Vérification d'intégrité et codes correcteurs d'erreurs de niveau applicatif (AL-FEC)

Mathieu Cunche

CONFIDENTIEL

Rapport de projet de fin d'études

INRIA Rhônes-Alpes, Projet PLANETE,

ENSIMAG, Master 2 CSCI.

26 septembre 2006

Mathieu Cunche
mathieu.cunche@ensimag.imag.fr

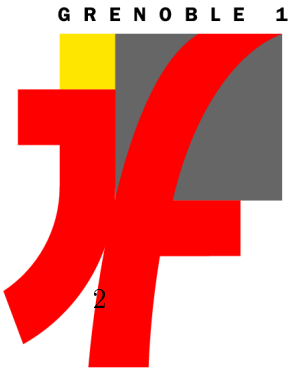
Maître de Stage : Vincent Roca

vincent.roca@inrialpes.fr


INRIA


PLANETE


INP Grenoble
ENSIMAG


GRENOBLE 1
UNIVERSITE
JOSEPH FOURIER
SCIENCES. TECHNOLOGIE. MEDECINE

Résumé

La diffusion de contenu à grande échelle est actuellement en plein essor. Ces nouveaux types de distribution d'information font usage de codes correcteurs d'erreurs au niveau applicatif (c'est-à-dire opérant dans la couche transport ou session de la pile OSI), par exemple de type LDPC. Ce type de diffusion peut également nécessiter l'utilisation d'autres fonctionnalités telles que le chiffrement ou la vérification d'intégrité. Dans le but d'économiser les ressources des plate-formes impliquées nous avons étudié la faisabilité d'un système utilisant les propriétés du code correcteur afin de *détecter des corruptions de données durant la transmission, avec ou non localisation des symboles corrompus*. La spécificité du travail est que l'on ne cherche pas à protéger chaque symbole transmis par un hash, mais au contraire à procéder de façon statistique, seul un sous ensemble réduit des symboles étant vérifiés.

Cette approche a fourni des résultats très prometteurs. En effet, nous avons identifié un phénomène de dispersion des corruptions au sein du codec LDPC, phénomène qui nous a ensuite permis de développer un prototype aux performances intéressantes. Ce système permet en effet de détecter les attaques lors de la transmission avec une faible probabilité d'erreurs, pour un coût très inférieur à une vérification par une fonction de hachage.

Ces résultats nous ont conduit à étudier ensuite un système permettant de localiser les données corrompues lors de la transmission. Il s'agit donc de la troisième contribution de ce travail, qui doit encore être finalisé.

Table des matières

Introduction

Ce document rapporte le travail que j'ai effectué au sein de l'équipe PLANETE dans la période de Mars à Juillet 2006. Il fait office de rapport de projet de fin d'études en vue de l'obtention du Master2 Professionnel (DESS) Cryptologie, Sécurité et Codage de l'Information de l'Université Joseph Fourier à Grenoble. Une version antérieure de ce document à été présenté en vue de l'obtention du diplôme d'ingénieur de l'ENSIMAG (École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble).

Nous assistons aujourd'hui à un développement rapide des technologies de diffusion de contenu (3G, DVB-H). Différentes solutions technologiques ont été apportées pour répondre aux problématiques de la transmission de contenu à grande échelle. Parmi ces solutions on peut noter l'utilisation de codes correcteurs pour palier aux problèmes des erreurs intervenant sur les canaux de transmissions, mais aussi de fonctions cryptographiques qui sont utilisées pour authentifier le contenu distribué ou pour en restreindre son accès aux seules personnes autorisées.

Ces applications impliquent souvent des plate-formes à ressource limitées (téléphone portable, PDA ...), il peut donc sembler opportun de faire fusionner plusieurs de ces opérations en une seule, dans le but d'économiser ces ressources. Les travaux effectués visaient à regrouper les opérations liées aux codes correcteurs avec les opérations de vérification d'intégrité et d'authentification.

Dans une première partie je présenterai les technologies impliquées, puis je présenterai la proposition d'ajout de fonctionnalité de détection de corruption à un codec de codes correcteurs, je terminerai par donner un aperçu d'un système permettant de localiser les données corrompues.

Chapitre 1

Cadre du projet de fin d'étude

1.1 l'INRIA

L'INRIA, Institut National de Recherche en Informatique et en Automatique est un organisme public civil de recherche français. Sous la tutelle du Ministère de la Recherche et du Ministère de l'Industrie, il a pour ambition de mettre en réseau les compétences et talents de l'ensemble du dispositif de recherche français, dans le domaine des sciences et technologies de l'information.

L'INRIA possède 6 unités de recherche situées à Rocquencourt, Rennes, Sophia Antipolis, Grenoble, Nancy et Bordeaux, Lille, Saclay. 3600 personnes, principalement scientifiques, travaillent au sein de l'institut sur plus de 138 projets.

Le site de Grenoble dans lequel j'ai effectué mon stage est situé au sein de la ZIRST (Zone pour l'Innovation et les Réalisations Scientifiques et Techniques) de Montbonnot et accueille plus de 400 personnes. Les projets de recherche sont principalement centrés autour de 3 thèmes :

- Les systèmes communicants (Web, réseaux, Internet ...)
- Les systèmes cognitifs (image, vidéo, réalité virtuelle ...)
- Les systèmes numériques (grilles de calcul, modélisation ...)

1.2 Le projet PLANETE

PLANETE (Protocoles et applications pour l'Internet) est un projet bilocalisé à l'INRIA Sophia Antipolis et à l'INRIA Rhône-Alpes. Ses activités sont centrées sur la conception, l'implémentation et l'évaluation de protocoles et d'applications Internet. Les axes de recherche balaient diffé-

rents domaines tels que la sécurité dans les réseaux sans architecture et avec contraintes, la communication de groupe "scalable", l'analyse de la dynamique des réseaux pair-à-pair, l'impact de l'hétérogénéité des réseaux sur les performances des protocoles. Les besoins des utilisateurs et les progrès technologiques ont amené à une hétérogénéité tant au niveau des infrastructures réseaux (ATM, réseaux sans fils, ADSL, réseaux mobiles Ad-hoc, etc ..) qu'au niveau des terminaux (mobiles ou fixes, avec des grandes puissances de calcul, des ressources limitées, etc ...). Dans le même temps l'apparition de nouvelles fonctionnalités sur les services internet pose de nouveaux problèmes liés à la mobilité et la "scalabilité". L'équipe privilégie une approche d'optimisation inter-couche pour rendre possible l'adaptation à toutes les couches des protocoles de transmission, pour permettre la reconfiguration des micro-terminaux et pour faciliter la mise à jour des protocoles implantés.

L'équipe collabore avec plusieurs partenaires académiques tels que l'UCL, l'UCI, le MIT, l'Université de Berne, l'ENS et aussi avec des partenaires industriels comme Alcatel, STMicroelectronics, FT R&D, Hitachi, Intel, Motorola, etc ...

Mon projet s'est déroulé dans le groupe de l'INRIA Rhône-Alpes. Ce groupe est actuellement composé de 2 chercheurs permanents et d'un doctorant auquel venait s'ajouter trois stagiaires. Durant ce stage j'ai pu enrichir mes connaissances sur les thèmes de recherche de l'équipe. En particuliers sur :

- les codes LDPC et la diffusion de contenu à grande échelle, thèmes de recherche de mon maître de stage Vincent Roca (chercheur permanent)
- la sécurité des réseaux de capteurs, sujet sur lequel Claude Castelucia (chercheur permanent), Aurélien Francillon (doctorant) et Jose Esparza (stagiaire ENSIMAG) travaillaient.
- le cryptage de la voix sur IP (VOIP), sujet étudié par Nicolas Bernard (stagiaire ENSIMAG)

1.3 Sujet du stage

Le sujet du stage était l'étude de codes hybrides. Dans un premier temps j'ai d'ajouté des fonctionnalités de vérification d'intégrité des données pour un faible coût en temps CPU au codec existant. Ce codec a été baptisé Vari-FEC. Après la découverte de propriétés intéressantes du codec, j'ai concentré mes efforts sur la mise au point d'un système permettant une discrimination des symboles corrompus.

Chapitre 2

Revue des technologies

2.1 Protocoles de transmission

Lors de la transmission de données sur un média, il peut arriver qu'une partie de l'information soit perdue ou altérée. Dans le cas des réseaux d'information comme internet, il n'est pas certain que le récepteur reçoive la totalité des paquets de données envoyés par l'émetteur. Une surcharge des infrastructures ou un média défaillant peuvent être à la source de ces pertes. Des protocoles comme TCP (Transmission Control Protocol) basés sur un système d'acquittement compensent ces erreurs de transmissions, en permettant à l'émetteur de renvoyer les paquets qui ont été perdus jusqu'à ce qu'ils aient pu être correctement reçus par le destinataire.

Dans le cadre d'une diffusion à grande échelle de type broadcast il est impossible d'utiliser ce genre de protocole, le nombre de récepteurs étant trop grand pour que l'émetteur puisse traiter au cas par cas les retransmissions des paquets. On préfère alors l'utilisation d'un protocole sans système d'acquittement comme le protocole UDP (User Datagram Protocol). Pour compenser les pertes de paquets nous disposons de deux solutions :

1. Un système de retransmission automatique (Automatic Repeat Request) des paquets perdus, utilisé au dessus d'UDP. Mais cette approche a des limites en terme de passage à l'échelle.
2. L'utilisation de codes correcteurs d'erreurs aussi appelés codes FEC (Forward Error Correction), qui permet la transmission d'informations redondantes avec les données originales. Ces informations redondantes sont utilisées pour reconstruire les données perdues lors de la transmission.

2.2 Les codes correcteurs d'erreurs

Comme leur nom l'indique, ces codes sont utilisés pour corriger les erreurs sur les données. Les codes Reed-Solomon, les codes de Hamming sont des exemples connus de codes correcteurs. Les codes LDPC (Low Density Parity Check) font eux aussi partie de la famille des codes FEC.

D'une manière générale les codes correcteurs ajoutent de la redondance à l'information à transmettre. Cette information redondante servira lors du décodage pour corriger les erreurs liées à la transmission. La correction des erreurs de transmission n'est pas leur seule utilisation, en effet ils sont aussi couramment utilisés pour réparer les erreurs pouvant apparaître sur des supports de stockage de données (CD-ROM, RAID).

L'information que nous voulons transmettre est représentée par une suite x de symboles appartenant à un alphabet A. L'encodage transforme ce message en une suite y de symboles appartenant à A'(A'pouvant être différent de A). On note E l'application d'encodage.

Il existe deux classes de codes : les codes par blocs et les codes convolutifs.

Les codes par bloc prennent en entrée un bloc de taille k appartenant à A^k et l'encodent en un bloc de taille n appartenant à A'^n(n est aussi appelé la longueur du code). On a donc $E : A^k \rightarrow A'^n$. Par la suite nous utiliserons le terme *fec_ ratio* pour désigner le rapport $\frac{n}{k}$.

Les codes convolutifs travaillent sur des blocs de taille quelconque ce qui revient à $E : A^{\mathbb{N}} \rightarrow A'^{\mathbb{N}}$.

2.3 Modèles de canal de transmission

Afin de mieux comprendre la problématique des erreurs de transmission, on utilise une modélisation de ces canaux. Il existe deux modèles simples de canal de transmission. Pour simplifier la description, les messages transmis sont à valeur dans {0,1}.

Le canal binaire à effacement ou BEC (Binary Erasure Channel) est un canal qui efface un message transmis avec la probabilité ε. On représente cela en ajoutant une troisième valeur possible à la sortie du canal. Cette valeur * est appelée effacement. Quelque soit la valeur du message en entrée du canal, il gardera sa valeur avec la probabilité 1-ε et il sera effacé avec la probabilité ε.

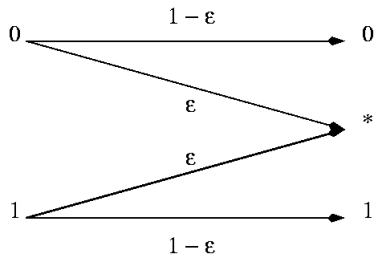


FIG. 2.1 – Canal binaire à effacement

Le canal binaire symétrique ou BSC (Binary Symmetric Channel) peut changer la valeur du message transmis. Ainsi le message *1* (resp *0*) aura la probabilité p d'être changé en *0* (resp *1*) et la probabilité 1-p de ne pas être modifié.

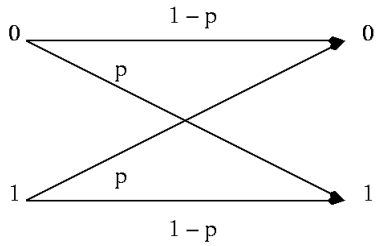


FIG. 2.2 – Canal binaire symétrique

Il faut préciser que dans le cadre de ce projet :
- Les transmissions s'effectuent sur un canal à effacement ;
- Les notions de paquets et de symboles sont équivalentes car il s'agit de codes correcteurs travaillant au niveau application (AL-FEC).

2.4 Les codes LDPC

2.4.1 Principes

Les codes LDPC sont des codes qui ont été inventés par Robert Gallager en 1962[8]. Ils ont ensuite été oubliés puis redécouverts 30 ans plus tard lorsque MacKay a montré qu'ils offraient des performances proches de celles des turbo-codes[9].

Les codes LDPC forment une classe de codes linéaires par blocs obtenus à partir d'un graphe bipartie. Soit G un graphe bipartie avec n noeuds de

gauche appelés noeuds-messages et r noeuds de droite appelés noeuds de vérification.

En associant chaque symbole à chaque noeud-message, ce graphe G donne naissance à un code linéaire de longueur n et de dimension au moins $n-r$. Ce code est obtenu en prenant les vecteurs $(x_1, x_2, ..., x_n)$ composés de symboles, et qui sont tels que pour tout noeud de vérification, la somme de tout les symboles des noeuds-messages adjacents est nulle.

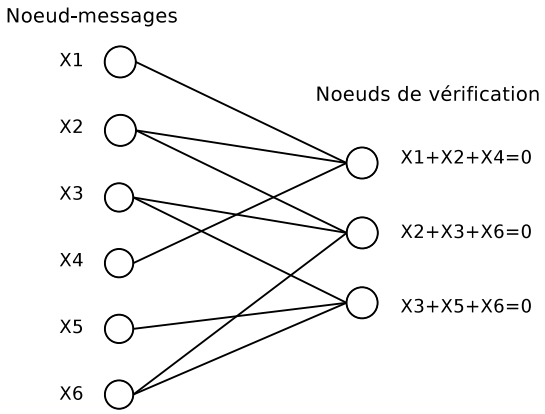


FIG. 2.3 – Graphe bipartie pour code LDPC

Encodage

Le vecteur $(x_1, x_2, ..., x_n)$ est en fait composé de k symboles sources $\{s_i\}_{1 \leq i \leq k}$ et de $n-k$ symboles de parités $\{p_i\}_{1 \leq i \leq n-k}$. Les symboles sources constituant le message à coder, on construit les symboles de parité de manière à ce que les conditions au niveau des noeuds de vérification soient respectées.

Décodage sur un canal symétrique

Les algorithmes de décodage sont des algorithmes itératifs, à chaque itération on utilise la valeur d'un symbole reçu pour faire progresser le décodage.

Il existe une classe d'algorithme de décodage appelé algorithme à transmission de message (message passing algorithms). A chaque itération de décodage, ces algorithmes font passer l'information des noeud-messages vers les noeuds de vérification puis des noeuds de vérification vers les noeuds-messages.

Une première sous-classe de ces algorithmes de décodages regroupe les algorithmes à "propagation de confiance" (belief propagation). Ces algorithmes

transmettent non pas des valeurs, mais des probabilités de valeurs. En effet l'information transmise d'un noeud-message à un noeud de vérification, est la probabilité que le noeud-message ait une certaine valeur, sachant la valeur de ce noeud et les informations communiquées au cours des itérations précédentes à ce noeud-message. Le décodage se poursuit jusqu'à ce que la probabilité des valeurs des noeuds-messages atteigne une valeur jugée suffisante.

Décodage sur un canal à effacement

Lorsque l'on utilise les codes LDPC sur un canal à effacement, il existe un algorithme de décodage relativement simple permettant de reconstruire l'information perdue. Sur ce type de canal nous transmettons un ensemble de symboles source et de symboles de parité, ce canal va effacer une partie de ces symboles transmis. Voici l'algorithme permettant de reconstruire les symboles effacés à partir du graphe G :

1. Initialiser tous les noeuds de vérification à 0
2. Pour chaque symbole S reçu, u étant le noeud-message correspondant, additionner sa valeur à chaque noeud de vérification adjacent au noeud-message u. Puis retirer u ainsi que tous ses arcs.
3. S'il y a un noeud de vérification v de degré 1, soit u l'unique noeud-message auquel il est relié et S le symbole correspondant à u. Donner la valeur de v au symbole S. Retirer v du graphe. Ajouter la valeur de S à tous les noeuds de vérification adjacents à u, puis retirer u ainsi que ses arcs de G.

Le coût en nombre d'opérations d'un tel algorithme est proportionnel au nombre d'arcs du graphe G. De plus si G est un graphe "clairsemé" (sparse), c'est-à-dire que le nombre d'arcs transversaux est petit, le temps de décodage est linéairement proportionnel à la longueur du code. Cependant il n'est pas garanti que l'on puisse décoder l'ensemble des symboles.

Représentation sous forme de matrice

Dans la plupart des cas on utilise une matrice H appelée matrice de parité pour représenter le graphe bipartie G. Chaque ligne de cette matrice représente un noeud de vérification et chaque colonne correspond à un symbole. Cette matrice est remplie de *0* et de *1*. Un *1* à la position (i, j) signifiant que le noeud de vérification i est relié par un arc au noeud-message j.

Cette représentation permet une interprétation simple du code. Par exemple, supposons que la ligne i de H ne contienne que des *0* sauf aux positions $\{j_1, j_{2,}, j_3, j_4, j_5\}$ où l'on trouve des *1*. En notant $\{x_1, x_2, x_3, x_4, x_5\}$ les symboles correspondant aux colonnes $\{j_1, j_{2,}, j_3, j_4, j_5\}$, on peut en déduire l'équation satisfaisant la condition du noeud de vérification i :

$x_1 + x_2 + x_3 + x_4 + x_5 = 0$

Ainsi si les symboles $\{x_1, x_2, x_3, x_4\}$ sont connus et que x_5 est inconnu alors on peut le reconstruire facilement de la manière suivante :

$x_5 = x_1 + x_2 + x_3 + x_4$

Dans la pratique on utilise la matrice de parité H pour définir le code. Ces matrices peuvent être de différents types. De manière générale elles sont composées d'une partie gauche contenant les colonnes correspondant aux symboles sources et d'une partie droite contenant les colonnes des symboles de parité. Je vais présenter quelques types de codes et le type matrice de parités qui les génère. Dans ces codes il y a autant de noeuds de vérification que de symboles de parité, il y a donc autant de lignes dans la matrice, qu'il y a de symboles de parité $(n - k)$.

2.4.2 Exemples de codes

Codes LDGM

Les codes LDGM *Low Density Generator Matrix* sont des codes linéaires (n ; k) par blocs. Un tel code ne fait intervenir qu'un seul symbole de parité par ligne. La matrice de parité H de ce code est la concaténation d'une matrice H_1 (partie symbole source) et d'une matrice identité Id_{n-k}.En notant s_i un symbole source et p_j un symbole de parité, pour ce code on obtient des équations du type : $s_1 + s_2 + s_3 + s_4 + p_1 = 0$

Nous présentons ici un exemple de code LDGM avec k=6 et n=9. Nous avons donc 6 symboles source avec 3 symboles de parité ce qui nous donne une matrice H de taille (3,9).

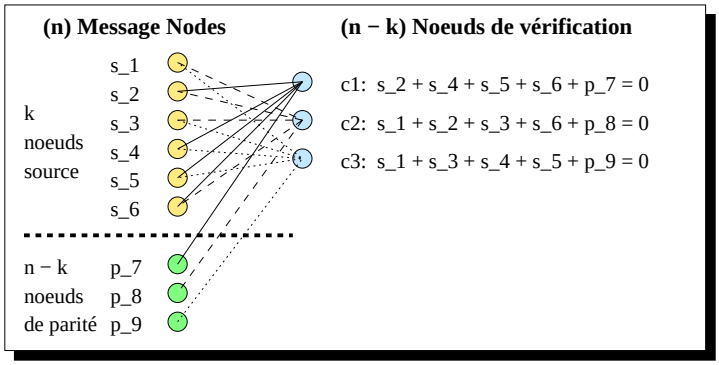


FIG. 2.4 – graphe bipartie LDGM

$$[\,H_1 \mid Id_3\,] = \left(\begin{array}{cccccc|ccc} s_1 & & \ldots & & & s_6 & p_7 & \ldots & p_9 \\ 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \end{array}\right) \begin{array}{c} c_1 \\ c_2 \\ c_3 \end{array}$$

FIG. 2.5 – matrice de parité LDGM

Codes LDPC-Staircaise

Ces codes sont proches des codes LDGM. En effet la seule différence est que la matrice de droite qui concerne les symboles de parité est remplacée par une matrice escalier, on a donc deux symboles de parité qui interviennent dans les équations (sauf pour la première). On obtient des équations de la forme : $s_1 + s_2 + s_3 + s_4 + p_1 + p_2 = 0$.

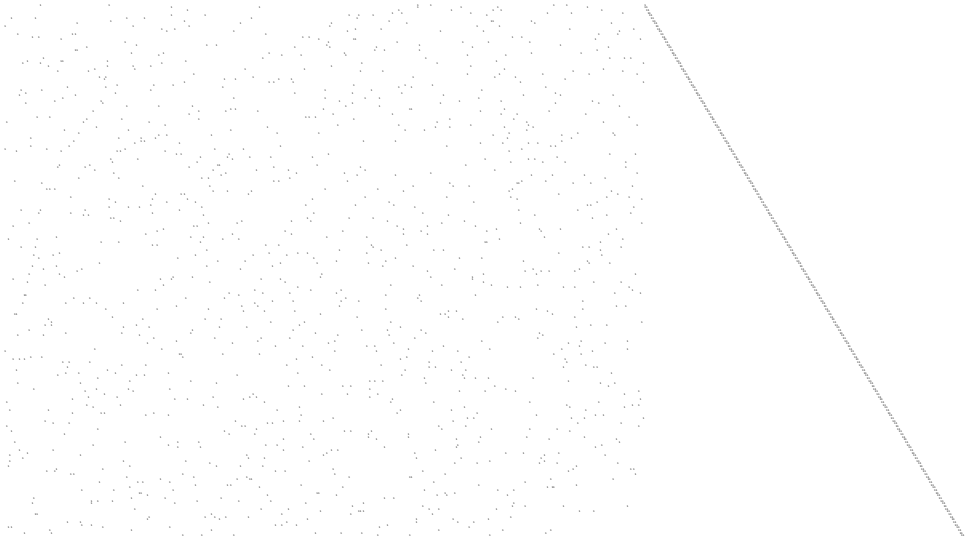


FIG. 2.6 – Une matrice de parité pour un code LDPC-Staircaise (k=400,n=600)

Codes LDPC-Triangle

Ces codes, qui ont été inventés au sein de l'équipe Planète [1], sont une autre variante des codes LDGM. Dans ces codes la matrice de droite est remplacée par une matrice triangulaire inférieure. Cette matrice est en fait une matrice escalier à laquelle on a ajouté des *1* sous la diagonale, la répartition des *1* sous la diagonale suivant une loi qui a été étudiée pour optimiser les performances du code.

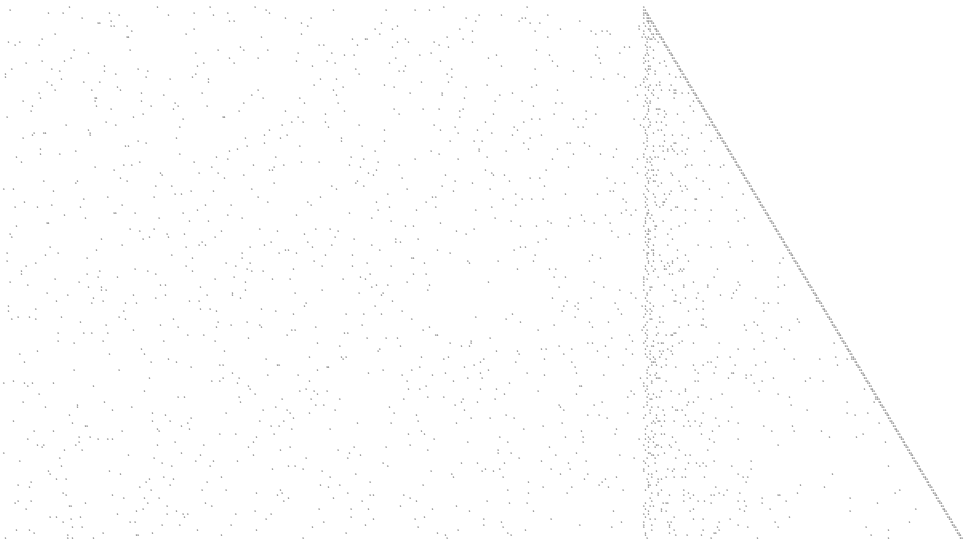


FIG. 2.7 – Une matrice de parité pour un code LDPC-Triangle (k=400,n=600)

2.4.3 Transmission des symboles

Après un encodage avec un code LDPC, nous possédons un ensemble de symboles source et de symboles de parité. Vient alors la question de l'envoi de ces symboles : quel ordre d'émission ? Quels symboles envoyer ?

Il existe de nombreuses configurations possibles qui possèdent leurs avantages et leurs inconvénients. On peut envoyer tout les symboles dans un ordre aléatoire, ou bien une partie des symboles source suivit de l'ensemble des symboles de parité. Le choix se fera en fonction des caractéristiques du canal de transmission et des besoins de l'utilisateur.

Le récepteur va recevoir un ensemble de symboles, il va en utiliser un nombre p pour effectuer le décodage. On définit alors la grandeur $\frac{p}{k}$ que l'on nomme le ratio d'inefficacité. Cela représente le surcoût du décodage en terme de symbole. Les codes LDPC-Triangle et LDPC-Staircase permettent d'obtenir des ratios d'inefficacité entre 1.08 et 1.1. Cela signifie qu'on utilise

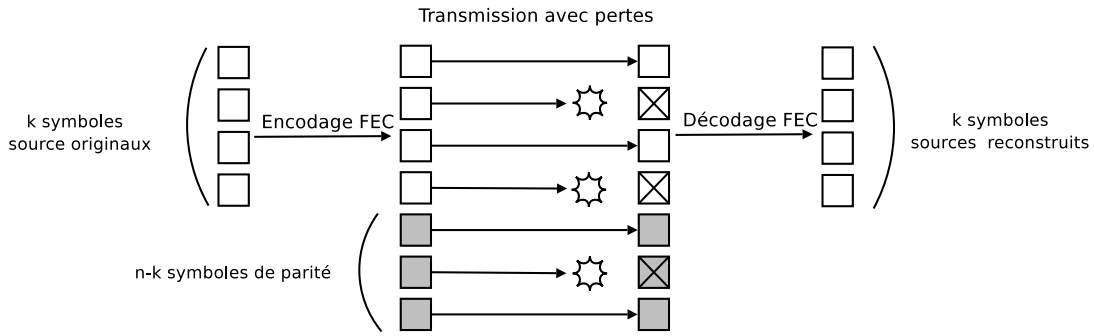


FIG. 2.8 – Schéma d'utilisation des codes FEC sur un canal avec perte

en moyenne 8 à 10% de symboles de plus que le nombre de symboles de l'objet source pour la reconstruction de celui-ci.

2.4.4 Utilisation

Ces codes sont particulièrement adaptés pour les transmissions sur un canal avec perte. On les utilise au niveau applicatif sur des fichiers (ou plus généralement objets) de taille importante, tandis que d'autres types de codes correcteurs corrigent d'éventuelles erreurs sur les couches inférieures (au niveau paquet IP par exemple). Au niveau applicatif ils permettent de réparer les pertes de paquets dûes à une coupure de la connexion, qui sont des phénomènes fréquents dans les réseaux sans fils. Ces codecs sont d'ailleurs intégrés à un ensemble de logiciels qui permettent une diffusion de type multicast sur un réseaux sans fils.

Un exemple d'utilisation est la réception de vidéo sur des plate-formes mobiles tels que des PDA ou des téléphones cellulaires. Ces plate-formes disposant des capacités limitées en calcul et en énergie, nous nous efforcerons de minimiser la consommation de ces ressources. Le fichier étant destiné à un grand nombre d'utilisateurs, il est broadcasté.

La perte de paquets peut alors intervenir lorsque le portable passe sous un tunnel. Dans ce cas, l'utilisateur n'aura pas reçu une partie des paquets (cad des symboles). C'est là que le code LDPC intervient : il va permettre de reconstruire les symboles manquant grâce à l'ensemble de symboles qu'il aura reçus. Ainsi l'utilisateur aura pû reconstituer le fichier original même s'il a manqué une partie de l'émission.

Chapitre 3

VeriFEC : vérification d'intégrité probabiliste

3.1 Présentation

3.1.1 Objectif

Nous souhaitons apporter une fonctionnalité de détection de corruption au codec LDPC existant. Cette fonction doit nous permettre de détecter après décodage, si un attaquant a corrompu un ou plusieurs symboles qui ont été reçus et donc l'objet décodé. Pour cela nous allons tirer partie de certaines propriétés de l'algorithme de décodage.

3.1.2 Modèle de l'attaquant

Possibilités de l'attaquant :

Nous supposons que l'attaquant est capable :
- d'intercepter tous les paquets en provenance de l'émetteur ;
- d'envoyer des paquets choisis au récepteur ;
- de corrompre ou de retransmettre sans erreur les paquets ;
- de choisir librement l'ordre de transmission des paquets.

L'attaquant est donc en coupure entre émetteur et récepteur et peut tout faire.

Objectifs de l'attaquant :

Le but de l'attaquant est de corrompre les données transmises de l'émetteur au récepteur.

Le premier type d'attaque serait de corrompre un maximum de données. Cette attaque est triviale puisqu'il peut stopper tous les paquets à destination du récepteur, et renvoyer des données aléatoires à la place.

L'attaquant peut aussi choisir d'être plus subtile en voulant corrompre une petite partie des données, tout en essayant de ne pas faire remarquer son attaque au récepteur.

3.2 Observations et analyse du problème

3.2.1 Le phénomène de propagation des corruptions

Approche théorique

Le décodeur FEC reconstruit les données manquantes avec les données sources et les données redondantes. De ce fait, un symbole reçu peut être utilisé pour reconstruire plusieurs symboles manquants. L'algorithme de décodage utilisant des opérations $\oplus$ pour reconstruire les symboles, les symboles reconstruits vont hériter aussi de cette corruption.

Notons $S_i^\diamond$la valeur corrompue du symbole S_i, et S_i^*la valeur correcte de ce symbole.

Supposons que nous avons reçu la valeur corrompue $S_0^\diamond$ du symbole S_0. Soit ε la différence entre la valeur correcte et la valeur corrompue de ce symbole. On a la relation suivante : $S_0^\diamond \oplus S_0^* = \varepsilon$.

D'après le code LDPC, S_0 est présent dans plusieurs équations, soit S_1, S_2, S_3 les autres symboles d'une de ces équations. Nous avons l'équation suivante : $S_0 \oplus S_1 \oplus S_2 \oplus S_3 = 0$

Supposons que S_1et S_2 on été reçues et que les valeurs reçues sont correctes, S_3 est donc le seul symbole inconnu de cette équation. L'algorithme de décodage va calculer la valeur $S_3^\dagger$ grâce à cette équation :

$S_3^\dagger = S_1^* \oplus S_2^* \oplus S_0^\diamond = S_1^* \oplus S_2^* \oplus S_0^* \oplus \varepsilon = S_3^* \oplus \varepsilon$

La valeur $S_3^\dagger$ calculée par le décodeur est différente de la valeur correcte S_3^*du symbole S_3. Le symbole S_3 est donc à sont tour corrompu. Si le décodeur utilise le symbole S_3 pour reconstruire d'autres symboles inconnus, ces symboles seront corrompus à leur tour. Un symbole peut être présent dans 3 équations différentes, il peut donc être potentiellement utilisé pour reconstruire directement plus d'un symbole.

Il peut donc se produire un effet d'avalanche qui propagera la corruption dans un ensemble de symboles reconstruits directement ou indirectement à partir d'un seul symbole corrompu.

Approche expérimentale

Nous avons vu que la corruption d'un symbole utilisé par le décodeur peut provoquer la corruption d'autres symboles. Nous appellerons cela le phénomène de propagation de corruption. Nous voulons maintenant observer l'importance de ce phénomène sur un schéma FEC standard (LDPC-Staircase avec un fec_ratio de 2.5) avec un objet source constitué de 20000 symboles. Pour chaque nombre de symboles corrompus durant la transmission nous avons effectué 2500 tests. La position des symboles corrompus est choisit aléatoirement.

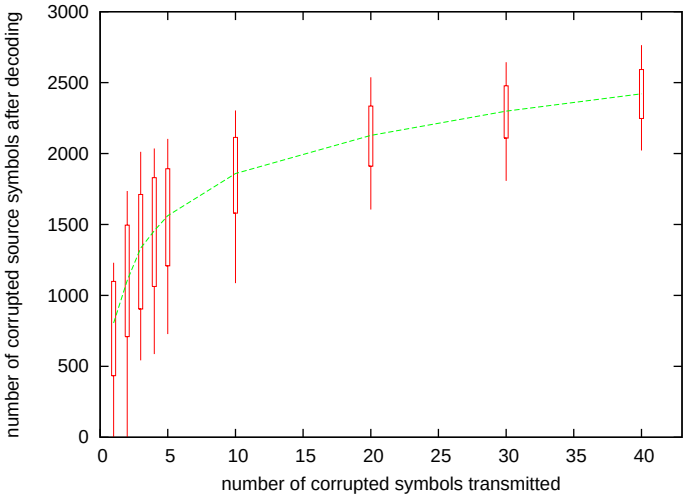


FIG. 3.1 – Nombre de symboles source corrompus après décodage (N_{cssad}) en fonction du nombre de symboles corrompus durant la transmission. moyenne/min/max/ intervalle de confiance à 90 %

Ce graphe montre l'importance du phénomène de propagation des corruptions. En effet, une simple corruption d'un symbole durant la transmission peut causer la corruption de 1230 symboles de l'objet décodé, ce qui représente plus de 5 % des données de l'objet.

On remarque que certains essais n'ont provoqué aucune corruption dans l'objet décodé. Cela provient du fait qu'un symbole n'est pas forcément utilisé par l'algorithme de décodage pour reconstruire l'objet. En effet si un symbole source est reçu mais que le symbole est déjà connu (parce qu'il a été reconstruit ou qu'il a déjà été reçu), alors ce symbole ne sera pas pris en compte dans le décodage.

On peut noter également que certains tests ne font apparaître que peu de

symboles corrompus. Cela veut dire qu'il y a des symboles qui sont utilisés pour reconstruire un petit ensemble des symboles de l'objet.

Malgré tout, si l'on s'intéresse à la moyenne , nous constatons que même la corruption d'un très petit nombre de symboles avant décodage mène à la corruption d'une partie importante de l'objet décodé. La corruption d'un symbole avant décodage provoque la corruption de 766 symboles après décodage ce qui représente 3.83% de l'objet décodé.

3.2.2 Premières conclusions

Nous pouvons tirer plusieurs conclusions de cette première expérience.

Pour l'attaquant

Nous avons vu que la corruption d'un seul symbole peut conduire à la corruption d'un grand nombre de symboles après décodage. Une corruption massive est donc facile à provoquer par un attaquant, sans pour autant avoir à corrompre tous les symboles transmis.

D'un autre côté il est difficile pour l'attaquant de provoquer une corruption limitée, sauf s'il est capable de choisir judicieusement les symboles qu'il va corrompre. J'entends par là, choisir les symboles qui ne provoquent qu'une ou très peu de corruptions sur les symboles de l'objet décodé. En effet nous avons vu que de tels symboles existent. En pratique, cela nécessite que l'attaquant :

- contrôle l'ordre dans lequel le décodeur traite les symboles reçus
- connaisse la matrice de parité LDPC utilisée.

Pour l'émetteur et le récepteur

Comme une simple corruption peu provoquer une corruption d'un grand nombre de symboles en sortie de décodeur, il est à priori envisageable pour le récepteur de détecter une attaque sans avoir à vérifier l'intégralité de l'objet décodé.

Pour favoriser cette détection il va falloir maximiser l'effet de ce phénomène de propagation de corruption. Pour cela il faut augmenter le nombre de symboles source qui va être reconstruit à partir des symboles reçus. Un moyen possible est d'utiliser un schéma de transmission qui ne transmet qu'une partie des symboles source, puisque les symboles source non transmis devront être impérativement reconstruits par le décodeur.

Nous avons vu que l'attaquant peut choisir judicieusement les symboles à corrompre pour limiter l'effet de propagation. Une parade à cette faiblesse

peut être l'introduction d'une file de symboles avant le décodeur. Les symboles seraient pris dans un ordre aléatoire par le décodeur. Ainsi l'attaquant ne pourrait pas connaître à l'avance l'ordre de décodage de l'objet, et ainsi ne pourrait pas choisir les symboles qui provoquent peu de propagations.

3.3 Proposition d'une solution : VeriFEC

3.3.1 Principes

L'idée principale est de vérifier un sous-ensemble des symboles de l'objet décodé dans le but de détecter une possible attaque. Grâce au phénomène de propagation de corruption nous savons qu'une attaque sur un symbole peut provoquer la corruption d'un grand nombre de symboles. En vérifiant seulement un sous-ensemble nous espérons détecter les attaques pour un coût inférieur à une vérification complète de l'objet par un hash global. Nous allons présenter les opérations à ajouter au protocole FEC standard pour permettre une telle vérification.

Du coté de l'émetteur

On choisit aléatoirement un sous ensemble Vde N_{verif} symboles source à partir d'une graine *verif_ seed*. Puis nous calculons le hash de ce sous-ensemble de symboles. Ce hash partiel de l'objet est appelé *partial_ hash*.

Que faut-il transmettre ?

L'émetteur doit transmettre une partie des symboles source et les symboles de parité. Il doit aussi transmettre les informations permettant au récepteur d'effectuer les vérifications : le hash partiel ainsi que la graine. Ces nouvelles informations sont appelées les données de vérification (*verif_ data)*.

- La graine va permettre au récepteur de reconstituer le sous-ensemble V après décodage de l'objet ;
- Le hash partiel va servir à vérifier l'intégrité du sous-ensemble V ;

Nous pouvons transmettre les données de vérification sur un canal sécurisé ou bien sur le même canal que les symboles. Dans le dernier cas l'attaquant pourrait modifier ou utiliser à son avantage les données de vérification. Plusieurs parades peuvent être mises en place :

- une première solution serait de transmettre également une signature de ces informations, afin d'empêcher un forgeage de ces données ;

– nous pouvons également chiffrer ces informations afin que l'attaquant ne puisse pas connaître le sous-ensemble V. Cela rendrait une attaque encore plus difficile car l'attaquant ne pourrait pas adapter son attaque en fonction des symboles corrompus.

Du coté du récepteur

Le récepteur commence par décoder les symboles reçus jusqu'au décodage complet de l'objet. Ensuite il sélectionne le sous-ensemble V grâce à la graine *verif_ seed*, puis il calcule le hash du sous-ensemble V. Finalement il compare la valeur calculée du hash partiel à la valeur reçue. Si les deux hash sont différents alors une corruption a été détectée. Sinon on ne peut pas conclure puisqu'il y a une probabilité que l'objet décodé soit corrompu sans que l'on ait détecté cette corruption.

Par la suite nous allons essayer de minimiser cette probabilité.

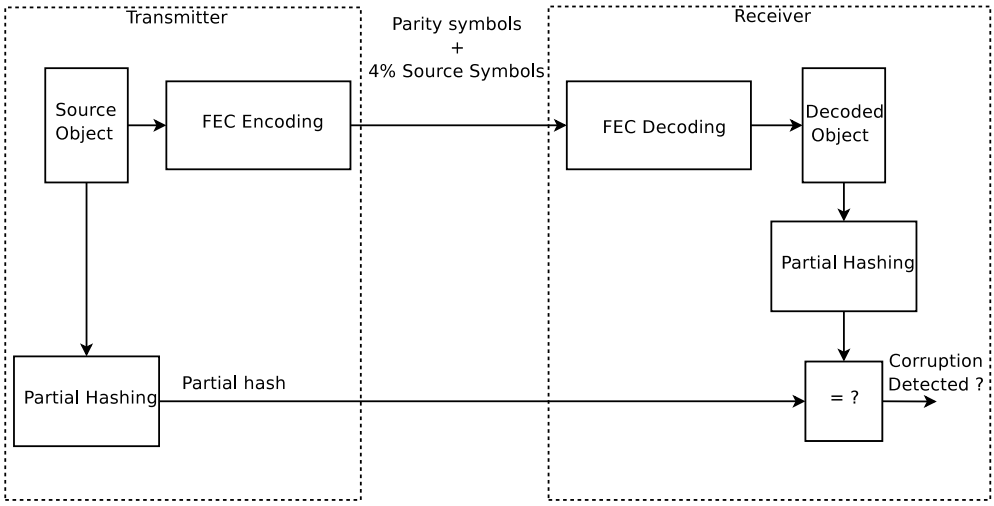


FIG. 3.2 – Schéma de principe de VeriFEC

3.3.2 Choix du modèle de transmission

Nous avons vu qu'il était intéressant de choisir un schéma de transmission qui n'envoie qu'une partie des symboles sources. Des travaux antérieurs [1] ont montré qu'une matrice LDPC-Staircase avec un FEC ratio de 2.5 et un ordre de transmission aléatoire permettent d'obtenir de bonnes performances avec un tel schéma de transmission.

3.3.3 Paramètres restant

Il reste plusieurs paramètres qui peuvent être ajustés afin de minimiser la probabilité de non détection de corruption et le coût en temps CPU :

1. Le pourcentage de symboles source transmis ;
2. Le ratio de vérification c'est à dire le pourcentage de symboles source de l'objet contenu dans le sous-ensemble V.

3.4 Expérimentations

Les expériences suivantes nous ont permis d'ajuster les paramètres restant du codec. Nous avons pris un objet de 20000 symboles et un FEC ratio de 2.5. La position des symboles corrompus était choisit aléatoirement.

Nous supposons que l'attaquant ne veut pas se faire détecter. Comme le nombre de corruptions après décodage augmente en moyenne avec le nombre de symboles corrompus avant décodage, un seul symbole choisit aléatoirement sera corrompu au cours de chaque expérience. Nous nous plaçons donc dans le cas le plus défavorable.

3.4.1 Ratio de symboles source transmis

Nous avons fait varier le nombre de symboles transmis, et nous avons observé l'évolution du nombre de corruptions provoquées sur les symboles de l'objet décodé, mais aussi sur le ratio d'inefficacité, qui doit impérativement rester bon (c'est à dire faible) sous peine de rendre le codage FEC inefficace. Nous avons effectué 1000 tests pour chaque pourcentage de symboles sources transmis.

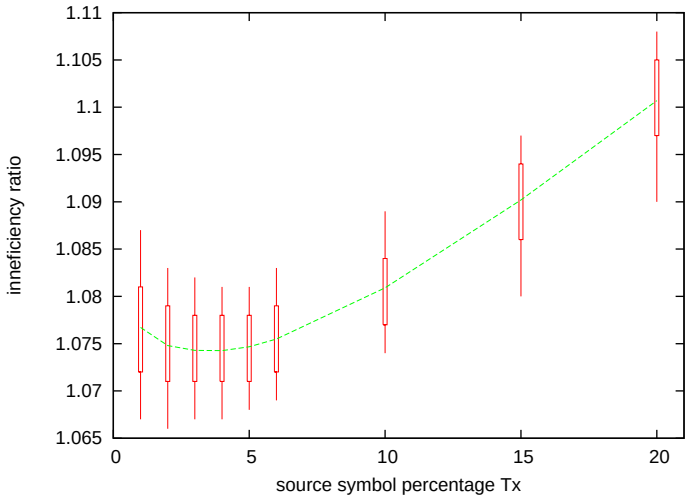


FIG. 3.3 – Ratio d'inefficacité en fonction du pourcentage de symboles source transmis moyenne/min/max/ intervalle de confiance à 90 %

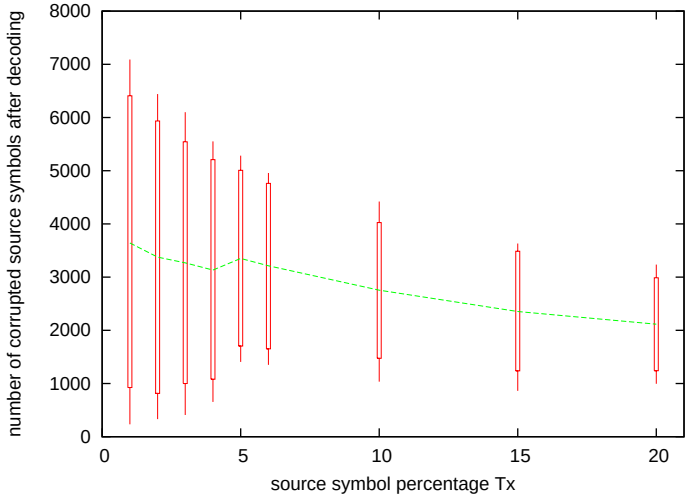


FIG. 3.4 – Nombre de symboles source corrompus après décodage en fonction du pourcentage de symboles source transmis moyenne/ intervalle de confiance à 90 % / intervalle de confiance à 95 %

Comme nous l'avions supposé le nombre de symboles corrompus dans l'objet décodé décroît quand le nombre de symboles source transmis aug-

mente. Nous observons également un minimum pour le ratio d'inefficacité lorsque le ratio de symboles source transmis est autour de 4%.

Nous choisissons donc cette valeur de 4% de symboles source, qui nous permet de minimiser le ratio d'inefficacité tout en ayant un nombre de corruptions proche du maximum obtenu avec cette expérience.

3.4.2 Nombre de vérifications

Il est clair que l'on augmente la probabilité de détection d'une attaque, en augmentant le nombre de symboles que l'on va vérifier après le décodage. Nous avons donc fait varier le nombre de symboles pris en compte pour calculer le hash partiel et nous avons observé le pourcentage de détection par notre système. Pour chaque *vérification data ratio* nous avons effectué 100 000 tests.

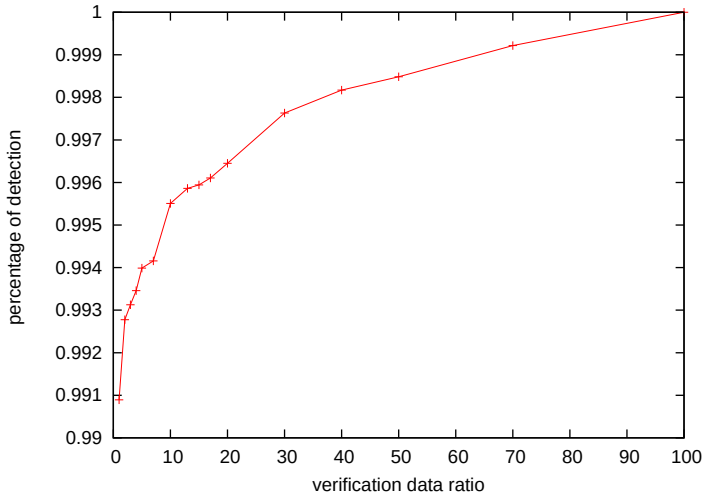


FIG. 3.5 – Probabilité moyenne de détection en fonction du pourcentage de symboles source vérifié après décodage.

Nous obtenons une courbe qui montre que le taux de détection augmente avec le nombre de vérifications. On remarque que pour atteindre un taux de non détection inférieur à 0.1% il faut vérifier plus de 70% des symboles de l'objet. Cela n'a aucun intérêt puisque cela revient presque à vérifier l'intégralité de l'objet. Cependant avec un ratio de vérification de 10% nous avons une probabilité de non-détection de 0.5%. De plus en vérifiant seulement 1% du fichier on obtient une probabilité de non détection de 1%.

Nous considérons que la vérification de 10% des symboles afin d'obtenir une probabilité de détection de corruption de 99.5% représente un bon compromis.

3.4.3 Évaluation des performances

Nous avons procédé à des mesures de performances afin de comparer les temps CPU du VeriFEC avec une solution ayant les étapes FEC et hachages séparées. Nous avons comparé les durées d'encodage et de décodage, et nous avons également mesuré le temps pour l'étape de vérification. Pour le VeriFEC il s'agit de l'étape de création du hash partiel ; pour la version avec la brique de hachage séparée de la brique FEC, il s'agit tout simplement du temps de calcul du hash sur l'objet complet. Ces mesures ont été effectuées sur un *Pentium-D bi-coeur* 64 bits, 3 Go de RAM. L'objet utilisé était constitué de 20000 symboles de 1024 octets, ce qui représente 20 Mo octets. Les test ont étés répétés 200 fois.

	VeriFEC	FEC + Hash	gain relatif
durée d'encodage	0.142 s	0.285 s	-50.21 %
durée de décodage	0.264 s	0.407 s	-35.18 %

Table 3.1: Comparaison des temps de calcul

Les gains observés au niveau du temps d'encodage et de décodage sont important. De plus on s'aperçoit que le gain provient du calcul de vérification. En effet nous obtenons une diminution de 95% au niveau du temps de vérification.

3.4.4 Impact de la taille de l'objet

Maintenant que nous avons fixé tous les paramètres du codec, nous voulons observer l'influence de la taille de l'objet source (plus particulièrement du nombre de symboles source) sur la probabilité de détection de corruption. Pour chaque taille d'objet source nous avons effectué 50 000 tests.

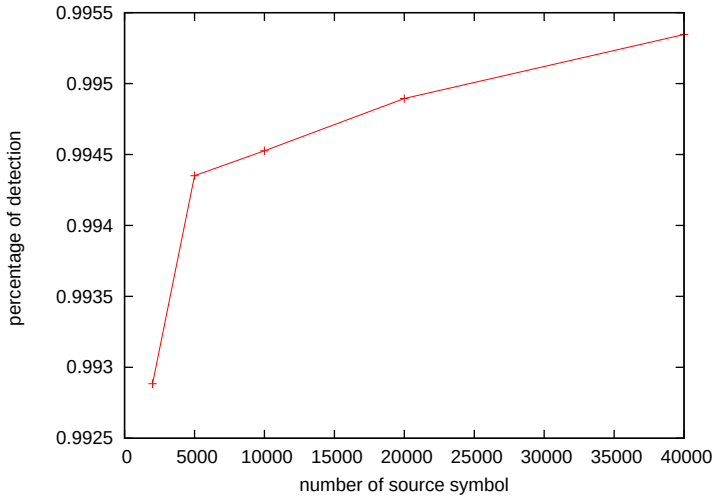


Figure 3.6: Probabilité moyenne de détection de corruption en fonction du nombre de symboles de l'objet source

Le graphe obtenu nous montre que l'on détecte d'autant mieux les corruptions que le nombre de symboles source est grand. Une première explication est que le nombre d'étapes de décodage augmente avec le nombre de symboles source. Le nombre de symboles pouvant être potentiellement corrompus lors du décodage augmente donc aussi. Cela veut aussi dire que la proportion de symboles dont la corruption provoque peu de corruptions après décodage diminue quand le nombre de symboles source augmente.

Afin d'améliorer les performances en terme de détection, il faudrait donc augmenter le nombre de symbole source. La taille de l'objet étant fixée on peut diminuer la taille des symboles afin d'augmenter leur nombre.

Cependant nous voyons que des objets de taille égale ou supérieur à 1000 symboles conduisent à de bon résultats. Les codes LDPC étant efficaces seulement avec les gros objets (taille $\geq$1000 symboles) le VeriFEC est parfaitement adapté.

3.4.5 Conclusions sur VeriFEC

Le VeriFEC permet donc de détecter les attaques consistant à corrompre un symbole durant la transmission pour un coût beaucoup plus faible qu'une vérification complète de l'objet. Cette détection n'est pas systématique et il est possible que des attaques ne soient pas détectées, c'est pourquoi on ne peut pas utiliser ce système pour certifier l'intégrité de l'objet décodé. Ce-

pendant la probabilité de ne pas détecter une telle attaque est faible (moins de 1%). On pourrait donc utiliser ce système comme un premier filtre, avant une vérification plus coûteuse de l'objet avec les méthodes traditionnelles.

Dans les travaux futurs, nous souhaitons étudier plus en détail le VeriFEC, en particulier les attaques possibles sur celui-ci ainsi que les parades à mettre en place le cas échéant.

Chapitre 4

DiFEC : Discrimination des symboles corrompus

4.1 Présentation

Nous avons vu qu'il était possible de détecter efficacement une corruption de symbole grâce au phénomène de propagation de corruption. Nous souhaitons maintenant apporter une nouvelle fonctionnalité qui permettrait de déterminer quels symboles ont été corrompus.

Le point difficile, qui constitue l'originalité de la solution, est que l'on ne cherche pas à mettre en place un système permettant de vérifier l'intégrité de chaque symbole transmis directement (par exemple au moyen d'un hash) comme cela se fait habituellement. Au contraire nous voulons procéder de façon statistique, seul un sous ensemble réduit des symboles étant vérifiés directement, l'intégrité des autres symboles étant alors déduite. Un gros intérêt est que l'on réduit largement l'overhead de transmission, puisque la majorité des symboles transmis le sont sans que l'on ait à ajouter la moindre information aux paquets transmis.

Le système que nous avons conçu procède en plusieurs étapes, et exploite les phénomènes mis en évidence dans le système VeriFEC.

Nous avons vu que lorsqu'un symbole corrompu est utilisé par le décodeur, tous les symboles reconstruits à partir de celui-ci héritent de cette corruption. La vérification d'intégrité sur un sous-ensemble de symboles de l'objet décodé va nous permettre de construire une liste des symboles source décodés corrompus : la *Corruption List*. A partir de cette liste nous allons remonter à la source des corruptions. Mais pour cela il nous faut garder en mémoire le schéma de décodage de l'objet afin de connaître les dépendances

entre les différents symboles. Pour cela nous utilisons une représentation sous forme de graphe. Une fois les symboles potentiellement corrompus localisés, il suffit d'effectuer un nouveau décodage en ayant écarté ces symboles.

Comme le système n'est pas parfait, il est possible que certains symboles corrompus aient échappé au crible, et inversement que des symboles corrects aient été marqués comme potentiellement corrompus. Par conséquent, le procédé est répété jusqu'à ce que l'on obtienne un objet décodé non corrompu, dont l'intégrité peut être prouvée au moyen d'un calcul de hash global.

4.2 Le graphe de décodage

Le graphe de décodage est une représentation d'une session de l'algorithme de décodage. Il représente les dépendances entres les différents symboles.

Le graphe de décodage est un graphe orienté, dans lequel les noeuds représentent les symboles. Un arc allant du noeud X au noeud Y indique que le symbole Y a été reconstruit à partir du symbole X.

On notera que les noeuds ne possédant pas d'arcs entrant représentent des symboles reçus. Les symboles reconstruits sont représentés par des noeuds possédant des arcs entrant.

On remarque également que les noeuds représentant des symboles reconstruits possèdent 3 arcs entrant. En effet dans notre code les équations de dépendance font intervenir 4 symboles (sauf pour la première ligne qui ne fait intervenir que 3 symboles), le symbole inconnu est donc reconstruit à partir de 3 autres symboles connus.

Cette représentation sous forme de graphe est pratique pour se donner une idée du déroulement du décodage. On peut facilement repérer les symboles qui ont été utilisés pour reconstruire un grand nombre de symboles (ceux qui possèdent un grand nombre de fils), et également les symboles qui n'ont pas été utilisés pour reconstruire d'autres symboles.

Cette représentation pourra être utilisé pour analyser plus finement l'algorithme de décodage et pour améliorer ce dernier. Ici il va servir de support à différents algorithmes pour la localisation de symboles corrompus.

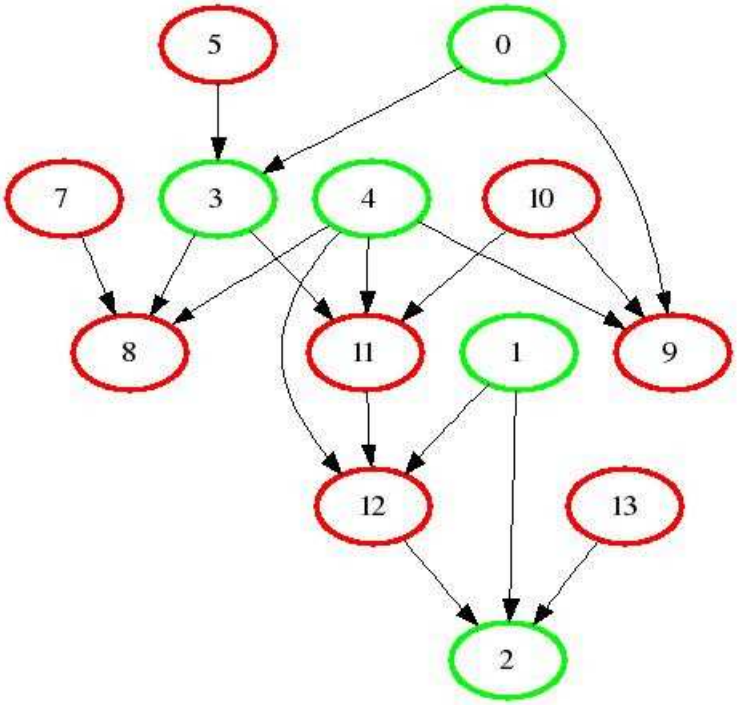


FIG. 4.1 – Graphe de décodage

4.3 Localisation des corruptions

4.3.1 Principes

Nous avons choisi une représentation sous forme de graphe afin d'utiliser des algorithmes de parcours de graphes qui vont localiser les symboles ayant provoqué des corruptions en chaîne. Le principe est de donner une note à chaque noeud en fonction des corruptions détectées dans l'ensemble de ses descendants. Plus la note d'un noeud est élevée, plus le symbole correspondant à de chances d'être corrompu.

Notre algorithme de recherche prend en entrée un graphe de décodage ainsi qu'une liste de symboles qui ont été détectés comme corrompus (*Corruption List*) après le décodage (grâce à une vérification d'intégrité symbole par symbole). Il renvoie une liste de N_{BL} symboles suspects qu'il faut considérer comme corrompus (*Black List*).

Pour créer cette *Black List*, l'algorithme parcourt le graphe et *noter* les noeuds en fonction de leur position par rapport aux noeuds de la *Corruption List*. Finalement il renvoie la *Black List* qui est composée des N_{BL}noeuds possédant les notes les plus élevées. Les noeuds mis dans *Corruption List* sont des noeuds correspondant à des symboles reçus.

Idéalement la *Black List* ne doit contenir que des symboles corrompus, et donc aucun symbole correct. Cela semble très difficile à réaliser vu l'approche statistique utilisée. Nous acceptons donc des erreurs : la *Black List* doit contenir la totalité des symboles corrompus, et le moins de symboles corrects possible.

La valeur de N_{BL}apparaît comme un paramètre important de notre système. En effet plus il sera grand plus le nombre de symboles considérés comme corrompus sera important. La probabilité de *Black-lister* les symboles effectivement corrompus augmente, de même que celle de *Black-lister* des symboles corrects.

4.3.2 L'attribution des notes

L'algorithme qui attribue les notes aux différents noeuds de l'arbre est la partie la plus importante du système. En effet c'est lui qui va permettre de repérer les symboles corrompus. Différents algorithmes ont été essayés et nous avons retenu deux algorithmes principaux. Il s'agit d'algorithmes parcourant le graphe. Ces graphes ayant une taille importante (nombre de noeud $\simeq$ nombre de symboles), nous avons été particulièrement attentif au coût de ces algorithmes.

Parcours vers le bas

Cet algorithme parcourt récursivement le graphe. Pour chaque noeud il calcule sa note en fonction de la note de tous ses fils. Si la note de certains fils n'est pas connue, alors il la calcule.

Cet algorithme considère le fait que *plus les fils d'un noeud sont suspects (note élevée), plus ce noeud est lui même suspect.*

La note d'un noeud de la *Corruption List* est fixée à une grande valeur.

Reste à définir la fonction qui attribue la note d'un noeud en fonction de la note de ses fils. Différentes fonctions ont étés essayées :

- la moyenne des notes des fils ;
- le maximum de la note des fils ; et
- la médiane de la note des fils.

L'utilisation de la moyenne semble être la meilleure solution. Cependant cette méthode pose un problème lié à la structure du graphe. En effet, ce type d'algorithmes récursifs est adapté à des arbres. Notre graphe n'est pas un arbre, c'est une sorte d'arbre dans lequel des branches peuvent se rejoindre.

On peut voir un exemple de branches qui se rejoignent sur la figure du graphe de décodage : du noeud 1 partent deux branches, une vers le noeud 12 et une autre vers le noeud 2. Mais le noeud 12 rejoint le noeud 2.

Ce type de schéma dans le graphe nous oblige à faire des opérations supplémentaires pour nous assurer que nous ne prenons pas en compte plusieurs fois un noeud. En effet la note du noeud 1 dépend de la note du noeud 12 et de celle du noeud 2. Mais la note du noeud 12 dépend elle aussi de la note du noeud 2. Avec un parcours naïf nous prendrions en compte deux fois la note du noeud 2 pour calculer la note du noeud 1.

Parcours vers le haut

Une manière d'éviter le problème des branches qui se rejoignent est d'effectuer un parcours à partir du bas. Pour chaque symbole de la *Corruption List* nous remontons dans l'arbre, en incrémentant la note de chaque noeud rencontré.

Cette méthode considère le fait que *chaque symbole corrompu de la* Corruption List *possède au moins un symbole reçu corrompu dans ses ancêtres.*

Ce type de parcours est très simple à réaliser, puisqu'il suffit de remonter dans le graphe de décodage. Finalement son coût est beaucoup plus faible que celui du parcours vers le bas.

4.3.3 Premiers résultats

Nous avons observé des résultats encourageant avec l'algorithme de parcours vers le haut. Dans le cas où un seul symbole reçu est corrompu, nous sommes capables de le repérer avec une forte probabilité, tout en conservant une *Black List* de petite taille (ce qui signifie que nous écartons peu de symboles corrects).

La corruption de plusieurs symboles reçus rend la tâche de recherche plus difficile. En effet, les effets de chaque corruption originale se chevauchent, et il devient plus difficile de localiser les symboles à la sources des corruptions détectées. Pour le moment l'algorithme écarte trop de symboles corrects pour être utilisable. Cependant nous pensons pouvoir l'améliorer en regroupant les informations obtenues après plusieurs sessions de décodage effectuées sur le même ensemble de symboles reçus, mais dans un ordre différent à chaque fois.

4.4 Conclusion sur le DiFEC

Le DiFEC est encore à l'état d'ébauche et peu de tests ont été effectués. Cependant cette première étude nous a permis d'observer des résultats encourageants. Nous avons manqué de temps pour mettre en place des batteries de tests rigoureux. Les premières expérimentations ont cependant montré qu'il était possible de mettre en place un algorithme efficace permettant la localisation des symboles corrompus.

Ce système sera amélioré en suivant les pistes mises en évidence par cette première étude.

Conclusion

Une première étude de ces codes LDPC m'a permis d'imaginer et de proposer un type de codec hybride permettant d'utiliser le code LDPC pour faire de la détection de corruption durant l'opération de décodage. Un prototype VeriFEC a été implémenté et il permet l'obtention de résultats intéressants. Des travaux doivent être poursuivis dans plusieurs directions :

- l'amélioration de la sécurité ; et
- la conception d'un système permettant de distinguer les symboles corrompus des symboles corrects

Un nouveau système permettant de distinguer les symboles corrompus des autres symboles est également en cours d'élaboration.

Durant ces quelques mois j'ai pû améliorer mes compétences techniques et mes connaissances sur les codes LDPC et les techniques de diffusion de contenus à grande échelle. J'ai également beaucoup appris sur la façon de faire de la recherche : la façon de rédiger un papier, les protocoles expérimentaux, la manière de présenter ses résultats. Grâce à ce stage j'ai pû découvrir le monde de la recherche et cette expérience renforce mon envie de poursuivre avec une thèse de doctorat.

Bibliographie

[1] 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", INRIA Research Report Number 5225, June 2004.

[2] C. Neumann, V. Roca, A. Francillon, D. Furodet "*Impacts of Packet Scheduling and Packet Loss Distribution on FEC Performances : Observations and Recommendations*", INRIA Research Report RR-5578, May 2005.

[3] V. Roca, C. Neumann, D. Furodet "*Low Density Parity Check (LDPC) Forward Error Correction*" ,IETF RMT Working Group, draft-ietf-rmt-bb-fec-ldpc-03.txt (travail en cours), Juillet 2006.

[4] C. Neumann, V. Roca, A. Francillon, D. Furodet "*Impacts of Packet Scheduling and Packet Loss Distribution on FEC Performances : Observations and Recommendations*" , ACM CoNEXT'05 Conference, Toulouse, France, October 2005.

[5] Amin Shokrollahi "*LDPC Codes : An Introduction*", notes de cours, April 2003,

[6] *On-the-Fly Verification of Erasure-Encoded File Transfers* (Extended Abstract) Max Krohn and Michael J. Freedman 1st IRIS Student Workshop on Peer-to-Peer Systems (ISW '03) Cambridge, MA, August 2003. http ://pdos.csail.mit.edu/~max/docs/kf03.pdf

[7] George I. Davida, Jeremy A. Hansen "*A preliminary exploration of Striped Hashing : a probabilistic scheme to speed up existing hash algorithms* " , Second International Conference on e-Business and Telecommunication Networks, October, 2005.

[8] R.G. Gallager, "*Low-density parity-check codes*" IRE Trans. Inform. Theory, vol. 8, pp. 21-28, Jan. 1962.

[9] D.J.C. MacKay and R.M. Neal, "*Near Shannon limit performance of low density parity check codes*" IEE Electroncs Letters, vol. 32, no. 18, pp. 1645-1655, 29th Aug. 1996.