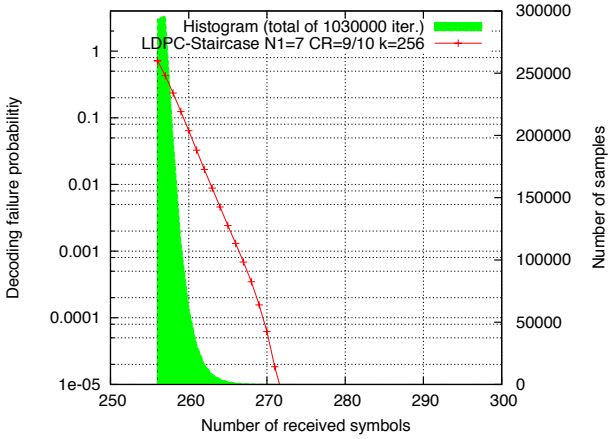
La figure 12.2 montre les différentes courbes dans le cas où le code rate est fixé à 0.9. Les simulations comportant 10^6 itérations, nous limitons les courbes à une précision de 10^{-5}.

12.4.2 *Code rate 2/3*

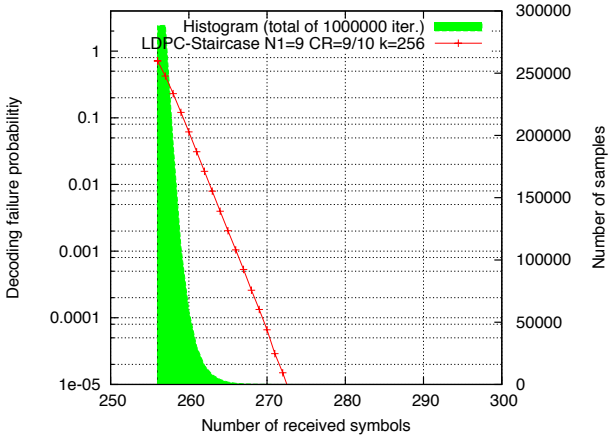
La figure 12.3 montre les différentes courbes dans le cas où le code rate est fixé à $2/3$. Les simulations comportant 10^6 itérations, nous limitons les courbes à une précision de 10^{-5}.

12.4.3 *Code rate 1/2*

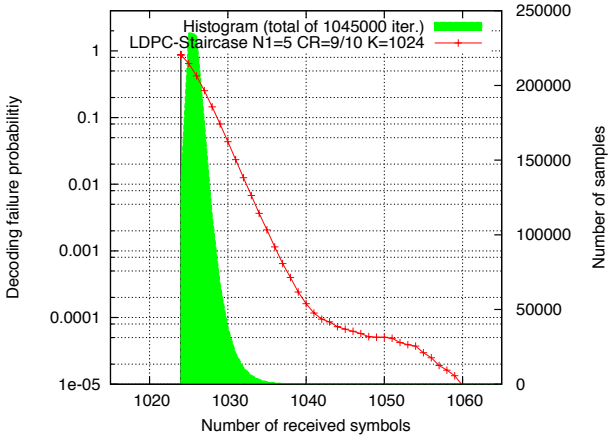
La figure 12.4 montre les différentes courbes dans le cas où le code rate est fixé à $1/2$. Les simulations comportant 10^6 itérations, nous limitons les courbes à une précision de 10^{-5}.



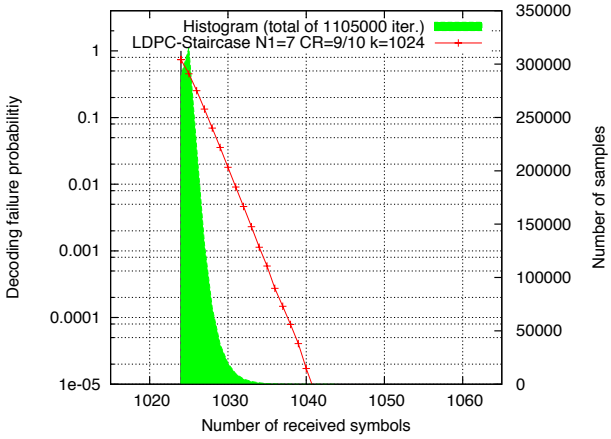
(a) $k = 256$ symboles, $N_1 = 7$



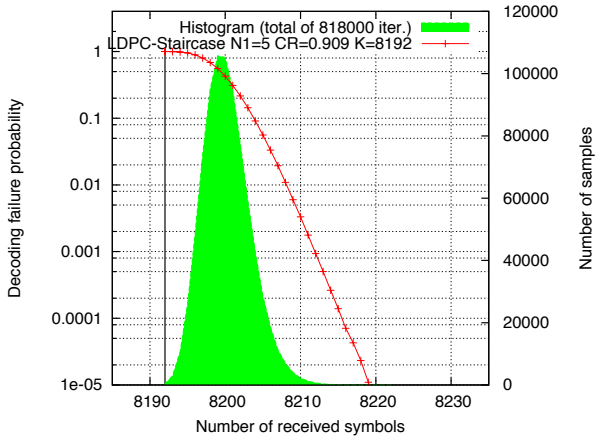
(b) $k = 256$ symboles, $N_1 = 9$



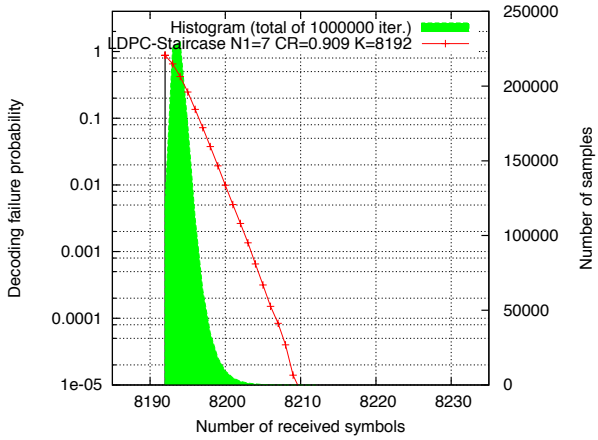
(c) $k = 1024$ symboles, $N_1 = 5$



(d) $k = 1024$ symboles, $N_1 = 7$

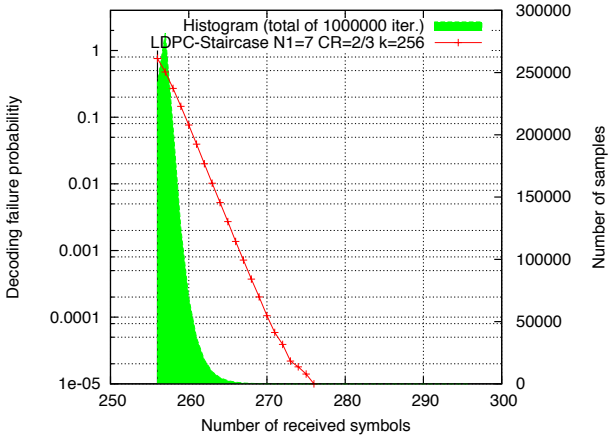


(e) $k = 8192$ symboles, $N_1 = 5$, $CR = 0.909$

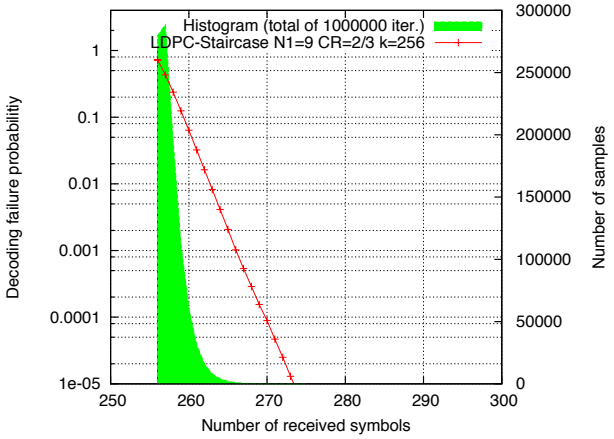


(f) $k = 8192$ symboles, $N_1 = 7$, $CR = 0.909$

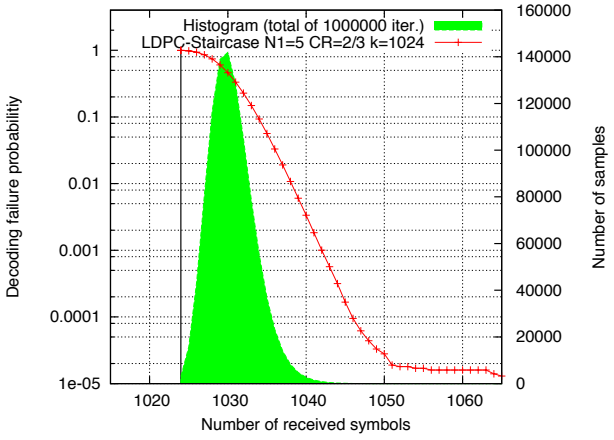
Figure 12.2: Probabilité d'échec de décodage des codes LDPC-Staircase pour différentes valeurs de k et N_1, quand $CR = 0.9$ ($CR = 0.909$ pour les cas où $k = 8192$).



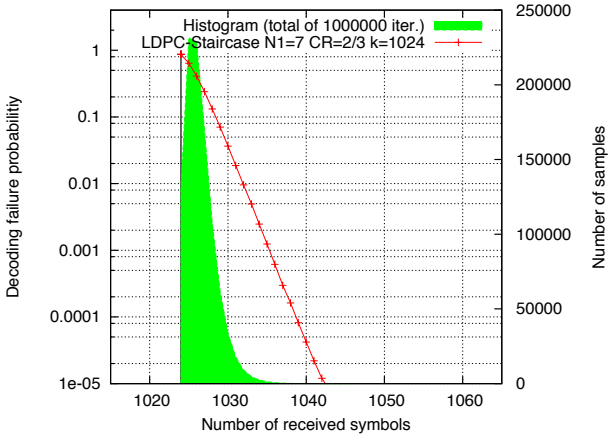
(a) $k = 256$ symboles, $N_1 = 7$



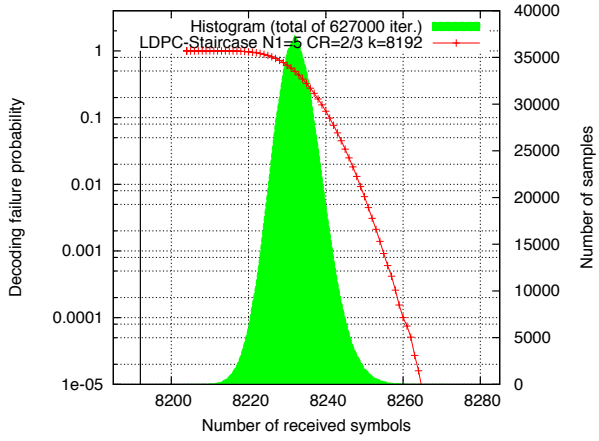
(b) $k = 256$ symboles, $N_1 = 9$



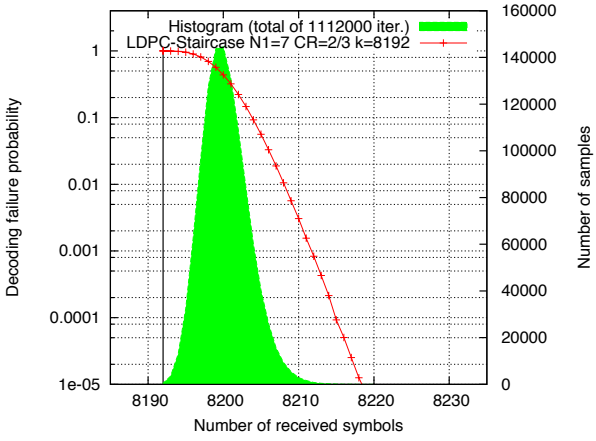
(c) $k = 1024$ symboles, $N_1 = 5$



(d) $k = 1024$ symboles, $N_1 = 7$

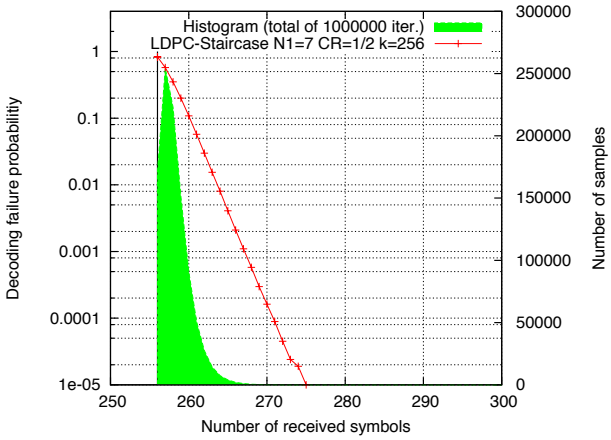


(e) $k = 8192$ symboles, $N_1 = 5$ (attention, l'échelle horizontale est ici différente)

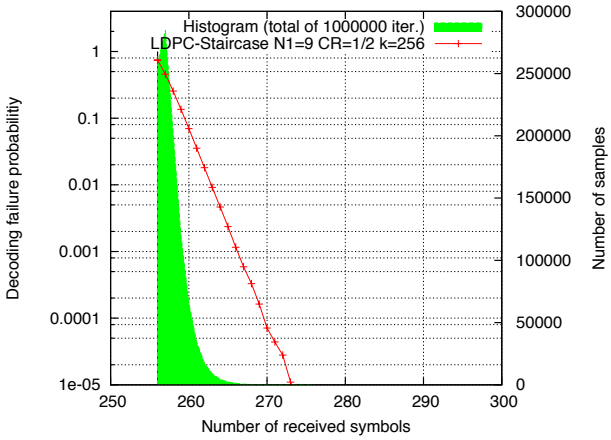


(f) $k = 8192$ symboles, $N_1 = 7$

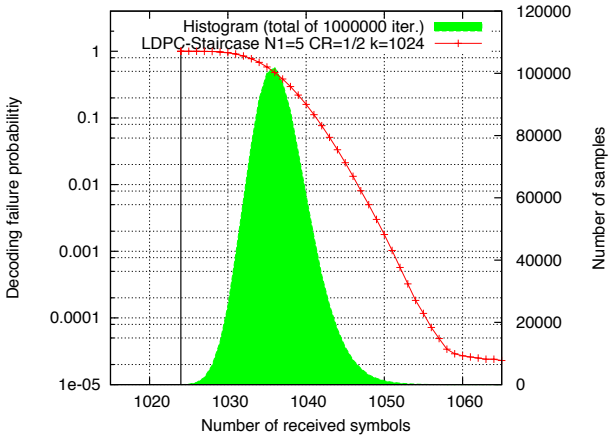
Figure 12.3: Probabilité d'échec de décodage des codes LDPC-Staircase pour différentes valeurs de k et N_1, quand $CR = 2/3$.



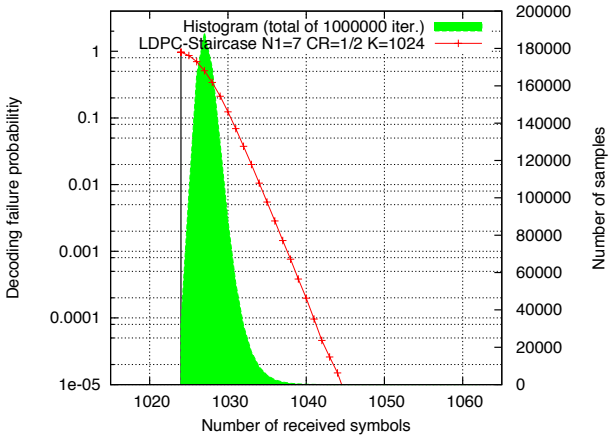
(a) $k = 256$ symboles, $N_1 = 7$



(b) $k = 256$ symboles, $N_1 = 9$

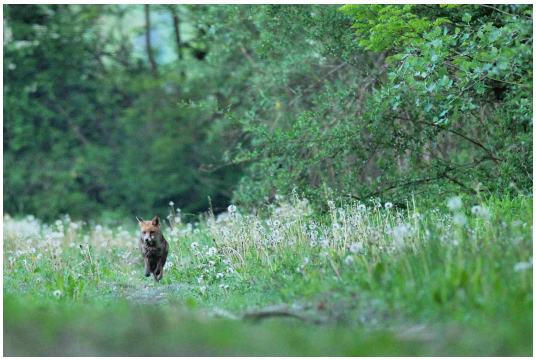


(c) $k = 1024$ symboles, $N_1 = 5$



(d) $k = 1024$ symboles, $N_1 = 7$

Figure 12.4: Probabilité d'échec de décodage des codes LDPC-Staircase pour différentes valeurs de k et N_1, quand $CR = 1/2$.



Footing matinal dans les pissenlits. Isère, 2010.

Annexe : Outils d'évaluation de performances, ou comment savoir si des codes AL-FEC sont bons ?

Résumé :
Ce chapitre présente différentes outils d'évaluation de performance de codes AL-FEC que nous avons développés.

Crédits : *Les outils présentés dans ce chapitre sont issus de développements et de discussions que j'ai menés avec C. Neumann, M. Cunche, J. Detchart, et V. Savin.*

12.5 De la nécessité d'un environnement d'évaluation riche

Evaluer les performances des codes AL-FEC, selon les multiples métriques possibles détaillées au sein du chapitre 3, ne peut se faire sans disposer d'une riche boite à outils, adaptable aux différents besoins. De plus cette boite à outils doit pouvoir être utilisée avec les différents codes AL-FEC dont on veut pouvoir comparer les performances, puisque c'est l'unique moyen d'être sûr de disposer des mêmes conditions expérimentales. L'environnement de simulation devra donc permettre de contrôler les paramètres suivant :

- la stratégie de décodage;
- le flux de paquets reçus;

12.5.1 Stratégie de décodage

Les codes AL-FEC ont des comportements différents en terme de stratégie de décodage. Ainsi, avec un code MDS, il suffira de compter le nombre de symboles disponibles et lancer le décodage du bloc au moment où il atteint k. A l'inverse, avec un code LDPC-Staircase (ou autre) qui présente un overhead non nul, une autre stratégie doit être mise en place. Avec des codes LDPC, nous effectuons un décodage IT au fil de l'eau, pour chaque symbole disponible. Ce décodage IT peut éventuellement aboutir après réception d'un nombre suffisant de symboles [14]. Dans le cas contraire, si le récepteur est en mesure de dire à un moment donné qu'aucun nouveau symbole ne sera disponible (chose aisée dans le cas d'une simulation) et si au moins k symboles

[14] Le décodage IT réussira en moyenne après réception de $k(1+\epsilon_{ITmoy})$ symboles, où ϵ_{ITmoy} est l'overhead moyen dans le cas d'un décodage IT.

sont disponibles, alors ce récepteur effectue un décodage ML qui peut ou non réussir.

Cette approche exploite au mieux la totalité des symboles disponibles au récepteur, en tirant parti du décodage IT, moins complexe, et garantit des capacités de décodage optimales, pour un flux de paquets reçus donné, par l'usage du décodeur ML au besoin. Par contre évaluer l'overhead permis par un décodeur ML, ϵ_{ML}, pour un flux de paquets donné nécessite de trouver le nombre minimum de paquets permettant un décodage, ce qui relève d'une autre démarche. Pour trouver ϵ_{ML} pour un flux donné, il faut, en partant d'une situation où le décodage réussit, figer l'ordonnancement des paquets et le canal, et augmenter progressivement le nombre de pertes jusqu'à trouver le seuil en deçà duquel tout décodage échoue.

[15]

Une autre approche est possible, plus agressive, qui tente un décodage ML plus tôt, dès lors qu'un seuil prédéfini du "nombre de symboles disponibles" est franchi. Cette stratégie peut être utile dans le cas d'un déploiement dans un scénario donné, de ces codes, par exemple lorsqu'il n'y a pas de mécanisme permettant de déterminer de façon fiable si de nouveaux symboles sont susceptibles d'arriver ou non. Cette stratégie requiert une phase préalable d'analyse de performances de ces codes afin de définir des seuils adéquats. Si ce n'est pas l'approche suivie pour les outils d'évaluation de performances, c'est en revanche l'approche suivie pour leur déploiement au sein de notre pile FLUTE/ALC.

[15] NB: il est possible de procéder dans l'autre sens : tester si le décodage réussit après réception des k premiers paquets du flux, si ce n'est pas le cas, tester avec les $k+1$ premiers paquets, etc. jusqu'à observer un décodage réussi. Des optimisations sont possibles, telle une recherche dichotomique, afin de trouver plus rapidement ϵ_{ML}.

12.5.2 *Définition du flux de paquets reçus : ordonnancement de paquets, modèle de canal, et modèle de réception*

Les performances que l'on obtient dépendent clairement du flux de paquets que le récepteur aura à disposition, qui à sont tour dépend de ce que l'émetteur envoie et du comportement du canal. Une autre façon de procéder, qui permet un contrôle plus fin, consiste à définir directement le modèle de réception, et donc de passer outre ordonnancement de paquets et modèle de canal (figure 12.5).

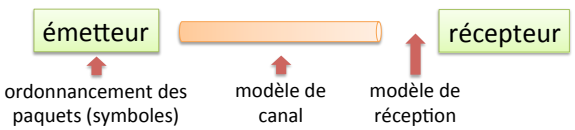


Figure 12.5: Les leviers pour définir le flux de paquets reçus : ordonnancement de paquets, modèle de canal, et modèle de réception.

Modèle de canal :
Les performances que l'on obtient dépendent tout d'abord du modèle de canal : canal à pertes sans mémoire (ou IID) ou canal avec mémoire et paramètres de pertes.

Le modèle de Gillbert-Elliot simple [31][25] permettra de modéliser nombre de ces canaux, via les deux paramètres p et q (figure 12.6). Plusieurs variantes existent pour ce modèle : nous considérons le modèle appelé "Simple Gilbert" qui modélise uniquement des pertes consécutives; les variantes permettront de modéliser des pertes corrélées ou encore des pertes corrélées et isolées [88].

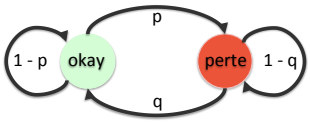


Figure 12.6: Modèle de pertes de Gillbert-Elliot.

On vérifiera que :

- pas de perte : ce cas correspond à $p = 0$.
- pertes IID (ou modèle de Bernoulli) : ce canal correspond au cas où $q = 1 - p$.

De même, nous pouvons déterminer le taux de pertes moyen (figure 12.7) : $l = \frac{p}{p+q}$.

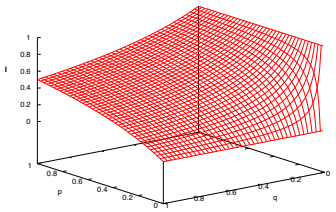


Figure 12.7: Taux de pertes moyens l en fonction des paramètres (p, q) du modèle Gillbert-Elliot.

Dès lors, si n paquets sont transmis (nous supposons que l'encodage se fait en un seul bloc), seulement $n_{rx} = n(1 - l)$ paquets sont reçus en moyenne. Un décodage sera impossible si moins de k paquets sont reçus, ce qui définit des zones de décodage théoriquement impossibles. La valeur frontière est atteinte lorsque $n_{rx} = n * CR$, soit lorsque[16] :

$$q_{seuil} = \frac{p}{\frac{1}{CR} - 1}$$

[16] Il faut noter que l'on manipule des probabilités. Dès lors une instance de canal de paramètre (p,q), pour un PRNG et une graine donnée, peut conduire à un nombre de paquets reçus différent de n_{rx}. Les frontières obtenues entre décodage possible et impossible lors de simulations ne seront pas nécessairement délimitées par la droite q_{seuil}, y compris pour un code idéal.

Ordonnancement de paquets à l'émission :
Avec les codes MDS les performances ne dépendront pas de la stratégie d'ordonnancement des paquets à l'émission. Dès lors que k symboles parmi n sont disponibles, par définition le décodage réussit.

Il existe des codes où cette dépendance des performances vis à vis de l'ordonnancement des paquets à l'émission est marginale. Il s'agit par exemple des codes produits par des combinaisons linéaires aléatoires à partir soit des symboles sources (cas des codes RLC) soit de symboles intermédiaires (cas des codes Raptor/RaptorQ). Il s'agit indéniablement d'une propriété intéressante.

En revanche, avec les codes LDPC, cette dépendance n'est pas nulle et pouvoir tester les performances pour plusieurs stratégies d'ordonnancement à l'émission est important. Nous revenons sur cet aspect plus loin (section 12.7).

Modèle de réception :
Une autre façon de procéder, qui permet un meilleur contrôle de ce

que reçoit un récepteur, consiste à définir un modèle de réception, qui supplante à la fois modèle de canal et stratégie d'ordonnancement.

12.6 Présentation des outils du projet OpenFEC

Dans le cadre du projet OpenFEC, nous avons repris et largement amélioré un certain nombre d'outils d'évaluation de performance. Ils sont au nombre de trois :

- l'application `eperftool`;
- le script de contrôle des simulations;
- les outils d'analyse de résultats;

12.6.1 Application eperftool

Pour un objet de taille donnée, cet outil effectue si besoin le découpage en blocs (nécessaire si la taille de l'objet excède les contraintes du code AL-FEC), encodage chaque bloc indépendamment, définit l'ordre de transmission désiré (par exemple une permutation globale de tous les paquets, contenant soit un symbole source, soit un symbole de redondance), simule les pertes sur le canal de transmission en utilisant un modèle prédéfini, et effectue le décodage de tous les blocs de l'objet, avant de vérifier l'intégrité de celui ci. Un modèle de réception peut également être utilisé en lieu et place du modèle de canal.

Cette application dispose de la flexibilité nécessaire pour pouvoir utiliser différents codes AL-FEC : Reed-Solomon sur $GF(2^4)$ et $GF(2^8)$, codes de parité 2D, LDPC-Staircase, codes linéaires binaires dont la matrice de parité est spécifiée au sein d'un fichier[17], et d'autres codes en cours de développement. Ceci est facilité par l'API unifiée OpenFEC.

[17] Spécifier une matrice de parité au sein d'un fichier permet d'utiliser des outils d'optimisation de code LDPC (par exemple) externes, de générer un code et d'en tester les performances avec l'environnement OpenFEC.

12.6.2 Le script de contrôle des simulations

Le script de contrôle des simulations est un script `PERL` qui permet, à partir d'un fichier de configuration, de lancer l'application `eperftool` autant de fois que nécessaire, avec le bon paramétrage. Il en résulte, pour chaque test, des résultats bruts qui sont alors enregistrés dans une bases de données `MySQL` (ou `SQLite`, ou au choix). Ces tests se font en parallèle sur tous les CPUs et tous les coeurs de la machine, si nécessaire, de façon automatique. Il est aussi possible, après un découpage manuel de l'espace de valeurs des paramètres d'intérêt, de lancer les tests sur plusieurs machines, afin de disposer d'un surcroît de puissance de calcul[18]. Dans les deux cas, la jonction des résultats

[18] Malgré ces traitements parallèles, des simulations en vue d'établir les probabilité d'échec de décodage en fonction du taux de perte/overhead avec une précision de 10^{-5} vont nécessiter plusieurs jours de calculs sur une machine double processeurs récente, disposant d'un total de 8 coeurs physiques.

dans la base est automatiquement réalisée lors de l'alimentation, via la connexion réseau, de la base `MySQL`.

12.6.3 *Les outils d'analyse de résultats*

Les outils d'analyse des résultats sont des scripts `PERL` qui extraient les données brutes de la base `MySQL`, effectuent les traitements statistiques désirés, et mettent en forme les résultats afin de produire les courbes et statistiques désirées. Le tracé final des courbes se fait au moyen de l'outil `gnuplot`.

12.7 *Etudes de performances avec des canaux Bernoulli et Gilbert-Elliot*

L'évaluation de performances de codes AL-FEC nécessite de choisir un type de canal représentatif des champs d'applications considérés. En particulier doit on considérer un canal sans mémoire pour lequel les pertes seront isolées, ou bien un canal à mémoire avec le modèle Gilbert-Elliot simple, qui permettra de considérer des rafales de pertes de paquets ?

Face à la difficulté d'assigner des valeurs adéquates aux paramètres $\{p, q\}$ dans le modèle de Gilbert-Elliot simple, on aura souvent recours au modèle de Bernoulli, couplé à un ordonnancement de paquets totalement aléatoire. Avec un ordonnancement aléatoire, les résultats seront équivalents que l'on considère le modèle Gilbert-Elliot simple et le modèle de Bernoulli, puisque dans les deux cas l'entrelacement lié à cet ordonnancement cassera l'effet de toute rafale de pertes éventuel. Un seul paramètre sera à considérer, le taux de pertes global.

Cependant si l'on veut étudier l'impact de l'ordonnancement sur les performances de codes AL-FEC, le modèle Guilbert-Elliot simple devra être considéré si l'on veut connaître l'impact éventuel de rafales de pertes. Pour mener à bien l'étude, nous proposons l'approche suivante [58] :

L'idée est de faire varier les paramètres p et q en considérant (par exemple) 21 valeurs entre 0 et 1, en incrémentant de 0.05. *Cela résulte en une grille de taille* 21×21, *correspondant à 441 canaux différents. Pour chaque canal (donc couple* $\{p, q\}$, *nous faisons (par exemple)* 1000 *tests. Dans certains tests, le décodage AL-FEC réussit et le récepteur peut reconstruire l'objet source, dans d'autres tests il y a échec. Les performances du code AL-FEC ou de l'ordonnancement utilisé avec ce code sera quantifié par le nombre total de tests réussis pour l'ensemble des canaux considérés. Cette approche permettra donc de comparer différents codes ou ordonnancements les uns avec les autres, en comparant directement et uniquement le nombre total de tests réussis. En fonction des besoins, une granularité plus fine pourra être considérée, ou bien un sous ensemble de l'espace* $\{p, q\}$ *sera considéré si celui ci correspond aux canaux que l'on sait intéressants pour l'usage envisagé.*

Il semble de prime abord que cette approche ignore la notion d'overhead. En fait un code/ordonnancement ayant un overhead de décodage faible permettra de décoder dans des conditions plus difficiles, donc pour un plus grand nombre de canaux qu'un code/ordonnancement moins performant. C'est par ce biais que les performances se manifesteront.

12.8 *Pour plus d'informations...*

Nos publications relatives au thème :

- "Impacts of the Packet Scheduling on the Performance of Certain Erasure Codes: Methodology and Application to GLDPC-Staircase", F. Mattoussi, V. Roca et B. Sayadi, [58]
  → *Cet article introduit une méthodologie pour une évaluation systématique de performances pour différents types d'ordonnancements de paquets et de canaux à mémoire, puis applique cette approche aux codes GLDPC-Staircase.*

Grands corbeaux dans la brume, Haute Savoie, août 2011.

List of acronyms

3G 3rd Generation

3GPP 3rd Generation Partnership Project

ADU Application Data Unit

ADUI Application Data Unit Information

AL-FEC Application-Level Forward Erasure Correction

ALC Asynchronous Layered Coding

ALF Applicatin Level Framing

API Application Programming Interface

ARQ Automatic Repeat reQuest

ASM Any Source Multicast

AWGN Additive White Gaussian Noise

BCH Bose, Ray-Chaudhuri, Hocquenghem code

BEC Binary Erasure Channel

BER Block Error Rate

BP Belief Propagation

BSC Binary Symmetric Channel

CPU Central Processing Unit

CR Code Rate

CRC Cyclic Redundancy Check

DASH Dynamic Adaptive Streaming over HTTP

DA-UEP Dependency-Aware Unequal Erasure Protection

DCCP Datagram Congestion Control Protocol

DE Density Evolution

DoS Denial of Service

DVB-H Digital Video Broadcasting - Handheld

DVB-SH Digital Video Broadcasting - Satellite Handheld

ECC Error-Correcting Code

eMBMS extended Multimedia Broadcast and Multicast Services

ESI Encoding Symbol Identifier

EXIT EXtrinsic Information Transfer

FAI Fournisseur d'Accès Internet

FCAST Scalable Object Delivery over ALC and NORM Protocols

FCS Frame Check Sequence

FEC Forward Erasure Correction

FEC Encoding ID FEC Encoding IDentifier

FEC Instance ID FEC Instance IDentifier

FEC OTI FEC Object Transmission Information

FECFRAME Forward Erasure Correction Framework

FFCI FEC Framework Configuration Information

FDT File Delivery Table

FDT Instance File Delivery Table Instance

FFT Fast Fourier Transform

FLID-SL Fair Layered Increase/Decrease with Static Layering

FLID-DL Fair Layered Increase/Decrease with Dynamic Layering

FLUTE File Delivery over Unidirectional Transport

FPI FEC Payload IDentifier

FSSI FEC-Scheme-Specific Information

GE Gaussian Elimination

GLDPC Generalized LDPC

GLDPC-Staircase Generalized LDPC-Staircase

GNU Gnu is Not Unix

GOE General Object Encoding

GOP Group Of Pictures

GPL General Public License

GZIP GNU Zip

HARQ Hybrid Automatic Repeat reQuest

HTML Hyper-Text Markup Language

HTTP Hyper-Text Transfer Protocol

IANA Internet Assigned Numbers Authority

IETF Internet Engineering Task Force

IGMP Internet Group Management Protocol

IID Independent and Identically Distributed

ILP Integrated Layer Processing

IP Internet Protocol

IPTV IP Television

IRA Irregular Repeat Accumulate

IT ITerative decoding

ISDB-Tmm Integrated Services Digital Broadcasting - Terrestrial mobile multi-media

LAN Local Area Network

LCT Layered Coding Transport

LDGM Low Density Generator Matrix

LDPC Low Density Parity Check

LDPC-Staircase Low Density Parity Check-Staircase

LT Luby Transform

LWC Low Weight Codeword

MB Macro Block

MPE-FEC Multiprotocol Encapsulation - FEC

MPE-FEC Multiprotocol Encapsulation - FEC

MPE-IFEC Multiprotocol Encapsulation – Inter-burst FEC

MBMS Multimedia Broadcast and Multicast Services

MAC Medium Access Control

MD5 Message Digest 5

MDS Maximum Distance Separable

ML Maximum Likelihood

MLD Multicast Listener Discovery

MTU Maximum Transmission Unit

NFS Network File System

NORM Negative acknowledgment Oriented Reliable Multicast Protocol

NTP Network Time Protocol

OSI Open Systems Interconnection

P2P Peer-to-Peer

PAR Parchive

PET Priority Encoding Transmission

PHY PHYsical layer

PGM Pragmatic General Multicast

PMTU Path Maximum Transmission Unit

PRNG Pseudo-Random Number Generator

OSI Open Systems Interconnection

QC Quasi-Cyclique

RA Repeat-Accumulate

RAID Redundant Array of Inexpensive Disks

Raptor Raptor^{tm}

RaptorQ RaptorQ^{tm}

RFC Requests For Comments

RLC Random Linear Codes

RLCC Receiver Driven Layered Congestion-Control

RLE Run-Length Encoding

RMT Reliable Multicast Transport

RMTP Reliable Multicast Transport Protocol

RTCP Real-Time Control Protocol

RTP Real-Time Protocol

RTT Round Trip Time

RTSP Real-Time Streaming Protocol

RS Reed-Solomon

Reed-Solomon Reed-Solomon

SBN Source Block Number

SDO Standards Development Organization

SDP Session Description Protocol

SIP Session Initiation Protocol

SSE Streaming SIMD Extensions

SGE Structured Gaussian Elimination

SIMD Single Instruction Multiple Data

SPC Single Parity Check

SRM Scalable Reliable Multicast

SSM Source Specific Multicast

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

TOI Transport Object Identifier

UDP User Datagram Protocol

UEP Unequal Erasure Protection

UOD Universal Object Delivery

UL-FEC Upper-Layer Forward Erasure Correction

WEBRC Wave and Equation Based Rate Control

XML eXtensible Markup Language

XOR eXclusive OR

XTP Xpress Transport Protocol

Bibliography

[1] *IP Datacast over DVB-H: Content Delivery Protocols (CDP)*, December 2005. DVB BlueBook A101.

[2] *IPDC over DVB-H: Content Delivery Protocols (CDP) Implementation Guidelines*, March 2007. DVB Document A113.

[3] *Section 22.4 Cyclic Redundancy and Other Checksums*. Cambridge University Press, numerical recipes: the art of scientific computing (3rd ed.) edition, 2007.

[4] 3GPP. Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs. TS 26.346, 3rd Generation Partnership Project (3GPP).

[5] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. Priority Encoding Transmission. *IEEE Transactions on Information Theory*, 42(6), November 1996.

[6] H. Asaeda. *IGMP/MLD-Based Explicit Membership Tracking Function for Multicast*, February 2013. IETF PIM Working Group, Work in Progress: <draft-ietf-pim-explicit-tracking-05>.

[7] I. Blake and C. Studholme. *Properties of Random Matrices and Applications*, December 2006. Unpublished report available at `http://stoa.dhs.org/~cvs/coding/`.

[8] N. Bonello, S. Chen, and L. Hanzo. Low-density parity-check codes and their rateless relatives. *IEEE Communications Surveys and Tutorials*, 13(1):3–26, 2011.

[9] A. Bouabdallah and J. Lacan. Dependency-aware unequal erasure protection codes. *Journal of Zhejiang University - Science A*, 7(1), 2006.

[10] S. Boucheron and K. Salamatian. About Priority Encoding Transmission. *IEEE Transactions on Information Theory*, 46(2), March 2000.

[11] J. Boutros, O. Pothier, and G. Zemor. Generalized low density (tanner) codes. *IEEE International Conference on Communications (ICC'99)*, 1999.

[12] R. Braden. *Requirements for Internet Hosts – Communication Layers*, October 1989. IETF Request for Comments, RFC 1122 (Standard).

[13] G. Bumgardner. *Automatic Multicast Tunneling*, July 2013. IETF Network Working Group, Work in Progress: <draft-ietf-mboned-auto-multicast-15>.

[14] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. *SIGCOMM Computer Communication Review*, 28(4), October 1998.

[15] G. Carle and E. W. Biersack. Survey of error recovery techniques for IP-based audio-visual multicast applications. *IEEE Network Magazine*, 11(6), November 1997.

[16] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM symposium on Communications architectures and protocols (SIGCOMM'90)*, September 1990.

[17] M. Cotton, L. Vegoda, and D. Meyer. *IANA Guidelines for IPv4 Multicast Address Assignments*, March 2010. IETF Request for Comments, RFC 5771 (Best Current Practice).

[18] M. Cunche. *Codes AL-FEC hautes performances pour les canaux à effacements : variations autour des codes LDPC*. PhD thesis, Inria, Université de Grenoble, France, May 2010.

[19] M. Cunche and V. Roca. Optimizing the Error Recovery Capabilities of LDPC-staircase Codes Featuring a Gaussian Elimination Decoding Scheme. In *10th IEEE International Workshop on Signal Processing for Space Communications (SPSC'08)*, October 2008.

[20] M. Cunche and V. Roca. Le RFC 5170 en pratique : conception et évaluation d'un codec AL-FEC LDPC-staircase hautes performances. In *Colloque Francophone sur l'Ingénierie des Protocoles (CFIP)*, October 2009.

[21] M. Cunche, V. Savin, and V. Roca. Analysis of Quasi-Cyclic LDPC codes under ML decoding over the erasure channel. In *IEEE International Symposium on Information Theory and its Applications (ISITA'10) (http://arxiv.org/abs/1004.5217)*, April 2010.

[22] M. Cunche, V. Savin, V. Roca, G. Kraidy, A. Soro, and J. Lacan. Low-rate coding using incremental redundancy for GLDPC

codes. In *IEEE International Workshop on Satellite and Space Communications 2008 (IWSSC'08)*, October 2008.

[23] D. Divsalar, H. Jin, and R. J. Mceliece. Coding Theorems for "Turbo-Like" Codes. In *Proceedings Thirty-Sixth Annual Allerton Conference on Communication, Control, and Computing*, 1998.

[24] T. Strayer (ed.) at al. Xpress Transport Protocol Specification, XTP Revision 4.0, March 1995.

[25] E. O. Elliott. Estimates of error rates for codes on burst-noise channels. *Bell-System Technical Journal*, 42:1977–1997, September 1963.

[26] B. Fenner, B. Haberman, H. Holbrook, I. Kouvelas, and S. Venaas. *Multicast Source Notification of Interest Protocol (MSNIP)*, March 2011. IETF Network Working Group, Work in Progress: <draft-ietf-magma-msnip-06>.

[27] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6), December 1997.

[28] Huawei Technologies France. *Performance verification results for MBMS EFEC candidates r3*, January 2013. Tdoc S4-130061, TSG SA4.72 meeting.

[29] R. G. Gallager. *Low Density Parity Check Codes*. PhD thesis, MIT, Cambridge, Mass., September 1960.

[30] R. G. Gallager. Low Density Parity Check Codes. *IEEE Transactions on Information Theory*, 27(5), 1962.

[31] E. N. Gilbert. Capacity of a burst-noise channel. *Bell-System Technical Journal*, 39:1253–1265, March 1960.

[32] C. Guillemot and O. Le Meur. Image inpainting: Overview and recent advances. *IEEE Signal Processing Magazine*, January 2014.

[33] B. Haberman. *Allocation Guidelines for IPv6 Multicast Addresses*, August 2002. IETF Request for Comments, RFC 3307 (Standards Track).

[34] Qualcomm Inc. *MI-EMO: FLUTE Enhancements - Design Details*, September 2013. Tdoc S4-130911, TSG SA4.75 meeting.

[35] Qualcomm Inc. *MI-EMO: FLUTE Enhancements - Next Steps*, December 2013. Tdoc S4-AHI423, MBS SWG ad-hoc 28 conference.

[36] K. Jeacle, J. Crowcroft, M. P. Barcellos, and S. Pettini. Hybrid reliable multicast with tcp-xm. In *ACM Conference on Emerging Network Experiment and Technology (CoNEXT'05)*, October 2005.

[37] H. Jin, A. Khandekar, and R. McEliece. Irregular Repeat-Accumulate Codes. In *2nd. Int. Conference on Turbo Codes*, September 2000.

[38] S. J. Johnson. Introducing Low-Density Parity-Check Codes. Technical report, Department of Electrical and Computer Engineering, University of Newcastle, Australia, 2006.

[39] E. Kohler, M. Handley, and S. Floyd. *Datagram Congestion Control Protocol (DCCP)*, March 2006. IETF Request for Comments, RFC 4340 (Standards Track).

[40] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo. *Reed-Solomon Forward Error Correction (FEC) Schemes*, April 2009. IETF Request for Comments RFC 5510 (Standards Track/Proposed Standard).

[41] B. A. LaMacchia and A. M. Odlyzko. Solving Large Sparse Linear Systems over Finite Fields. In *Advances in Cryptology (Crypto'90), LNCS 537, Springer-Verlag*, 1991.

[42] A. Larzon, M. Degermark, S. Pink, L-E. Jonsson, and G. Fairhurst. *The Lightweight User Datagram Protocol (UDP-Lite)*, July 2004. IETF Request for Comments, RFC 3828 (Standard Track).

[43] M. Lentmaier and K.S. Zigangirov. On generalized low-density parity-check codes based on Hamming component codes. *IEEE Communications Letters*, 3(8):248–250, 1999.

[44] A. Li. RTP Payload Format for Generic Forward Error Correction, December 2007. IETF Request for Comments RFC 5109.

[45] M. Luby, T. Gasiba, T. Stockhammer, and M. Watson. Reliable Multimedia Download Delivery in Cellular Broadcast Networks. *IEEE Transactions on Broadcasting*, 53(1), March 2007.

[46] M. Luby, G. Gemmell, L. Vicisano, J. Crowcroft, and B. Lueckenhoff. *Asynchronous Layered Coding: a Scalable Reliable Multicast Protocol*, March 2000. IETF RMT, Work in Progress.

[47] M. Luby and V. Goyal. *Wave and Equation Based Rate Control (WEBRC) Building Block*, April 2004. IETF Request for Comments, RFC 3738 (Experimental).

[48] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. *Raptor Forward Error Correction Scheme for Object Delivery*, October 2007. IETF Request for Comments, RFC 5053 (Standards Track/Proposed Standard).

[49] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder. *RaptorQ Forward Error Correction Scheme for Object Delivery*, August 2011. IETF Request for Comments, RFC 6330 (Standards Track/Proposed Standard).

[50] M. Luby and T. Stockhammer. *Universal Object Delivery using RaptorQ*, March 2011. IETF RMT Working Group, Work in Progress: <draft-luby-uod-raptorq-00>.

[51] M. Luby, M. Watson, and L. Vicisano. *Layered Coding Transport (LCT) Building Block*, October 2009. IETF Request for Comments, RFC 5651 (Standards Track/Proposed Standard).

[52] M. Luby, M. Watson, and L. Vicisano. *Asynchronous Layered Coding (ALC) Protocol Instantiation*, April 2010. IETF Request for Comments, RFC 5775 (Standards Track/Proposed Standard).

[53] V. Lucas. *Contrôle de congestion équitable pour le multicast et interface avec le niveau applicatif*. PhD thesis, LSIIT, Université de Strasbourg - CNRS, France, Sep 2010.

[54] D. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, ISBN: 0521642981, 2003.

[55] K. Matsuzono, J. Detchart, M. Cunche, V. Roca, and H. Asaeda. Performance Analysis of a High-Performance Real-Time Application with Several AL-FEC Schemes. In *35th IEEE Conference on Local Computer Network (LCN'10)*, October 2010.

[56] F. Mattoussi, V. Roca, and B. Sayadi. Complexity Comparison of the Use of Vandermonde versus Hankel Matrices to Build Systematic MDS Reed-Solomon Codes. In *13th IEEE Int. Workshop on Signal Processing Advances in Wireless Communications (SPAWC 2012)*, June 2012.

[57] F. Mattoussi, V. Roca, and B. Sayadi. Design of Small Rate, Close to Ideal, GLDPC-Staircase AL-FEC Codes for the Erasure Channel. In *IEEE Global Communications Conference (GLOBECOM 2012)*, December 2012.

[58] F. Mattoussi, V. Roca, and B. Sayadi. *Impacts of the Packet Scheduling on the Performance of Certain Erasure Codes: Methodology and Application to GLDPC-Staircase*, October 2013. submitted for publication.

[59] F. Mattoussi, V. Savin, V. Roca, and B. Sayadi. Optimization with EXIT Functions of GLDPC-Staircase Codes for the BEC. In *13th IEEE Int. Workshop on Signal Processing Advances in Wireless Communications (SPAWC 2012)*, June 2012.

[60] F. Mattoussi, B. Sayadi, and V. Roca. Good Coupling Between LDPC-Staircase and Reed-Solomon for the Design of GLDPC Codes for the Erasure Channel. In *IEEE Wireless Communications and Networking Conference (WCNC'13)*, January 2013.

[61] M.-J. Montpetit, T. Mirlacher, and M. Ketcham. IPTV: An End to End Perspective. *Journal of Communications (Invited Paper)*, 5(5), 2010.

[62] M.-J. Montpetit and M. Médard. Community Viewing meets Network Coding: New Strategies for Distribution, Consumption and Protection of TV Content. *Second W3C Web and TV Workshop*, 2011.

[63] C. Neumann, V. Roca, A. Francillon, and D. Furodet. Impacts of Packet Scheduling and Packet Loss Distribution on FEC Performances: Observations and Recommendations. In *ACM Conference on Emerging Network Experiment and Technology (CoNEXT'05)*, October 2005.

[64] C. Neumann, V. Roca, and R. Walsh. Large scale content distribution protocols. *ACM Computer Communication Review (CCR) (invited paper)*, 35(5), October 2005.

[65] J. Nonnenmacher, E. W. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. *IEEE/ACM Transactions on Networking*, 6(4), August 1998.

[66] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh. *FLUTE - File Delivery over Unidirectional Transport*, October 2004. IETF RMT Working Group, Request For Comments, RFC 3926.

[67] T. Paila, R. Walsh, M. Luby, V. Roca, and R. Lehtonen. FLUTE - File Delivery over Unidirectional Transport, November 2012. IETF Request for Comments, RFC 6726 (Standards Track).

[68] S. Paul, K. K. Sabnaniand J. C-H. Lin, and S. Bhattacharyya. Reliable multicast transport protocol rmtp. *IEEE Journal on Selected Areas in Communications*, 15(3), April 1997.

[69] C. Perkins and O. Hodson. Options for Repair of Streaming Media, June 1998. IETF Request for Comments, RFC 2354.

[70] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Technical Report CS-96-332, University of Tennessee, July 1996.

[71] C. Pomerance and J. W. Smith. Reduction of huge, sparse matrices over finite fields via created catastrophes. *Experimental Mathematics, Vol. 1, No. 2*, 1992.

[72] V. Roca. On the Use of On-Demand Layer Addition (ODL) with Multi-Layer Multicast Transmission Techniques. In *2nd Workshop on Networked Group Communication (NGC2000)*, November 2000.

[73] V. Roca. Simple Authentication Schemes for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols, April 2012. IETF Request for Comments, RFC 6584 (Standards Track).

[74] V. Roca and B. Adamson. *FCAST: Object Delivery for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols*, July 2013. IETF RMT Working Group, IETF Request for Comments, RFC 6968 (Experimental).

[75] V. Roca, M. Cunche, and J. Lacan. *LDPC-Staircase Forward Error Correction (FEC) Scheme for FECFRAME*, December 2012. IETF FECFRAME Working Group, IETF Request for Comments, RFC 6816 (Standards Track).

[76] V. Roca, M. Cunche, J. Lacan, A. Bouabdallah, and K. Matsuzono. *Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME*, February 2013. IETF FECFRAME Working Group, IETF Request for Comments, RFC 6865 (Standards Track).

[77] V. Roca, M. Cunche, C. Thienot, J. Detchart, and J. Lacan. RS + LDPC-Staircase codes for the erasure channel: Standards, usage and performance. In *9th IEEE Int. Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2013)*, August 2013.

[78] V. Roca, A. Francillon, and S. Faurite. *TESLA source authentication in the ALC and NORM protocols*, April 2010. IETF Request for Comments, RFC 5776 (Experimental).

[79] V. Roca and B. Mordelet. Design of a multicast file transfer tool on top of ALC. In *7th IEEE Symposium on Computers and Communications (ISCC'02), Toarmina, Italy*, July 2002.

[80] V. Roca and C. Neumann. Design, Evaluation and Comparison of Four Large Block FEC Codecs, LDPC, LDGM, LDGM-Staircase and LDGM-Triangle, plus a Reed-Solomon Small Block FEC Codec. Research Report 5225, INRIA, June 2004.

[81] V. Roca, C. Neumann, and D. Furodet. *Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes*, June 2008. IETF Request for Comments, RFC 5170 (Standards Track/Proposed Standard).

[82] V. Roca, A. Roumy, and B. Sayadi. *The Generalized Object Encoding (GOE) Approach for the Forward Erasure Correction (FEC) Protection of Objects and its Application to Reed- Solomon Codes over GF(2x)*, March 2012. IETF RMT Working Group, Work in Progress: <draft-roca-rmt-goe-fec-01.txt>.

[83] V. Roca, A. Roumy, and B. Sayadi. *The Generalized Object Encoding (GOE) LDPC-Staircase FEC Scheme*, July 2012. IETF RMT Working Group, Work in Progress: <draft-roca-rmt-goe-ldpc-01.txt>.

[84] R. M. Roth and G. Seroussi. On generator matrices of MDS codes. *IEEE Transactions on Information Theory*, 31(6), nov 1985.

[85] A. Roumy, V. Roca, and B. Sayadi. Memory Consumption Analysis for the GOE and PET Unequal Erasure Protection Schemes. In *IEEE International Conference on Communications*, February 2012.

[86] A. Roumy, V. Roca, B. Sayadi, and R. Imad. Unequal erasure protection and object bundle protection with the generalized object encoding approach, July 2011.

[87] L. Salamanca, P.M. Olmos, J.J. Murillo-Fuentes, and F. Perez-Cruz. MAP decoding for LDPC codes over the binary erasure channel. In *IEEE Information Theory Workshop (ITW)*, October 2011.

[88] S. Salsano. Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel. Technical report, Technical report, University of Roma, http://netgroup.uniroma2.it/twiki/bin/view.cgi/Main/NetemCLG, August 2012.

[89] 3GPP Technical Specification Group Services and System Aspects. *Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs (Release 6)*, March 2006. 3GPP TS 26.346 v6.4.0.

[90] A. Shokrollahi. LDPC codes: An introduction. *Digital Fountain, Inc., Technical Report*, April 2003.

[91] M. A. Shokrollahi and M. Luby. Raptor codes. *Foundations and Trends in Communications and Information Theory*, 6(3-4):213–322, May 2011.

[92] A. Soro, M. Cunche, J. Lacan, and V. Roca. Erasure Codes with a Banded Structure for Hybrid Iterative-ML Decoding. In *IEEE Global Communications Conference (GLOBECOM 2009)*, November 2009.

[93] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano. PGM Reliable Transport Protocol Specification, December 2001.

[94] E. Stauffer, B. Shen, S. Chakraborty, D. Tujkovic, J. Huang, S. Shet, and K. Rath. *Supercharged Codes*, September 2013. IETF RMT Working Group, Work in Progress: <draft-stauffer-rmt-bb-fec-supercharged-03>.

[95] T. Stockhammer. Dynamic Adaptive Streaming over HTTP – Design Principles and Standards. *Second W3C Web and TV Workshop*, 2011.

[96] C. Studholme and I. Blake. Windowed Erasure Codes. *IEEE Int. Symposium on Information Theory*, pages 509–513, July 2006.

[97] C. Studholme and I. Blake. Random Matrices and Codes for the Erasure Channel. *Algorithmica*, 56(4), April 2008.

[98] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

[99] R. Michael Tanner. On quasi-cyclic repeat-accumulate codes. In *in Proc. 37th Allerton Conf. Communication, Control and Computing*, pages 249–259, 1999.

[100] C. A. Taylor. *Code Poetry and Text Adventures*, 2013. http://catid.mechafetus.com/news/news.php.

[101] P.-U. Tournoux, E. Lochin, J. Lacan, A. Bouabdallah, and V. Roca. On-the-fly erasure coding for real-time video applications. *IEEE Transactions on Multimedia*, 13(4), August 2011.

[102] T. Turletti and C. Huitema. Videoconferencing on the internet. *IEEE/ACM Transactions on Networking*, 4(3), June 1996.

[103] R. Urbanke and A. D. Wyner. Packetizing for the Erasure Broadcast Channel with an Internet Application. In *International Conf. on Combinatorics, Information Theory and Statistics*, 1997.

[104] M. Watson, A. Begen, and V. Roca. *Forward Error Correction (FEC) Framework*, June 2011. IETF Request for Comments, RFC 6363 (Standards Track/Proposed Standard).

[105] M. Watson, M. Luby, and L. Vicisano. *Forward Error Correction (FEC) Building Block*, August 2007. IETF Request for Comments, RFC 5052 (Standards Track/Proposed Standard).

[106] J. Zinky, A. Caro, and G. Stein. *Random Binary FEC Scheme for Bundle Protocol*, August 2012. IETF Network Working Group, Work in Progress: <draft-irtf-dtnrg-zinky-random-binary-fec-scheme-00>.

[107] M. Zivkovic. On Two Probabilistic Decoding Algorithms for Binary Linear Codes. *IEEE Transactions on Information Theory*, 37(6):17707–1716, November 1991.

[108] V. V. Zyablov and M. S. Pinsker. Decoding complexity of low-density codes for transmission in a channel with erasures. *Probl. Peredachi Inf.*, 48:18–28, 1974.

Résumé : Ce document discute de la fourniture de services de diffusion où des contenus de toute nature doivent être transmis efficacement à un ensemble de récepteurs. Plusieurs problèmes se posent :

- la robustesse face aux problèmes de transmission, qu'il s'agisse d'erreurs (inversion de bits) ou de pertes (de paquets);
- l'efficacité du service fourni, en termes de performances de correction d'erreurs et/ou effacements, et de faible complexité algorithmique;
- le respect d'éventuelles contraintes temps réelles de certains contenus;
- et enfin le passage à l'échelle lorsque le nombre de récepteur augmente significativement.

La problématique est assez large et nous nous "limitons" à deux aspects et composants essentiels que sont :

- les codes correcteurs de niveau applicatif, ou AL-FEC, pour le canal à effacements, sans lesquels l'efficacité du service serait fortement compromise, et
- les protocoles de transport qui définissent les règles d'échanges d'informations entre soure(s) et récepteur(s) et garantissent le niveau de fiabilité nécessaire.

Tout au long de ce document, nous détaillons ces deux axes et positionnons nos contributions et leur usage effectif dans le panorama actuel.

Summary : This work focuses on the provision of broadcast services where contents of any kinds must be transmitted in an efficient manner to a set of receivers. Several problems arise:

- the robustness in front of transmission issues, no matter whether it consists of errors (bit inversion) or erasures (of packets);
- the efficiency of the provided service, in terms of error and/or erasure recovery performance and low algorithmic complexity;
- the fulfilment of the potential real-time constraints of some contents;
- and finally the scalability of the service when the number of receivers significantly increases.

The problem space is rather large and we "limit ourself" to two essential components, namely:

- the application-level correction codes, of AL-FEC, for the erasure channel and without which the efficiency of the service would be largely compromised, and
- the transport protocols that define the rules for the exchange of information between the source(s) and the destination(s), and provide the required reliability.

Throughout this document, we detail these two axes and position our contributions and their actual usage in the current situation.