Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par **l'Institut Supérieur de l'Aéronautique et de l'Espace**
Spécialité : STIC Réseaux, télécoms, systèmes et architectures

---

Présentée et soutenue par **Alexandre SORO**
Le 3 décembre 2010

Mécanismes de fiabilisation pro-actifs

JURY

M. Kavé Salamatian, président, rapporteur
M. Daniel Augot, rapporteur
M. Jérôme Lacan, directeur de thèse
M. Vincent Roca
M. Valentin Savin

École doctorale : **Mathématiques, informatique et télécommunications de Toulouse**

Unité de recherche : **Équipe d'accueil ISAE-ONERA MOIS**

Directeur de thèse : **M. Jérôme Lacan**

Mécanismes de fiabilisation pro-actifs

Depuis l'avènement de la théorie de l'information, la question de la fiabilisation des transmissions est allée de pair avec l'évolution des technologies et des applications.
La fiabilisation des transmissions couvre l'ensemble des techniques permettant de lutter contre les erreurs et les pertes d'un canal de transmission. Parmi les mécanismes existants, la fiabilisation pro-active consiste à mettre en place une redondance au niveau d'un émetteur, celle-ci permettant de compenser à priori les pertes subies par le canal. La fiabilisation pro-active trouve son sens lorsque l'émetteur ne peut avoir d'information sur l'état actuel du canal de transmission, ou lorsque elle est inutile du fait de contraintes temporelles, mais aussi lorsqu'un émetteur doit s'adapter aux contraintes de plusieurs récepteurs. Initialement, les mécanismes de fiabilisation pro-actifs utilisant cette redondance sont connus sous le nom de codes correcteurs. La problématique associée à ces codes est alors une problématique d'optimisation : il s'agit de créer des codes flexibles permettant une génération rapide de la redondance par l'émetteur et une récupération des données initiales par le récepteur à faible coût, tout en conservant la meilleure capacité de correction possible.
Par extension, la question de la fiabilisation concerne également l'étude de l'impact de mécanismes de suppression de redondance sur la fiabilité d'un système. Dans des réseaux plus spécifiques, où la question de l'utilisation de la bande passante est cruciale, des mécanismes de compression protocolaire peuvent être mis en place afin de diminuer la proportion de trafic engendrée par les en-têtes protocolaires.

Mots-clés : Fiabilisation, ROHC, Canal à effacements, Matrice bande, Codes de Reed-Muller, Codes de Reed-Solomon, FFT

Pro-active mechanisms for reliability

Since the works of Shannon on information theory, the question of transmission reliability is a crucial point.
Transmission reliability covers the techniques that allow to fight errors and losses of a transmission channel. Among them, pro-active mechanisms consist in constructing redundancy for a sender, in order to compensate the estimated losses of the transmission channel. This technique is especially valuable when the feedback of the receiver is not available or unusable, or when there are several receivers. This family of solutions is known as error correcting codes. For these codes, the main point is to determine codes that are easily encodable and decodable with the best correction capacity.
The reliability issue can be extended to the study of the impact of suppressing redundancy on the reliability of a system. In specific networks, where the bandwidth is costly, one may use compression protocols to reduce the protocols headers size, which play the role of a natural redundancy.

Keywords : Reliability, Header Compression, Erasure Channel, Banded Matrices, Reed-Muller Codes, Reed-Solomon Codes, Fast Fourier Transform (FFT)

Remerciements

Mes premiers remerciements vont à mon directeur de thèse, Jérôme Lacan, pour son merveilleux encadrement et sa disponibilité. Ses conseils, son soutien et sa grande connaissance du domaine m'ont permis de ne pas perdre le fil de ma thèse, lorsque les difficultés se présentaient. Je le remercie également pour la confiance qu'il m'a accordée lorsqu'il m'a proposé cette thèse.

Je me dois également de remercier Daniel Augot et Kavé Salamatian d'avoir accepté de rapporter ma thèse, et par là même, l'ensemble des membres du jury. Je souhaite également remercier l'ANR et l'ensemble des membres du projet CAPRI-FEC pour les discussions que nous avons pu avoir lors de nos réunions. Je remercie notamment Mathieu Cunche et Jonathan Detchart pour leurs connaissances poussées dans le domaine de la chimie.

Je tiens également à remercier Patrick Sénac et tout le département DMIA de l'ISAE de m'avoir accueilli dans un environnement agréable et de grande qualité. Je remercie particulièrement Yves Caumel pour sa grande accessiblité et son partage enthousiaste de ses vastes connaissances mathématiques. De la même manière, je remercie Fabrice Francès pour ses précieux conseils qui m'ont aidé à appréhender les différents codes que j'ai implémentés. Je remercie Emmanuel Lochin, pour avoir partagé avec moi ses connaissances du réseau, mais aussi pour son humour décapant et ses nombreuses anectodes. Je remercie enfin Tanguy Pérennou pour ses théorèmes fondateurs et Pierre de Saqui-Sannes pour m'avoir notamment supporté une année en tant que voisin de bureau. Je n'oublie pas également de remercier Bernard Jarlan et René Espourteau pour leur bonne humeur et leur grande disponibilité.

Je remercie également mes petits camarades thésards avec qui j'ai pu partager ces trois années dans la bonne humeur : Pierre-Ugo, Guillaume, Rémi, Anh Dung, ainsi que les anciens : Manu, Hervé, Dino et Lei. Je remercie notamment ceux qui ont du me supporter dans leur bureau : Benjamin, GuoDong, Ali, Huyen Chi et bien sûr Hugo qui, je pense, saura reprendre avec brio, le flambeau que je lui ai légué.

Je tiens particulièrement à remercier Thomas qui a également partagé mon bureau pendant deux années, et je lui souhaite, bien entendu, le meilleur pour la fin de sa thèse et pour la suite de sa carrière.

Je remercie également Emmanuel Chaput pour m'avoir fait confiance avec Jérôme lors de mon passage au laboratoire TéSA. Je tiens également à remercier chaleureusement mon professeur de mathématiques du lycée, Marc Meyraud, pour m'avoir donné le goût des mathématiques, et sans qui je ne serais peut-être pas en train de rédiger ce manuscrit.

Je remercie mes amis qui ont toujours été là : Nicolas, Emilie, Nicolas (Choco). J'ai également une pensée pour ceux que j'ai un peu perdu de vue durant cette dernière

année de thèse : Julien, Paul, Almed, Mazo, Tomasi, Gilles, Najwa et Adia.

Je remercie mes parents qui ont toujours été là pour moi, qui ont tout fait pour mon bonheur et qui m'ont toujours soutenu dans mes choix. Je les remercie également d'avoir toujours cru en moi, probablement plus que moi-même, et d'avoir été là dans les moments difficiles. Je ne les remercierais jamais assez pour tout.

C'est à eux que je dédie ce manuscrit.

Décembre 2010,
Alexandre SORO

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Les thèses les plus fausses sont souvent les plus belles.

Pierre DANINOS.

Sommaire

1.1 Présentation de la problématique et des objectifs

Depuis les années 1950 et l'avènement de la théorie de l'information [1], la question de la fiabilisation des transmissions est allée de pair avec l'évolution des technologies et des applications.

La fiabilisation des transmissions couvre l'ensemble des techniques permettant de lutter contre les erreurs et les pertes d'un canal de transmission. Ces canaux de transmission peuvent être d'une très grande variété : filaires ou sans-fil, point-à-point ou avec relais, unidirectionnels, bidirectionnels ou même en multi-diffusion. Ce vaste environnement appelle en conséquence un large panel de méthodes de fiabilisation. Parmi ces méthodes, la fiabilisation pro-active consiste à mettre en place des mécanismes de redondance au niveau d'un émetteur, celle-ci permettant de compenser à priori les pertes subies par le canal. La fiabilisation pro-active trouve son sens lorsque l'émetteur ne peut avoir d'information sur l'état actuel du canal de transmission parce que la transmission est unidirectionnelle ou sans voie de retour possible, mais aussi lorsqu'un émetteur doit s'adapter aux contraintes de plusieurs récepteurs dans le cadre de scénarios de multi-diffusion, ou lorsque elle est inutile du fait de contraintes temporelles. Un moyen de se prémunir des pertes engendrées sur un canal de transmission est donc de rajouter une information complémentaire en se basant sur une connaissance à priori du canal et de l'application visée.

Initialement, les mécanismes de fiabilisation pro-actifs désignent donc l'ensemble des méthodes permettant de créer de l'information complémentaire, *i.e.* de la redondance, transmise conjointement à l'information initiale. Cette famille est généralement connue sous le nom de codes correcteurs. La problématique associée à ces codes est alors une problématique d'optimisation : il s'agit de créer des codes flexibles permettant une génération rapide de la redondance par l'émetteur et une récupération des données initiales par le récepteur à faible coût, tout en conservant la meilleure capacité

de correction possible. La capacité de correction d'un code est la capacité d'un code à être décodable avec un minimum de données reçues parmi l'information initiale et la redondance. Ceci permet une première distinction entre deux grandes familles de codes correcteurs : les codes dits à Maximum Distance Séparable (MDS) qui permettent de retrouver les données initiales à partir de n'importe quelle information reçue de taille égale à la taille initiale, et les codes non-MDS pour lesquels le décodage nécessite généralement de l'information supplémentaire. Si les premiers codes sont optimaux et semblent répondre à la problématique, leur inconvénient est d'être d'une complexité élevée, et en pratique, ces codes sont souvent limités à une protection de quelques centaines d'éléments. La seconde famille permet quant à elle une plus grande flexibilité, et les complexités associées peuvent être bien plus faibles que celle des codes MDS. Toute la problématique de ces codes est donc de se rapprocher le plus possible de la capacité de correction des codes optimaux tout en gardant une complexité mathématique faible. Selon le type d'application visé, l'intérêt pour l'une ou l'autre des familles de codes sera par ailleurs différent.

La gestion de la quantité de redondance et de la fiabilité se pose dans plusieurs domaines des transmissions réseau. Un domaine particulièrement intéressant est celui de la compression d'entête où les mécanismes déployées doivent permettre d'atteindre la meilleur compromis entre le niveau de compression et celui de la fiabilité. Ce problème se pose, par exemple, dans le contexte des réseaux satellitaires où la question de l'utilisation de la bande passante utilisée et du coût engendré est cruciale. Ainsi, des mécanismes de compression protocolaire peuvent être mis en place afin de diminuer la proportion de trafic engendrée par les en-têtes protocolaires. Un en-tête TCP/IPv4 a une taille nominale de 40 octets, et la compression d'en-têtes permet une réduction de l'en-tête moyen jusqu'à 2-3 octets. On voit donc ici tout l'intérêt du déploiement de ce type de mécanismes, notamment pour les petits paquets. En contrepartie, cette compression s'accompagne d'une baisse de la fiabilité de la connexion correspondante. En effet, comme tous les champs protocolaires ne sont pas répétés pour chaque paquet, lorsque une perte surgit, celle-ci peut avoir des conséquences sur la récupération des en-têtes suivants, si ceux-ci dépendent de l'en-tête perdu. Dans ce contexte, la fiabilisation de la transmission consiste en l'étude de l'impact de cette compression au niveau protocolaire sur la transmission des données, sa modélisation et sa paramétrisation. Un point important concerne également la comparaison entre mécanismes avec et sans voie de retour et l'impact du temps de latence sur la comparaison entre chaque mécanisme.

1.2 Organisation du document

Dans le Chapitre 2, nous présenterons les principaux concepts de la théorie de l'information, des codes à effacements, et de la compression protocolaire. Nous étudierons également les principales contributions à ces problématiques de fiabilisation.

Dans le Chapitre 3, nous présenterons les mécanismes de compression d'en-têtes et leur fonctionnement, et nous proposerons un modèle mathématique permettant

une étude et une paramétrisation des mécanismes de compression.

Dans le Chapitre 4, nous présenterons un code correcteur qui a la particularité d'être décodable de manière hybride, grâce à une matrice génératrice comprenant une partie contrainte dans une bande, mais aussi une matrice de parité creuse, permettant un décodage itératif.

Dans le Chapitre 5, nous présenterons un code correcteur sur les canaux à effacements, basé sur les codes de Reed-Muller, d'une complexité minime. Nous présenterons plusieurs mécanismes visant à améliorer les performances intrinsèques du code. La simplicité de ce code lui permet d'être un candidat de choix pour des codes de petite taille et tout particulièrement à fort taux de codage.

Dans le chapitre 6, nous présenterons un code correcteur à Maximum Distance Séparable (MDS), basé sur la transformée de Fourier rapide, permettant l'utilisation de codes optimaux jusqu'à des tailles de blocs de plusieurs milliers d'éléments.

Le dernier chapitre permettra de présenter l'ensemble des conclusions de ces travaux, ainsi que les perspectives futures.

Chapitre 2

Etat de l'art et principales notions

What we do in life echoes in eternity.

Maximus Decimus Meridius.

Sommaire

2.1 La théorie de l'information

Si les problématiques étudiées par les différents chapitres de ce manuscrit peuvent être variées, elles ont toutes pour origine la théorie de l'information. Ainsi, il semble nécessaire dans un premier temps de présenter les principales notions de cette théorie.

Il est communément admis que la théorie de l'information est née en 1948 suite aux travaux fondateurs de Claude Shannon [1]. Dans article, Shannon démontre que pour n'importe quel type de canal de transmission, il était possible de transmettre sur ce canal avec un taux d'erreur arbitrairement proche de zéro si tant est que le taux de codage utilisé était inférieur à un seuil appelé capacité du canal.

Malheureusement, si leur existence a été démontrée, la construction de codes correcteurs performants voire optimaux à la fois en termes de capacité de correction d'erreurs et de complexité mathématique reste un problème ouvert.

Tout d'abord, toute transmission se base sur un canal de transmission. Ce canal peut être modélisé par le triplet : un ensemble d'entrées à valeurs dans un alphabet A,

un ensemble de sorties à valeurs dans un second alphabet B et une loi de transition définissant la probabilité d'obtenir une sortie $Y \in B$, sachant que la valeur d'entrée est $X \in A$. Cette probabilité peut se noter $P(Y|X)$. La connaissance du canal nous donne ainsi la loi de transition pour chaque couple (X, Y).

Soit une variable aléatoire X à valeurs dans $(x_0, x_1, ..., x_n)$ un ensemble fini de valeurs. On définit l'entropie de X, notée $H(X)$, qui mesure l'incertitude sur la variable X par la formulation suivante :

$$H(X) = -\sum_{i=0}^{n} P(X = x_i) \log_2 P(X = x_i)$$

L'entropie est une fonction positive qui s'annule si et seulement si X est déterminée ou en d'autres termes si $\exists i \in \{0, 1, ..., n\}/P(X = x_i) = 1$. Par ailleurs, l'entropie peut être vue comme le nombre minimal de bits nécessaires au codage de X.

De manière générale, l'objectif du récepteur/décodeur est de déduire la valeur d'entrée X qui a été transmise en fonction de la valeur Y de l'alphabet de sortie qu'il a reçue. Pour cela, on définit alors l'information mutuelle des variables X et Y, notée $I(X, Y)$ qui représente l'information que l'on peut déduire sur X à partir de Y. L'information mutuelle s'exprime de la manière suivante :

$$I(X, Y) = H(X) - H(X|Y)$$

Lorsque le canal de transmission est sans erreur, la connaissance de Y permet d'obtenir X de manière déterministe : $H(X|Y) = 0$. L'information mutuelle est alors maximale. Dans le pire cas, le canal est tellement mauvais que la connaissance de Y n'apporte aucune information sur X et dans ce cas, $H(X|Y) = H(X)$, l'information mutuelle est nulle rendant toute transmission impossible.

La capacité C d'un canal de transmission se définit de la manière suivante :

$$C = \max_X(I(X, Y))$$

Dans le cadre d'un canal parfait, la capacité du canal est optimale $C = \max_X(H(X))$ alors qu'elle est nulle lorsque l'entrée et la sortie sont totalement décorellées.

2.2 Les codes à effacements

De manière générale, les codes correcteurs d'erreurs désignent l'ensemble des mécanismes mathématiques permettant la correction de différents types d'erreurs pouvant affecter un ensemble de données. Ces codes sont ainsi capables de détecter et corriger des erreurs de transmission, mais également des pertes d'information. Dans ce dernier cas, la famille de codes correspondante est connue sous la dénomination de codes correcteurs pour le canal à effacements, ou plus singulièrement codes à effacements.

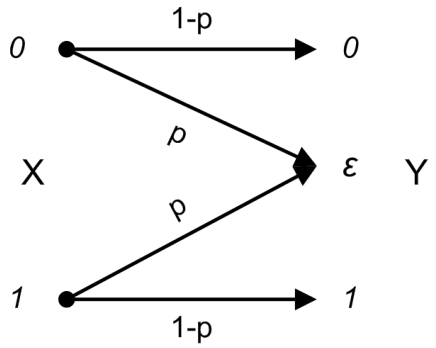

Figure 2.1 – Représentation du canal binaire à effacements

Dans cette partie, nous nous efforcerons donc de présenter les concepts relatifs aux codes correcteurs et plus particulièrement aux codes à effacements. Nous effectuerons également une présentation des principales familles de codes à effacements , qui serviront de référence pour la suite de ce manuscrit.

2.2.1 Capacité du canal à effacements

Suite aux résultats fondateurs concernant la théorie de l'information, présentés dans la partie 2.1, Peter Elias, en 1955, présente un modèle de canal binaire à effacements [2], connu sous le nom de *Binary Erasure Channel* (BEC). Sur ce canal, les erreurs modélisées correspondent à une perte d'information. Il n'y a pas d'altération de l'information reçue : si celle-ci est reçue, elle est considérée comme correcte.

Nous considérons un canal à effacements de paramètre p, c'est à dire que la probabilité qu'un bit soit effacé est égale à p. Ce canal possède donc un alphabet d'entrée $A = \{0, 1\}$ et un alphabet de sortie $B = \{0, \epsilon, 1\}$. L'état de sortie ϵ correspond à celui d'un bit effacé/perdu. En accord avec la Figure 2.1, la loi de transition du canal à effacements binaire est :

$$P(Y = 0|X = 0) = 1 - p$$
$$P(Y = \epsilon|X = 0) = p$$
$$P(Y = \epsilon|X = 1) = p$$
$$P(Y = 1|X = 1) = 1 - p$$

Dans le cadre du canal binaire uniforme nous avons $H(X) = H(Y) = 1$. La capacité du canal BEC s'exprime vaut donc :

$$C = \max_X((1 - p)H(X)) = 1 - p$$

En d'autres termes, sur le canal à effacements dont la probabilité d'effacements est p, la capacité correspondante est $1 - p$.

2.2.2 Le canal à effacements de paquets

Dans la majorité des applications, les pertes considérées ne concernent pas des bits isolés mais plutôt des groupements de bits, pouvant aller jusqu'à plusieurs millions de bits. Il est alors judicieux d'introduire un modèle plus général que le canal à effacements : le canal à effacements de paquets.

Dans ce cadre, les groupements de bits se font dans ce qu'on appelle des paquets. Les pertes imputables à ce canal correspondent alors à des pertes de paquets. Par définition, lorsque un paquet est reçu, il n'est pas altéré, ce qui est garanti en pratique par un mécanisme de vérification d'intégrité : lorsque une erreur est détectée dans un paquet, celui-ci est alors effacé.

Dès lors, par la suite, les opérations sur le canal à effacements de paquets peuvent être vues comme les opérations sur la canal à effacements binaire sur les bits de même position dans leur paquet correspondant, celles-ci étant effectuées alors en parallèle. Nous verrons par la suite que ceci à une incidence très forte sur les résultats pratiques de vitesse des codes correcteurs étudiés.

Les contextes d'utilisation du canal à effacements de paquets sont très larges, on pourra alors citer parmi les exemples d'utilisation :

- L'ensemble des réseaux de communication modernes. En effet, la plupart des protocoles modernes (TCP,UDP,...) considèrent que, dès lors qu'une erreur binaire est détectée, le paquet protocolaire associée est effacé et n'est pas communiqué à la pile protocolaire de niveau supérieur. Ces protocoles fonctionnent donc comme des canaux à effacements de paquets naturels.
- Les mécanismes de stockage et de partage distribués. Le principe de ces applications est de considérer l'information comme répartie sur un nombre conséquent d'unités de stockage, qui peuvent à tout moment, être indisponibles. Là aussi, la filiation avec les canaux à effacements de paquets est naturelle. L'information peut être stockée localement, comme par exemple, pour des mécanismes de type RAID, ou globalement, comme dans les systèmes P2P.

2.2.3 Quelques définitions pour les canaux à effacements de paquets

Définition 2.1. *On appelle corps fini (ou de Galois) un ensemble fini d'éléments sur lequel sont définies l'addition, la soustraction, la multiplication et la division. Pour chaque entier premier p et entier m, il existe un unique corps fini (à un isomorphisme près) contenant $q = p^m$ éléments, et noté $GF(q)$ ou plus communément $\mathbb{F}_q$.*

Trois cas particuliers vont nous intéresser pour la suite des travaux.

Lorsque $m = 1$, le corps fini est équivalent à celui des entiers dont les opérations sont définies *modulo* p. Nous étudierons plus précisément ce type de corps dans le chapitre 6.

Lorsque $p = 2$, les éléments du corps sont donc des combinaisons de m bits. En conséquence, l'addition et la soustraction dans ce corps sont équivalentes au XOR

bit à bit, qui est très efficace dans une implémentation pratique sur processeur. En contrepartie, la multiplication et la division sont moins directes que dans le cas précédent.

Enfin, lorsque à la fois $p = 2$ et $m = 1$, le corps fini est réduit au corps des binaires $\mathbb{F}_2 = \{0, 1\}$.

Définition 2.2. *On appelle élément primitif ou générateur d'un corps fini $\mathbb{F}_q$, un élément α tel que $\mathbb{F}_q = \{1, \alpha, \alpha^2, ..., \alpha^{n-1}\}$. En d'autres termes, il est possible de décrire chaque élément du corps fini comme une puissance de α. Tout corps fini contient au moins un élément primitif.*

Les travaux présentés par la suite concernent les codes correcteurs protégeant une quantité finie de données à la fois. Ces codes sont donc appelés codes bloc. Les codes bloc linéaires bénéficient d'une structure d'espace vectoriel qui en font des codes particulièrement pratiques à utiliser.

Définition 2.3. *On appelle code bloc linéaire $\mathcal{C}$ de longueur n et de dimension k un sous-espace vectoriel de dimension k de l'espace vectoriel $\mathbb{F}_q^n$. Les éléments de $\mathcal{C}$ sont appelés mots du code.*

Définition 2.4. *On appelle symbole les éléments du corps fini sur lequel travaille le code. Ces symboles composent les mots du code. Ainsi dans le cadre du code à effacements de paquets, chaque paquet est assimilé à un symbole ou découpé en groupes de symboles, selon l'application. Les effacements de paquets, correspondent alors à l'effacement de l'ensemble de ses symboles.*

A partir de sa longueur et de sa dimension, on peut définir le rendement (ou ratio/taux de codage) d'un code. Ce rendement permet d'obtenir la quantité de redondance ajoutée par le code.

Définition 2.5. *On appelle rendement d'un code de dimension k et de longueur n le rapport $R = k/n$.*

Définition 2.6. *On appelle matrice génératrice du code, la matrice de l'application linéaire de $\mathbb{F}_q^k$ dans $\mathbb{F}_q^n$ dont l'image est le code. La multiplication d'un élément source par cette matrice est connue sous le nom d'encodage. Les lignes de la matrice génératrice $G \in \mathcal{M}(\mathbb{F}_q)_{k,n}$ d'un code linéaire de paramètres n et k forment une base de l'espace vectoriel du code.*

Soit X un élément source de $\mathbb{F}_q^k$ et G la matrice génératrice d'un code (k, n). X est encodé en un mot de l'espace de sortie $Y \in \mathbb{F}_q^n$ par la multiplication à gauche par la matrice génératrice du code : $Y = XG$.

Définition 2.7. *Un code linéaire est dit systématique, lorsque pour chaque vecteur de l'espace source, chaque symbole de ce vecteur se trouve parmi les symboles encodé. La matrice génératrice d'un tel code contient donc, à une permutation près, la matrice identité. Plus précisément la matrice génératrice G_s peut s'exprimer sous la forme :*

$$G_s = (Id_k|M)$$

De nombreux types de codes utilisent cette structure de matrice génératrice, cependant une large famille de code, notamment les codes de type LDPC, utilisent la structure suivante.

Définition 2.8. *Soit $\mathcal{C}$ un code de dimension k et de taille n. Une matrice H est appelée une matrice de parité de code si et seulement si $\forall X \in \mathcal{C}, HX = 0$ et H est de rang plein. En d'autres termes, H est une matrice de parité si et seulement si H est de rang plein et $Ker(H) = \{\mathcal{C}\}$*

Propriété 2.9. *Soit un code systématique $\mathcal{C}$ de paramètres (k, n), dont la matrice génératrice G s'écrit sous la forme : $G = (Id_k|M)$. Alors la matrice $H = (-M^T|Id_{n-k})$ est une matrice de parité de ce code.*

Démonstration. Directe. □

Le décodage d'un code correcteur ayant comme objectif de retrouver la séquence ayant été la plus probablement envoyée, et donc la plus proche de celle reçue, il est nécessaire d'introduire une notion de distance. Dans la théorie des codes, la mesure utilisée pour la distance est la distance de Hamming, basée sur le poids de Hamming.

Définition 2.10. *Le poids de Hamming d'une chaîne de symboles sur un alphabet donné est le nombre de symboles différentes du symbole nul. Dans le cas du corps $\mathbb{F}_2$, le poids de Hamming d'un vecteur est le nombre de bits à 1 du vecteur.*

Définition 2.11. *La distance de Hamming entre deux vecteurs a et b de symboles est le poids de Hamming du vecteur $a - b$.*

On montre aisément que cette définition vérifie les conditions nécessaires à une distance (symétrie, nullité, inégalité triangulaire).

L'application de la distance de Hamming à un code, amène à s'intéresser à la distance minimale d'un code.

Définition 2.12. *La distance minimale d_{min} d'un code linéaire est la distance de Hamming minimale existant entre deux mots du code distincts. Grâce à la structure d'espace vectoriel du code, elle est équivalente au poids minimal de l'ensemble des mots du code.*

Cette distance minimale d'un code a un impact direct sur la capacité de correction d'un code. En effet, celle-ci traduit la distance minimale existant entre deux mots du code, et donc dans le cadre d'un code à effacements, celui-ci sera toujours capable de corriger un nombre d'effacements strictement inférieur à cette distance.

Cette distance minimale possède une borne supérieure qui dépend de la longueur et de la dimension du code. Les codes qui atteignent cette borne supérieure sont appelés codes à maximum distance séparable (MDS) et possèdent une capacité de correction optimale.

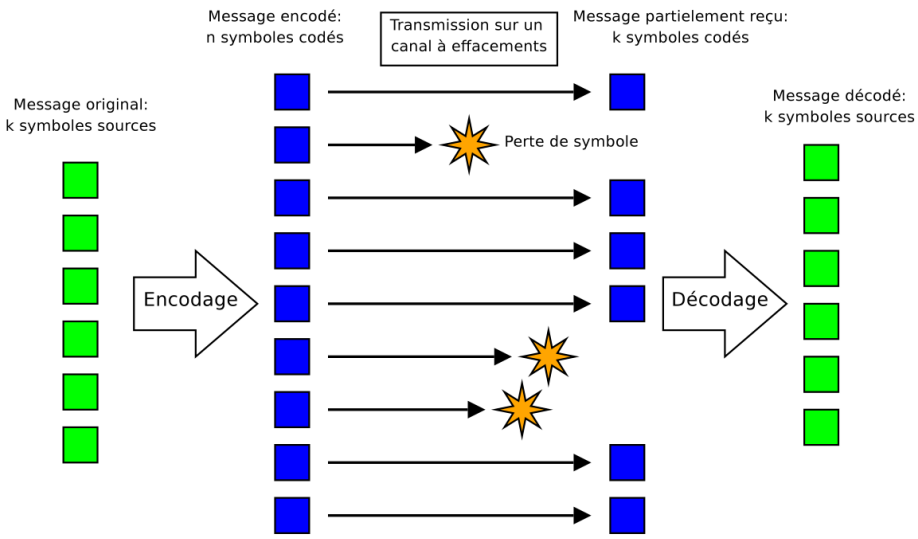

Figure 2.2 – Représentation de l'encodage et du décodage d'un code à effacements

Définition 2.13. *Un code linéaire de longueur n et de dimension k est dit MDS si et seulement si $n - k = d_{min} - 1$. On dit alors que le code atteint la borne de Singleton [3].*

Dans le cadre des codes à effacements, un code est dit MDS si et seulement si tout sous-ensemble de k éléments reçus parmi les n symboles encodés permet de décoder les k symboles sources.

Nous pouvons alors résumer le principe d'encodage et de décodage d'un code à effacements grâce à la Figure 2.2.

Cette notion de code MDS est essentielle dans le cadre des codes à effacements, car les contraintes associées à ces codes entraînent que leur champ d'application pratique est totalement différent. Nous proposons alors de séparer les codes MDS et les codes non-MDS pour la suite de notre étude.

2.2.4 Les codes à effacements MDS

Il a été prouvé qu'en dehors des cas triviaux (code à répétition,...), il n'existe pas de codes MDS sur le corps des binaires. Ces codes MDS possèdent un intérêt théorique capital, en étant notamment, la clé d'un partage optimal de l'information [4] [5] [6].

Dans le cas général, les principaux codes MDS connus sont les codes de Reed-Solomon et leurs extensions. Le principe de ces codes a été introduit par Irvine Reed et Gustave Solomon [7] en 1960. L'idée est de considérer chaque symbole d'un bloc comme un coefficient d'un polynôme que l'on va sur-échantillonner. Le décodage sera alors équivalent à l'interpolation d'un polynôme de degré $k-1$ sur au moins k positions.

Les codes de Reed-Solomon (RS) jouissent d'une grande popularité et se retrouvent dans des milieux aussi variés que les CD ou les sondes Voyager. En effet,

dès la fin des années 1960, la possibilité de décoder efficacement de tels codes sur le canal à erreurs est apparue grâce notamment aux travaux de Berlekamp [8] [9] et Massey [10]. L'algorithme de Berlekamp-Massey permet ainsi d'effectuer le décodage des codes RS sur le canal à erreurs avec une complexité $\mathcal{O}(n \log^2 n)$.

Cependant les mécanismes utilisés par cet algorithme ne sont pas directement adaptables au canal à effacements. C'est ainsi qu'à ce jour, la plupart des algorithmes actuels implémentant les codes de Reed-Solomon sur les effacements sont de complexité quadratique $\mathcal{O}(n^2)$. Des travaux ont été menés sur des algorithmes à complexité sous-quadratique sur ce canal [11], dès les années 1980, mais à ce jour aucune implémentation majeure de ces résultats n'a été menée.

Une des raisons de ce manque de résultats pratique provient sûrement du fait que les codes de Reed-Solomon sont très largement associés aux corps finis de type $\mathbb{F}_{2^m}$ qui, comme nous l'avons vu, sont très pratiques pour une implémentation machine, mais qui malheureusement ne permettent pas à ce jour de décodage sous-quadratique simple. Il faut toutefois noter que des travaux récents [12] ont mis en évidence la possibilité de créer des codes MDS avec un complexité associée de $\mathcal{O}(n \log^2 n)$, grâce à l'utilisation des transformées de Walsh.

La méthode la plus courante pour créer un code de Reed-Solomon sur le canal à effacements est de considérer directement la vue polynomiale du code, grâce à l'utilisation d'une matrice de Vandermonde comme matrice génératrice. Une approche détaillée de ces codes a été présentée par Plank [13] [14]. Luigi Rizzo a proposé une implémentation [15], qui sert à ce jour, de référence pour l'évaluation des codes RS. Ces codes ont été standardisés auprès de l'IETF [16].

Une matrice de Vandermonde est une matrice V de taille $m \times n$ construite à partir d'un vecteur de taille m, $(\alpha_0, \alpha_1, ..., \alpha_{m-1})$. Dans notre cas, ce vecteur contiendra des éléments d'un corps fini. Dans une matrice de Vandermonde, toutes les éléments d'une colonne suivent une progression géométrique. En d'autres termes :

$$V = \{v_{i,j}\}_{0 \leq i < m; 0 \leq j < n}, \quad v_{i,j} = \alpha_i^j$$

ou de manière plus explicite :

$$V = \begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \alpha_0^3 & \dots & \alpha_0^{n-1} \\ 1 & \alpha_1 & \alpha_1^2 & \alpha_1^3 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \alpha_2^3 & \dots & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_{m-1} & \alpha_{m-1}^2 & \alpha_{m-1}^3 & \dots & \alpha_{m-1}^{n-1} \end{pmatrix}$$

Lorsque la matrice de Vandermonde est carrée ($m = n$), celle-ci est inversible si et seulement si les α_i sont tous différents deux à deux. Ceci peut-être déterminé par le fait que la famille des vecteurs lignes est une famille libre ou liée suivant le cas, ou bien grâce au calcul du déterminant $\Pi_{0 \leq i < j < n}(\alpha_j - \alpha_i)$.

De la même manière, il a été démontré que toute sous-matrice carrée $m \times m$ d'une

matrice de Vandermonde $m \times n$, où $n \geq m$ était inversible ce qui prouve le caractère MDS du code.

Il est alors intéressant de remarquer que si V est une matrice de Vandermonde de taille $k \times n$, où les α_i sont tous différents, et si $a = (a_0, a_1, ..., a_{k-1})$ représente un vecteur d'éléments du corps fini avec le polynôme associé $a(x) = \sum_{i=0}^{k-1} a_i x^i$, alors le produit matrice-vecteur est équivalent à l'évaluation du polynôme $a(x)$ sur les points α_i :

$$\begin{pmatrix} a(\alpha_0) \\ a(\alpha_1) \\ a(\alpha_2) \\ \vdots \\ a(\alpha_{k-1}) \end{pmatrix} = \begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \alpha_0^3 & \dots & \alpha_0^{n-1} \\ 1 & \alpha_1 & \alpha_1^2 & \alpha_1^3 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \alpha_2^3 & \dots & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_{m-1} & \alpha_{m-1}^2 & \alpha_{m-1}^3 & \dots & \alpha_{m-1}^{n-1} \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{k-1} \end{pmatrix}$$

L'encodage d'un code RS peut alors être vu comme l'évaluation d'un polynôme de degré $k-1$ sur n points. Le décodage d'un tel code peut alors être vu comme un problème d'interpolation polynômiale. Cette vision permet alors de retrouver l'ensemble des résultats existant sur l'interpolation polynomiale, et permet de démontrer d'une autre manière le caractère MDS de ces codes.

En effet, les résultats d'interpolation polynomiale montrent qu'il existe un unique polynôme de degré $k-1$ passant par k points précis.

Si ces codes sont des codes optimaux du point de vue de leur capacité de correction, et qu'ils possèdent des implémentations efficaces, ils sont cependant pénalisés par leur complexité d'encodage et de décodage quadratique $\mathcal{O}(n^2)$. En pratique, sur le canal à effacements de paquets, cela se traduit par des vitesses d'encodage/décodage prohibitives dès lors que la longueur des codes dépasse plusieurs centaines d'éléments. C'est d'ailleurs cette complexité intrinsèque et l'impossibilité d'utiliser des codes MDS pour de grandes longueurs qui a permis la redécouverte et le déploiement croissant de solutions sous-optimales mais largement moins coûteuses de type LDPC et Raptor, que nous détaillerons dans le paragraphe suivant.

Il existe une autre méthode de construction de codes MDS, basée non pas sur des matrices de Vandermonde, mais sur des matrices de Cauchy, qui possèdent des propriétés similaires. Une matrice de Cauchy C de taille $m \times n$ est construite grâce à deux vecteurs du corps fini : $(x_0, x_1, ..., x_{m-1})$ et $(y_0, y_1, ..., y_{n-1})$ tous deux à deux distincts,

$$C = \{c_{i,j}\}_{0 \leq i < m; 0 \leq j < n}, \quad c_{i,j} = \frac{1}{x_i - y_j}$$

Cette matrice possède les mêmes propriétés d'encodage, de décodage, et d'optimalité que les codes de Reed-Solomon classiques. Les codes construits sur ces matrices de Cauchy ont été introduits pour la première fois par McWilliams et Sloane [17].

Ces matrices ont été utilisées pour la première fois en tant que codes à effacements en 1989 par Rabin [6] pour améliorer la fiabilité du stockage distribué. Dans le cadre de la fiabilisation de transmissions dès 1995 [18] couplées à des mécanismes d'optimisation bas niveau. Les résultats de simulation sur ces codes ont montré que ceux-ci respectaient mieux la complexité quadratique que les codes de Rizzo et obtenaient de meilleurs performances sur des longues tailles de code, sans que celles-ci soient réellement utilisables. Ce code possède des éléments brevetés.

Les matrices de Cauchy ont également servi de support sur des codes implémentés au sein de l'ISAE par Jérôme Fimes et al. [19]. Le corps fini d'étude est ici identique à celui que nous utiliserons lors du chapitre 6. Contrairement aux codes précédents, les opérations basiques sont ici sur le corps des entiers *modulo* un nombre premier, ce qui a permis de nombreuses optimisations, notamment sur des systèmes 64-bit. Les résultats de simulation ont montré que ces codes étaient bien plus rapides que les deux autres présentés ci-dessus, tout en étant, néanmoins, toujours soumis à une complexité intrinsèquement quadratique.

Dans le chapitre 6, nous proposerons une implémentation des codes de Reed-Solomon, non pas sur les corps du type $\mathbb{F}_{2^m}$, mais sur des corps du type $\mathbb{F}_p$ où p est un nombre de Fermat premier. Nous verrons que sur ce type de corps, il est possible de définir un encodage et un décodage systématique sur le canal à effacements de paquets de complexité $\mathcal{O}(n \log n)$ grâce à l'utilisation de transformées de Fourier. Nous verrons que ces codes permettent alors une utilisation de codes MDS sur des longueurs pouvant atteindre plus de 10.000 symboles avec un vitesse de plus de 50Mbps.

2.2.5 Les codes à effacements non-MDS

Comme nous l'avons vu, le défaut majeur des codes MDS actuels concerne leur complexité théorique prohibitive pour de de grandes longueurs. L'extension naturelle de l'algorithme de décodage des codes MDS aux autres types de codes peut être défini de la manière suivante.

Définition 2.14. *Le décodage à maximum de vraisemblance d'un code à effacements quelconque consiste à vérifier que la sous-matrice de la matrice génératrice correspondant aux symboles reçus est de rang k, où k est la dimension du code. Si cette condition est vérifiée, la sous-matrice de rang k est inversée et les symboles d'information sont obtenus en multipliant la matrice inversée par le vecteur de symboles reçus.*

Cet algorithme peut être appliqué à n'importe quel code à effacements. Il est est optimal en termes de capacité de correction et donne donc les limites du code par rapport à ce paramètre. Le point faible de cet algorithme est sa complexité : l'inversion de la matrice est de complexité cubique (pour une matrice non-structurée) et le produit matrice-vecteur est de complexité quadratique.

L'objectif d'un algorithme de décodage d'un code donné est de se rapprocher des performances ML en termes de capacité de correction tout en réduisant la complexité

du codage et du décodage. La capacité de correction peut être exprimée formellement de la manière suivante :

Définition 2.15. *L'inefficacité d'un code linéaire de dimension k et de taille n correspond au nombre moyen de symboles nécessaires pour le décodage rapporté au nombre de symboles sources. Ce ratio est donc supérieur ou égal à un dans le cas général et est égal à un si et seulement si le code est un code MDS. Soit $\mathcal{C}$, et nb_r le nombre moyen de symboles nécessaires au décodage, alors l'inefficacité du code peut s'exprimer :*

$$ineff_{\mathcal{C}} = \frac{nb_r}{k}$$

Lorsque l'inefficacité est exprimée sous forme de pourcentage, elle correspondra tout simplement à $ineff_{\mathcal{C}} - 1$.

La valeur $nb_r - k$ correspond à ce que l'on appellera le nombre moyen de symboles supplémentaires, ou extra-symboles.

Un code non parfait sera ainsi évalué selon deux métriques : sa vitesse et sa capacité de correction, dont une bonne indicatrice est son inefficacité.

Plusieurs grandes familles de code se distinguent pour ces codes non parfaits. Nous retiendrons principalement les codes de Reed-Muller, les codes de type LDPC et les codes dits *sans rendement*. La construction des codes de Reed-Muller étant présentée dans le chapitre 5, nous nous attardons ici sur les deux autres types de codes.

2.2.5.1 Les codes à matrice de parité creuse : LDPC

Les origines des codes *Low Density Parity Check* ou LDPC proviennent de la thèse de Robert Gallager [20] présentée en 1963. Leur appellation provient tout simplement du fait que la densité de la matrice de parité de ces tels codes est faible, ou en d'autre termes, qu'ils possèdent une matrice de parité creuse. Les éléments de ces matrices étant binaires, cela signifie que celles-ci possèdent un nombre limité d'éléments à 1. Cette faible densité de matrice permet d'utiliser un type particulier de décodage basé sur le principe de propagation de croyance ou *belief propagation*. Ces algorithmes ont pour intérêt d'être de complexité linéaire en la longueur du code, tout en conservant une inefficacité relativement faible.

Le lecteur avisé aura remarqué qu'il est tout à fait possible d'utiliser ce mécanisme directement sur la matrice génératrice du code. Cependant, la diversité du code, qui impacte directement sur sa capacité de correction, s'en trouve alors largement amoindrie, du fait que la matrice génératrice doive être creuse. Vu autrement, chaque symbole source se retrouve alors dans un nombre très limité de combinaisons linéaires (de symboles d'encodage), et celui-ci ne sera alors décodable que lorsque au moins un de ces symboles d'encodage sera reçu. Ces codes sont connus sous le nom de Low Density Generator Matrix (LDGM) et nécessitent un ajustement précis de leur matrice afin d'être réellement efficaces. Les LDGM peuvent également être considérés comme des codes LDPC particuliers, dans le sens, où les LDGM sont des codes

LDPC dont chaque symbole de redondance ne dépend uniquement que des symboles sources, et pas d'autres symboles de redondance.

En tout état de cause, l'état des technologies a fait que, suite à leur découverte, les codes LDPC ont été "oubliés" pendant une trentaine d'années, jusqu'à qu'ils soient "redécouverts" par MacKay [21] dans les années 1990. Depuis ceux-ci constituent un domaine d'étude ouvert qui a permis la création de nombreux codes dérivés pour le canal à effacements, tels que les codes Tornado [22], les codes LT [23] et leurs successeurs, les codes Raptor [24], que nous étudierons dans la section suivante de ce manuscrit.

Les méthodes de décodage de type *belief propagation* sur le canal à effacements proviennent plus ou moins directement des travaux de Zyablov et Pinsker [25] en 1974. On parlera alors de décodage de type *itératif*. ce décodage n'est pas optimal pour un code donné, par rapport à décodage au maximum de vraisemblance (ML) donné, mais possède l'intérêt majeur d'être de complexité linéaire.

Tout comme pour les autres codes linéaires, la capacité de correction des codes LDPC dépend du poids des mots de codes et en particulier de la distance minimale du code. Lorsqu'ils sont décodés avec un décodeur itératif, les capacités de correction des codes LDPC seront également limitées par des caractéristiques telles que la longueur des cycles dans le graphe associé et la taille des *stopping sets*, que nous présenterons dans la suite de cette section.

Ces codes peuvent être utilisés en tant que codes systématiques, selon que les symboles sources sont envoyés ou non en plus des symboles de redondance.

Dans la suite de cette section, nous présenterons les codes LDPC, leur encodage, leur décodage ainsi que le problème des *stopping sets*.

Représentation des codes LDPC Deux représentations usuelles et totalement équivalentes existent pour représenter un code LDPC. Selon le niveau de simplification apporté à la résolution d'un problème donné, l'une ou l'autre des représentations sera utilisée.

La première représentation d'un code LDPC est basée sur un graphe biparti ou graphe de Tanner [26]. Un graphe est dit biparti s'il existe deux ensembles de nœuds et un ensemble d'arêtes, tel que chaque arrête relie deux nœuds d'un ensemble différent.

Un code LDPC peut être défini comme un graphe biparti dont l'ensemble des nœuds de gauche, appelés nœuds variables V représente un mot de code, et l'ensemble des nœuds de droite, correspond à des nœuds de contrainte C. La représentation du graphe de Tanner d'un code LDPC se trouve sur la Figure 2.3.

Dès lors, une suite de symboles constitue un mot du code si et seulement si pour chaque nœud de contraite, la somme des symboles correspondants aux nœuds variables adjacents est nulle.

La représentation d'un code LDPC sous la forme d'un graphe de Tanner est particulièrement utile pour représenter le mécanisme de passage de messages qui intervient dans le décodage itératif. De plus, cette représentation permet également de faire ap-

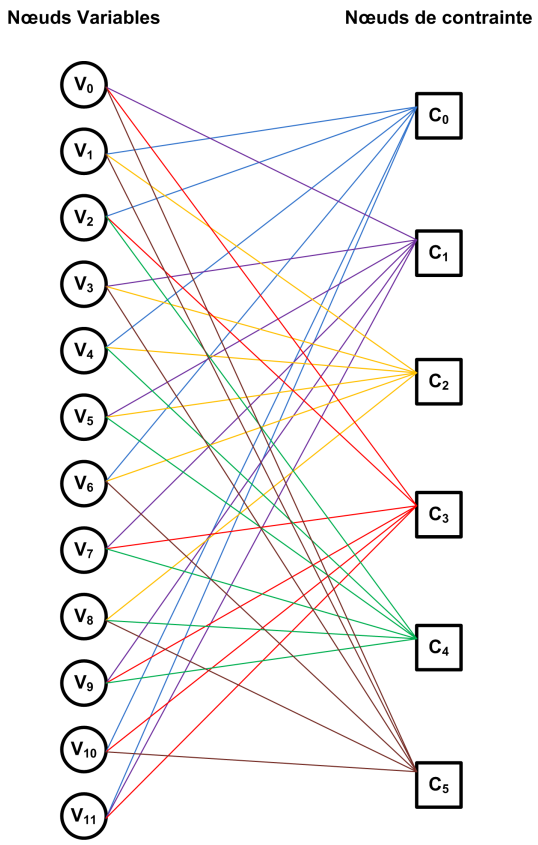

Figure 2.3 – Graphe de Tanner d'un code LDPC

pel aux puissants outils de la théorie des graphes pour étudier et concevoir les codes LDPC.

Il est possible de représenter ce graphe sous la forme d'une matrice dont chaque colonne correspond à un nœud variable et chaque ligne un nœud de contrainte. Les éléments non-nuls de cette matrice, d'indices (i, j) correspondent alors aux arêtes reliant le nœud variable j au nœud de contrainte i. Cette matrice est bien une matrice de parité du code car tout vecteur du code produira, par définition, une sortie nulle par cette matrice.

Pour correspondre à un code LDPC, il est alors nécessaire de garantir que le nombre d'arêtes du graphe soit limité. En rappelant que pour un code linéaire donné, il existe plusieurs matrices de parité équivalentes à des combinaisons linéaires près, on choisira alors avec soin, la matrice de parité la plus creuse possible.

D'après la définition 2.8, une matrice de parité d'un code linéaire de dimension k et de longueur n sur $\mathbb{F}_q$ est une matrice de rang plein de taille $(n-k)\times n$. Lorsque le code est systématique, les k premières colonnes (à une permutation près) correspondent aux symboles sources et les autres aux symboles de redondance. Dans la suite de

ces travaux, nous travaillerons avec une matrice de parité binaire. Les symboles de redondance seront alors appelés symboles de parité. La partie gauche correspondant aux symboles sources sera appelée A et celle des symboles de redondance U.

Si S représente un ensemble de symboles sources et P un ensemble de symboles de parité correspondant, la produit matriciel $(A|U)(S|R)^T$ est alors nul. La matrice de parité peut être vue comme un ensemble de combinaisons linéaires (par ligne) dont le résultat est nul pour un mot de code donné (sources et parités).

A titre d'exemple, la matrice de parité du code correspondant au graphe de Tanner de la Figure 2.3 est la suivante ($k = 6, n = 12$) :

$$H = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

ce qui donne six combinaisons linéaires entre les six premiers symboles sources et les six symboles de redondance.

La version matricielle permet de définir la notion de degré (ligne ou colonne), correspondant au nombre d'entrées non-nulles. Un code LDPC sera dès lors considéré comme régulier ou irrégulier selon que la distribution de ses degrés lignes et colonnes est constante.

Encodage des codes LDPC L'encodage classique par la matrice génératrice n'a pas forcément de sens ici pour deux raisons majeures : cette matrice n'est pas initialement déterminée, et, pour une matrice génératrice non-creuse, le produit matrice-vecteur s'accompagne nécessairement d'une complexité quadratique.

L'encodage des codes LDPC peut être directement effectué sur la matrice de parité. L'idée est alors de construire celle-ci (par des combinaisons linéaires) afin de faire apparaître une structure quasi-triangulaire inférieure sur la partie des symboles de redondance U [27].

Dans la suite de nos travaux, la matrice U sera toujours considérée comme triangulaire inférieure. La conséquence est qu'il est alors possible de calculer chaque symbole de parité directement par une combinaison de symboles sources et de symboles de parité déjà calculés. La matrice de parité étant creuse, l'encodage effectué est alors proche d'une complexité linéaire.

Décodage des codes LDPC En plus d'un décodage à maximum de vraisemblance (ML) optimal mais complexe, les codes LDPC sont adaptés pour un décodage dit itératif sur le canal à effacements.

Comme nous l'avons précédemment évoqué, le décodage itératif est sous-optimal en terme de capacité de correction, mais possède l'intérêt d'avoir une complexité linéaire en la longueur du code.

Le principe du décodage itératif introduit par Gallager puis par Zyablov peut très bien s'interpréter sur le graphe de Tanner associé à un code LDPC, comme sur la figure 2.4. Sur le graphe de Tanner, tous les symboles du mot de code sont reliés entre eux grâce aux nœuds de contraintes. On rappelle alors que la somme de ces nœuds variables doit être nulle sur chaque nœud de contrainte. En d'autres termes, dans la cas binaire, tout nœud variable peut s'exprimer comme la somme des autres nœuds variables reliés par le même nœud de contrainte.

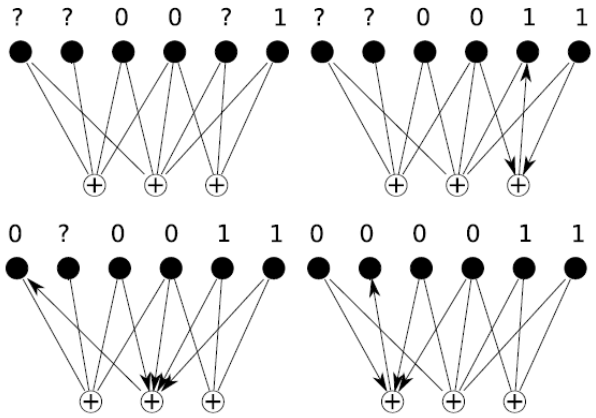

Figure 2.4 – Exemple de décodage itératif sur un graphe de Tanner

Ainsi, pour chaque nœud de contrainte, le décodage itératif peut s'effectuer si et seulement si un seul des symboles relié est inconnu. Celui-ci permettra éventuellement par la suite, lui même étant alors décodé, de décoder un autre symbole du mot de code grâce à un autre nœud de contrainte, et ainsi de suite.

Soit d_m le degré moyen des colonnes de la matrice associée. Lorsque le décodage itératif est complet, celui-ci a donc consisté en $(n-k)$ sommes, correspondant aux symboles manquant, de $d_m - 1$ éléments en moyenne. Le nombre d'opérations maximal effectué est donc $(n-k) \times (d_m - 1)$. d_m étant fixé pour un code donné, on retrouve la complexité linéaire de décodage annoncée.

Bien entendu, les limites d'un tel décodage apparaissent dès lors qu'il n'y a plus à un moment donné, de nœud de contrainte relié à des symboles dont seulement un seul est inconnu. Dès lors, le décodage est impossible, et deux options s'offrent alors.

La première solution consiste à attendre l'arrivée de nouveaux symboles qui a un impact immédiat sur la capacité de correction du décodage itératif et plus précisément son inefficacité.

La seconde solution revient à tenter un décodage ML sur le bloc de la matrice correspondant aux symboles non décodés. Cette solution apporte cependant deux inconvénients. Tout d'abord le succès d'un tel décodage n'est pas garanti, et même si celui-ci réussit, il peut avoir un coût important, si tant est que le système restant à décoder est large. En outre, cette méthode a pour effet de briser la complexité linéaire théorique du décodage itératif.

Le cas des codes LDPC à structure droite triangulaire inférieure Comme nous l'avons évoqué précédemment, les codes LDPC dont la matrice U est triangulaire inférieure permettent un encodage en temps linéaire des symboles de redondance, et ce dans leur ordre naturel. On observe une sorte d'accumulation de ces symboles au fur et à mesure de leur construction, due aux éléments non-nuls et non-diagonaux de U. Ainsi il est courant de trouver ces codes sous la dénomination de codes "Repeat Accumulate". Ce type de code ainsi que cette dénomination ont été introduits par D. Divsalar et *al.* [28].

Parmi les codes LDPC de ce type, on trouve notamment les codes dits LDPC-Staircase et LDPC-Triangle standardisés par l'IETF [29].

De manière synthétique, la matrice U des codes LDPC-Staircase possède une structure en escalier : la diagonale et sa diagonale inférieure sont égales à 1, lorsque tout le reste de la matrice est à 0. Vu autrement, un symbole de redondance est la somme de symboles sources et du symbole de redondance directement antécédent. La partie de gauche possède quant à elle un degré constant, les éléments non-nuls de chaque colonne étant générées pseudo-aléatoirement.

Les codes LDPC-Triangle sont un alternative à ces codes, où de nouvelles arêtes sont déployées entre des symboles de redondance et d'autres générés précédemment. Il est ainsi toujours possible d'utiliser un encodage linéaire, la matrice U devenant alors une matrice triangulaire inférieure plus remplie. En conséquence, si la fiabilité du code semble améliorée, ceci est bien entendu fait au prix d'un plus grand nombre d'opérations somme.

Néanmoins, malgré leur simplicité apparente, ces codes permettent d'obtenir un bon compromis entre une faible efficacité et une faible complexité.

Dans le chapitre 4, nous présenterons des codes à décodage hybride, permettant un décodage itératif sur leur matrice de parité, dont la structure droite est triangulaire inférieure.

La problématique des stopping sets Afin de comprendre les limites du décodage itératif d'un code LDPC, il est important de s'intéresser au mécanisme des *stopping sets*.

Définition 2.16. *Soit un code LDPC et son graphe de Tanner associé. Un ensemble de nœuds variables (de symboles) est un stopping set du graphe si et seulement si tous les nœuds de contrôle reliés à cet ensemble le sont par au moins deux de ces nœuds variables.*

La taille d'un stopping set correspond au cardinal de l'ensemble des nœuds variables.

Ces *stopping sets* sont particulièrement gênants pour l'inefficacité d'un code LDPC. Imaginons un *stopping set* d'un graphe dont tous ses symboles sont effacés. L'ensemble des nœuds de contrôle reliés l'étant par au moins deux de ces symboles à chaque fois, il est impossible de terminer le décodage itératif tant qu'un seul des éléments du *stopping set* ne sera pas reçu.

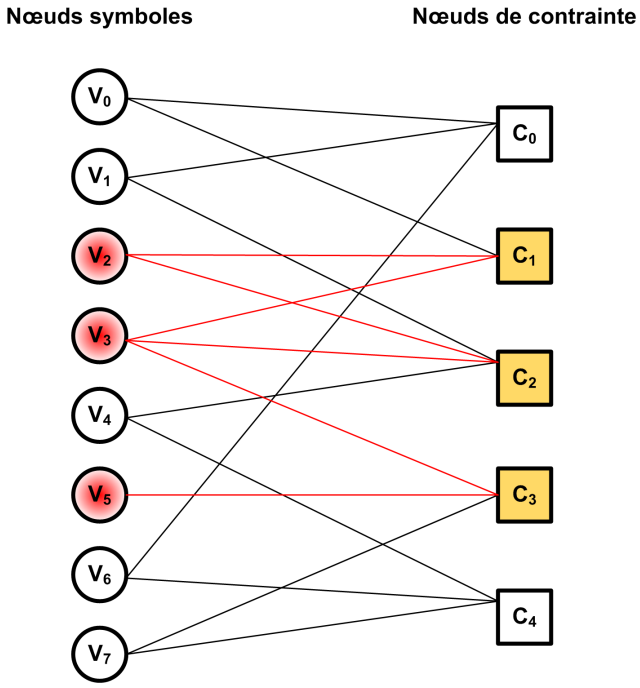

Figure 2.5 – Exemple de stopping set dans un graphe de Tanner d'un code LDPC

A titre d'exemple la Figure 2.5 montre un *stopping set* de taille 3 : $\{v_2, v_3, v_5\}$. On peut observer que, même si tous les autres symboles sont reçus, le décodage ne peut pas aboutir tant que v_2, v_3 ou v_5 ne sera pas reçu.

On aperçoit dès lors l'importance de la taille d'un *stopping set*, et plus précisément de la taille des plus petits *stopping sets* d'un même graphe de Tanner. En effet, plus le *stopping set* est petit, plus la probabilité de non-réception de tous ses élements est forte, empêchant un décodage itératif complet.

Ces résultats mettent en lumière l'intérêt de surveiller ces *stopping sets* lorsque un code LDPC est déployé, dans la mesure du possible. En effet, cette étude est souvent rendue complexe par les contraintes que l'on impose aux codes LDPC en plus des contraintes initiales.

2.2.5.2 Les codes sans rendement

Les codes sans rendement (*rateless* en anglais), ou codes fontaine, sont une catégorie de code partageant une paternité lointaine avec les codes LDPC. Ceux-ci sont nommés ainsi car ils ne possèdent pas de rendement défini *à priori* : il est possible de générer de nouveaux symboles de redondance à la volée, à partir d'un ensemble de k symboles sources, qui sont par ailleurs décorrélés des symboles précédemment générés.

Il convient bien de noter que s'il est possible de générer à volonté des symboles, il n'est pas possible d'en générer cependant une infinité. Tout d'abord de manière

théorique, les symboles de redondance correspondant à des combinaisons linéaires de k symboles sources, le nombre de symboles de redondance différents sera donc de 2^k. La seconde limitation est par contre bien plus pratique. En effet, afin de distinguer chaque symbole de redondance, on affecte à ceux-ci un identifiant unique. Par exemple, dans la RFC5053 [30] qui définit l'implémentation de codes Raptor *rateless*, cet identifiant est codé sur 16 bits, ce qui limitera alors en pratique la longueur du code à 65536.

Les codes LT La première version de codes sans rendement permettant la génération de symboles de redondance à la volée provient historiquement de Michael Luby, qui donna son nom à ces codes : *Luby Transform* codes [23].

Le principe de ces codes est le suivant. Soit un ensemble de k symboles source. Pour chaque symbole de redondance, on tire un degré d, qui est choisi suivant une distribution de degrés particulière, un décalage compris entre 0 et k et un intervalle compris entre 1 et $k-1$. Ce symbole de redondance sera alors la somme de d symboles sources, dont le premier sera celui d'indice égal au décalage choisi, et les suivant seront ceux d'indice égal au précédent auquel on ajoute l'intervalle *modulo* k.

Par exemple, soit un ensemble de $k = 100$ symboles sources, et soit un symbole de redondance à créer. Soit $d = 5$ son degré, $o = 33$ son décalage et $i = 43$ son décalage. Ce symbole de redondance correspondra à la somme des symboles source d'indice 33, 76, 19, 62 et 5.

En pratique le décalage et l'intervalle sont choisis par une générateur pseudo-aléatoire.

Ici, chaque symbole de redondance n'étant la somme que de symboles sources, ce code est donc un code de type LDGM irrégulier.

M. Luby a montré que lorsque ces codes sont décodés de manière itérative, ceux-ci sont asymptotiquement optimaux, sur le canal à effacements, si tant est que la distribution des degrés soit soigneusement choisie. Dans le même article, Luby propose alors d'utiliser la distribution de degrés Soliton robuste. En utilisant cette distribution, on montre qu'il est possible de décoder avec $\frac{\log^2(k)}{\sqrt{k}}$ extra-symboles en moyenne avec une complexité globale de $\mathcal{O}(k \log k)$. On voit dès lors qu'en faisant tendre la dimension du code vers l'infini, l'inefficacité se rapproche inexorablement de 1, et les codes LT deviennent alors asymptotiquement MDS avec une complexité de décodage logarithmique.

En pratique, les codes LT possèdent néanmoins deux défauts majeurs et rédhibitoires :

- Pour des dimensions de codes classiques ($k < 50.000$), ces codes souffrent d'une forte inefficacité. Les résultats d'expérimentation sur des distributions de codes LT particulières ont montré une inefficacité moyenne supérieure à 10%.
- Ces codes ne sont pas, et ne peuvent pas, par nature, être systématiques.

Par ailleurs, tous comme les codes LDPC, les performances de ces codes sur de petites dimension ($k < 100$) sont très médiocres, ceci étant du au manque de "relations" inter-symboles inhérent à ce types de codes.

Les codes Raptor Les codes Raptor ont été crées suite à ces résultats afin de pallier les défauts majeurs des codes LT. L'idée proposée par Amin Shokrollahi [24] est d'ajouter un mécanisme de précodage aux codes LT, permettant à la fois de régler le problème du systématisme et de l'inefficacité, sans pour autant apporter de complexité théorique supplémentaire.

A ce jour, les codes Raptor sont généralement considérés comme le meilleur compromis existant sur les codes à effacements. Les résultats de simulation que nous détaillerons par la suite montrent que l'inefficacité de ces codes est faible, tout en supportant des débits élevés. Ces codes ont été standardisés au sein de l'IETF [30], du 3GPP [31] et du DVB [32]. Ils sont en outre l'objet de nombreux brevets.

La partie qui va suivre provient de l'étude que l'ISAE et l'INRIA Grenoble ont effectué lors du développement d'un codec Raptor dans le cadre d'un contrat avec le CNES [1]. La description de ces codes provient de la standardisation de ces codes à l'IETF.

Le codage Raptor correspond à la concaténation de deux étapes : un pré-codage produisant des symboles intermédiaires et un encodage LT de ces symboles intermédiaires afin de créer les symboles réellement transmis. Le mécanisme global de l'encodeur est schématisé sur la Figure 2.6.

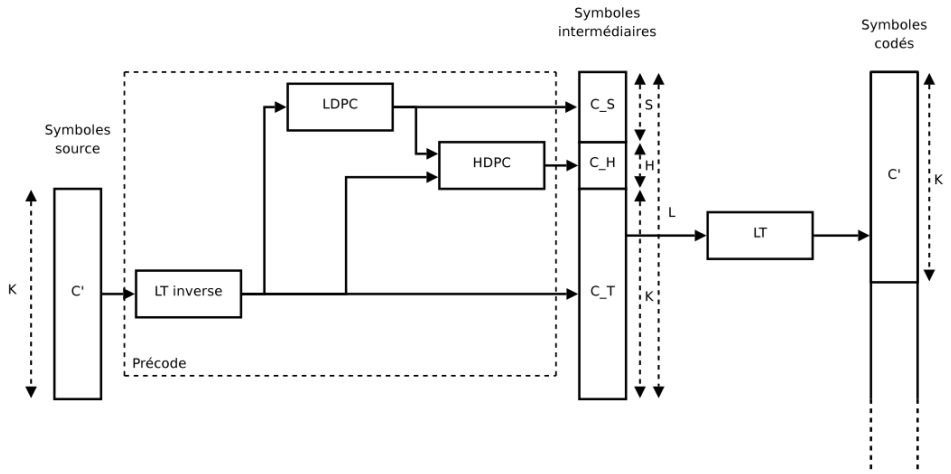

Figure 2.6 – Schéma général de l'encodeur Raptor

Détaillons tout d'abord le précodage Raptor. Celui-ci est composé de trois blocs :

- Un bloc LT inverse ou LT^{-1} qui correspond comme son nom l'indique à l'inverse d'un encodage LT. L'intérêt d'un tel bloc, placé au début du précode, est de garantir que le code produit est systématique, les symboles intermédiaires passant par la suite par un encodeur LT.
- Un bloc LDPC qui permet de créer s symboles, appelés symboles LDPC, à partir

1. "Logiciel de codage/décodage Raptor", Copyright ©CNES, Tous droits réservés.

de la sortie du LT^{-1}. Son nom vient du fait que la matrice génératrice de ces symboles est creuse.
- Un bloc HDPC, qui produit h symboles appelés *half* symboles, et qui prend en entrée la sortie du LT inverse et les symboles LDPC.

Par rapport à la version des codes LT présentés ci-dessus et basés sur des distributions de type Soliton, l'encodeur LT des codes Raptor est adapté dans la RFC5053. La distribution Soliton qui permet un encodage asymptotiquement optimal, possède un degré moyen qui augmente avec la dimension du code, ce qui en conséquence, implique que la distribution LT classique n'est pas de complexité linéaire. Pour cela, la distribution de degrés présente dans le standard est indépendante de cette dimension. Ceci permet aux codes Raptor d'être, tous comme les codes LDPC, de complexité linéaire en la dimension du code. Bien entendu, l'optimalité asymptotique de la distribution est perdue, mais ce n'est pas en soi un problème, car, comme nous l'avons vu, celle-ci n'est même pas visible pour des dimensions de code usuelles.

Comme nous l'avons vu, l'encodage Raptor consiste en une étape de précodage produisant des symboles intermédiaires et un encodage LT de ces symboles.

Soit $c^{'}$ un ensemble de k symboles sources. Le précodage consiste à créer un ensemble c de $l = k+s+h$ symboles intermédiaires. Nous avons détaillé les l relations existantes entre ces symboles intermédiaires et les symboles sources.

Si G_{LT} correspond à la matrice de l'encodeur LT, nous avons :

$$c^{'} = G_{LT} \times c$$

Soit G_{LDPC} la matrice de taille $k \times s$ qui permet de générer les symboles LDPC c_s à partir des symboles c_t en sortie de LT^{-1}. Nous avons :

$$c_s = G_{LDPC} \times c_t \Leftrightarrow c_s + G_{LDPC}\, c_t = 0$$

De la même manière, si G_{Half} représente la matrice de taille $h \times (k+s)$ qui permet des générer les *half* symboles c_h à partir de la concaténation de c_t et c_s :

$$c_h = G_{Half} \times \begin{pmatrix} c_t \\ c_s \end{pmatrix} \Leftrightarrow c_h + G_{Half} \begin{pmatrix} c_t \\ c_s \end{pmatrix} = 0$$

On peut alors représenter ces l relations par une matrice A carrée. Nous avons alors :

$$\begin{pmatrix} 0_{s+h} \\ c^{'} \end{pmatrix} = \begin{pmatrix} G_{LDPC} & Id_s & 0 \\ G_{Half} & Id_h & \\ G_{LT} & & \end{pmatrix} \times \begin{pmatrix} c_t \\ c_s \\ c_h \end{pmatrix}$$

Une représentation plus claire de A se trouve sur la Figure 2.7. Un exemple réel de matrice A pour $k = 500$ symboles se trouve sur la Figure 2.8. Sur ce schéma, les éléments en noir représentent les indices égaux à 1 dans la matrice.

Le codec Raptor standardisé précise que les tailles s et h de ces symboles supplémentaires sont dépendantes de k, mais restent néanmoins faibles relativement à k. A

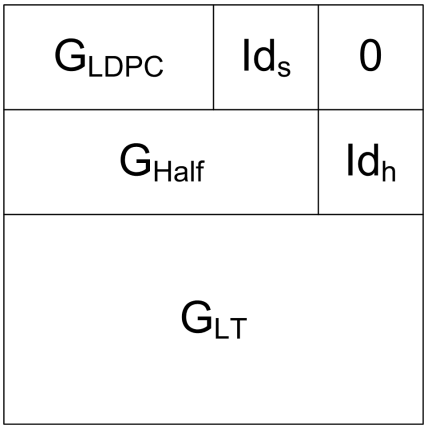

Figure 2.7 – Matrice A du code Raptor

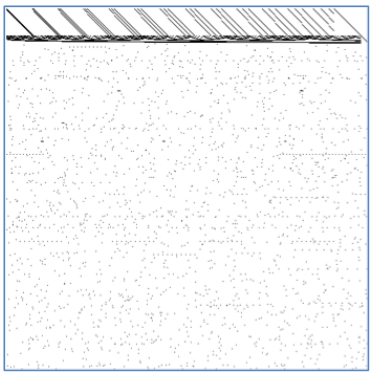

Figure 2.8 – Exemple de matrice A pour $k = 500$ et $l = 553$

titre d'exemple, pour un code de dimension 1000, s et h sont égaux respectivement à 59 et 13, ce donne un nombre de symboles intermédiaires de 1072.

La matrice A donne une relation directe entre les symboles intermédiaires et les symboles sources : $\binom{0_{s+h}}{c} = A \times c$. Le codec Raptor s'arrangeant pour toute dimension de code, à ce que A soit inversible, le précodage peut alors se ramener au produit des symboles sources (auquel on aura rajouté $l - k$ zéros au début) par A^{-1}.

L'encodeur systématique Raptor peut alors se ramener au produit matriciel $LT \times A^{-1}$. La matrice produit pour $k = 500$ est représentée sur la Figure 2.9 pour $n = 1000$ symboles générés.

Plusieurs points méritent une attention sur ce schéma :

- Le nombre de lignes est ici arbitrairement fixé à 1000. Bien entendu, les codes Raptor étant des codes sans rendement, il est possible de créer de nouveaux symboles à la volée, qui seraient alors représentés par de nouvelles lignes.
- La matrice comporte l colonnes. Comme nous l'avons fait remarquer, le vecteur d'entrée de cette matrice est le vecteur des symboles sources auquel sont ap-

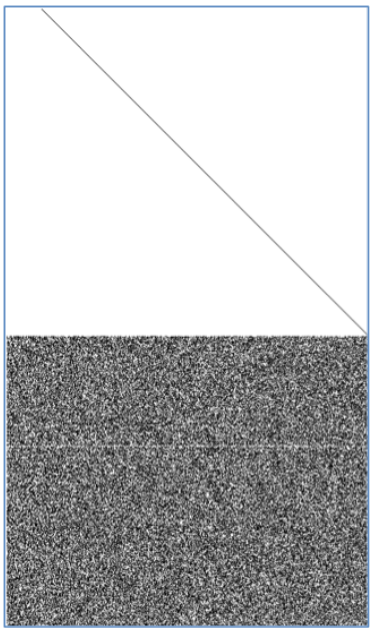

Figure 2.9 – Matrice de l'encodeur global Raptor pour $k = 500$ et $n = 1000$ symboles générés

posés au début $l - k$ symboles nuls. Au final, ces $l - k$ premières colonnes sont inutiles.

- La matrice comporte une diagonale démarrant après les $l-k$ premières colonnes. Ceci valide le caractre systématique du code Raptor.
- La partie non-systématique de la matrice globale n'a pas de structure particulière.

Toutes les informations concernant le décodage efficace des codes Raptor ne sont pas connues. Certains éléments permettent tout de même de comprendre le décodage de ces codes.

Le décodage des symboles sources à partir des symboles intermédiaires est un simple "encodage" LT de complexité linéaire. Le point crucial du décodage est donc la récupération de tous ces symboles intermédiaires. Sur ce point, les algorithmes de décodage utilisés sont similaires à ceux des codes LDPC, à savoir un décodage itératif, un décodage ML, ou un mélange des deux. A ce sujet, une méthode de décodage particulière a été brevetée [33] pour améliorer les performances d'un décodage ML par pivot de Gauss.

Si le nombre de symboles intermédiaires est de l, n'oublions pas cependant que ceux-ci sont liés par $s + h$ relations entre eux, ce qui rend bien le décodage possible dès lors que k symboles ont été reçus.

Les résultats de simulation que nous avons effectué sur la version standardisée de ces codes ont montré que le nombre moyen d'extra-symboles nécessaires au décodage des codes Raptor était quasiment constant quel que soit la dimension du code, et

proche de 2. Ceci est en accord avec les résultats officiels [34] qui donnaient un nombre moyen d'extra-symboles proche de $1,96$. L'intérêt majeur de ces codes est donc d'avoir un nombre d'extra-symboles moyen constant, et donc une inefficacité de plus en plus proche de l'optimal lorsque la dimension du code augmente. Néanmoins, nous avons également pu vérifier que ce nombre est sous-estimé lorsque les dimensions de codes sont faibles (< 100), comme annoncé, ce qui laisse entrevoir que ces codes, tout comme les codes LDPC ne sont pas adaptés pour ces dimensions, et où l'on favorisera les solutions de type Reed-Solomon ou Reed-Muller.

N'ayant pas accès au décodage réellement utilisé sur ces codes, et devant nous contenter d'une version standardisée de vitesse inférieure, nous n'avons pu comparer la vitesse de décodage de ces codes par rapport à d'autres codes. Sur ce point, la seule donnée existante provient de l'article original présentant ces codes qui annonce : *"The Raptor implementation of Digital Fountain reaches speeds of several gigabits per second, on a 2.4-GHz Intel Xeon processor, while ensuring very stringent conditions on the error probability of the decoder, even for very short lengths."* qui ne précise ni la dimension du code, ni le type de décodage, ni un ordre de vitesse précis.

En conclusion, les codes Raptor représentent d'excellents codes à effacements en termes de capacité de correction, point que nous avons pu corroborer lors de nos simulations. Néanmoins, leur complexité linéaire associée n'est pas clairement définie en pratique.

2.3 La compression d'en-têtes

2.3.1 Introduction

2.3.1.1 Le meilleur compromis "redondance/fiabilité" : l'objectif commun des codes à effacements et de la compression d'en-têtes

Cette section présente l'autre domaine d'application de la théorie de l'information dans le domaine des réseaux que nous avons abordé dans cette thèse : la compression d'en-têtes. La différence entre le domaine des codes à effacements et celui de la compression d'en-têtes réside dans le fait que, pour les codes à effacements, les données sont considérées être déjà compressées. On ne gère alors que l'ajout de redondance et la gestion de cette redondance par rapport à la fiabilité. Pour la compression d'en-têtes, les données à traiter, c'est-à-dire les en-têtes, contiennent déjà une redondance naturelle. Pour obtenir un bon compromis "redondance/fiabilité" (équivalent au ratio "compression/fiabilité) on pourrait se ramener au schéma des codes à effacements en compressant les données au maximum, puis en rajoutant de la redondance pour lutter contre les erreurs et les pertes de paquets. Ceci a déjà été proposé, par exemple dans [35]. Toutefois, vu le type de redondance (principalement inter-paquets), il apparaît que la meilleur compromis "redondance/fiabilité" est obtenu en intégrant directement la gestion de la fiabilité dans les mécanismes de compression en réalisant ainsi un type particulier de codage conjoint source-canal.

2.3.1.2 Introduction à la compression d'en-têtes

La compression d'en-têtes protocolaire a pour origine la volonté de diminuer la proportion de données prise par les piles protocolaires, au détriment des données effectives, et ceci, dans le but d'augmenter le débit disponible dans des réseaux à forte contrainte de bande passante. Le domaine d'application initial, dans le début des années 1990 concernait en majorité les réseaux classiques, mais avec l'avènement de nouveaux types de communication, cette théorie vise à s'étendre aux réseaux à fort coût, tels les réseaux satellitaires, mais aussi à forte contrainte, tels les réseaux mobiles. Par ailleurs, le développement d'applications de type VoIP a mis en évidence le besoin d'une telle compression, où la pile protocolaire peut représenter jusqu'à 50% de la taille des paquets transmis.

2.3.2 Compressed TCP / Van Jacobson

Une des premières approches concernant la compression d'en-têtes provient de deux protocoles, respectivement Thinwire-I et Thinwire-II définis par Farber et al. [36] en 1984. Ces deux mécanismes visaient à la compression des en-têtes TCP/IP dans le cadre d'une communication "mono-caractère" à très faible débit. En effet, de nombreuses applications, de type Telnet, nécessitaient l'envoi de données caractère par caractère, ainsi que leur acquittement respectif. En conséquence, pour chaque caractère, était apposé l'en-tête complet TCP/IP d'une longueur de 40 octets. Dans ce cas là, le paquet TCP/IP transmis est d'une longueur de 41 octets avec seulement un seul octet d'information.

Thinwire-I fonctionne sur un principe de drapeaux. Lorsque un nouveau paquet est envoyé, seuls les octets changeants entre l'en-tête précédent et le nouvel en-tête sont transmis. Pour cela, chaque modification est notifié par un drapeau d'un octet. Les quatre premiers bits permettent de fournir le décalage en octets entre le dernier octet du précédent champ modifié et le début du nouveau champ modifié. Les quatre derniers bits définissent la longueur de la modification en octets. La fin de l'en-tête est signalée par un drapeau nul. Cette technique de compression permet de réduire, dans le cas général, la taille moyenne de l'en-tête TCP/IP à 17 octets. Par rapport à ce mécanisme, Thinwire-II permet de définir des classes types. Le principe consiste à définir des "squelettes" de type de modifications. Dans la plupart des cas, les différences, et donc les modifications entre paquets, sont sensiblement identiques d'un paquet à l'autre. En définissant des classes types, Thinwire-II permet de s'affranchir de mécanismes de drapeaux et permet une compression jusqu'à 13 octets.

Compressed TCP, ou CTCP en abrégé, est un mécanisme de compression d'en-tête standardisé [37] défini par Van Jacobson en 1990. On peut également trouver cette algorithme sous le nom VJ, en référence à son auteur. Comme son nom l'indique, ce mécanisme se charge de la compression des en-têtes TCP/IP, en IPv4 seulement. Ce mécanisme de compression permet d'obtenir une réduction de l'en-tête jusqu'à 3

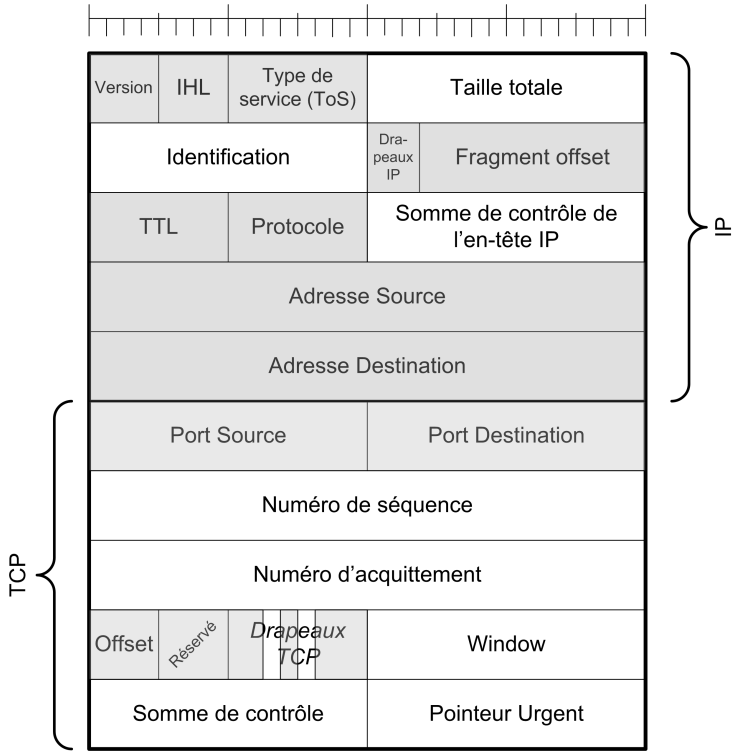

Figure 2.10 – Schéma général de l'en-tête TCP/IPv4

octets.

Le premier point de CTCP est, tout comme Thinwire, de s'affranchir des champs considérés comme invariants ou quasi-invariants lors d'une même connexion TCP. Ces champs sont représentés en grisé sur la Figure 2.10. On considère ici que ces champs sont conservés en mémoire par le mécanisme de compression du côté de l'expéditeur, et par le décompresseur du destinataire. Ce premier mécanisme permet de réduire de moitié l'ensemble de l'en-tête à transmettre. Etant donné que le quadruplet {Adresse source, Port source, Adresse destination, Port destination} n'est plus disponible dans l'en-tête compressé, CTCP définit un mécanisme d'identifiant de connexion (CID) codé sur un octet permettant aux nœuds intermédiaires de pouvoir faire transiter correctement ces en-têtes. Nous voyons ainsi apparaître la nécessité d'avoir une prise en charge de CTCP par tous les nœuds intermédiaires.

CTCP considère que la couche liaison sous-jacente est capable de fournir la taille des paquets reçus, ce qui rend inutile la transmission du champ *Taille Totale*. La *somme de contrôle de l'entête IP* ne protégeant plus que le numéro d'identification, celle-ci est aussi inutile et également supprimée. Elle sera recréée par la suite au niveau du récepteur à des fin de compatibilité. A ce niveau là, l'étude des champs invariants et caduques ramène la taille de l'en-tête TCP/IP à transmettre à 16 octets.

Si les champs restants ne sont pas considérés comme invariants, ils ne varient pas

néanmoins tous en même temps. CTCP transmet uniquement les champs variants d'un en-tête à l'autre. Afin de permettre au décodeur de savoir si telle ou telle modification de champ est présente, chaque champ est associé avec un drapeau d'un bit permettant de marquer sa présence ou son absence. Ce mécanisme permet de ramener la taille moyenne nécessaire à une dizaine d'octets.

Enfin, le dernier mécanisme mis en œuvre dans CTCP concerne l'encodage dit *delta encoding*. Comme son nom l'indique, l'encodage delta consiste à n'envoyer que la différence de valeur entre un champ et le précédent. Ceci est particulièrement utile pour les champs de 2 ou 4 octets, qui évoluent de façon incrémentale (comme généralement les numéros d'identification). En effet, si la modification est comprise entre 1 et 255, ce champ ne sera codé que sur un seul octet. Le schéma général de l'en-tête compressé CTCP se trouve sur la Figure 2.11.

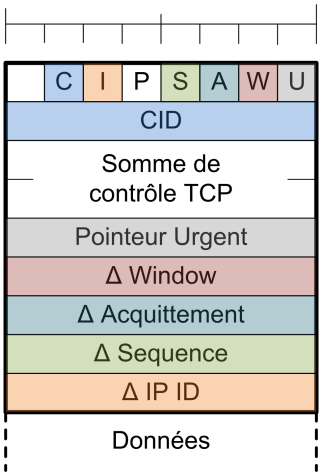

Figure 2.11 – Schéma général de l'en-tête CTCP

Sur le premier octet, nous trouvons donc les drapeaux permettant d'identifier la présence du champ correspondant dans la suite de l'en-tête. Le bit C permet lui aussi d'annoncer l'absence ou la présence du CID, qui peut être omis, comme par exemple dans le cas de connexions directes. Le bit P est une transposition direct du bit Push de TCP. Il est possible de transmettre des deltas négatifs ou supérieurs à 255 avec ce mécanisme. Dans ce cas, le premier octet du champ est codé à 0. Un delta de 0 n'ayant aucun sens ici car il serait inutile, le delta est ainsi codé sur les deux octets suivants.

Comme nous pouvons le voir, le premier bit de l'en-tête CTCP est inutilisé. En dehors du standard, il est cependant courant de voir ce bit utilisé pour la notification de congestion (ECN) [38]. Il est aussi possible de trouver un drapeau à cette position, celui-ci permettant la définition d'extensions supplémentaires à TCP dans l'octet suivant.

Dans le cas général, la taille moyenne de l'en-tête CTCP est d'environ 5 octets avec ce mécanisme, et peut même atteindre 3 octets. Dans ce cas là, l'en-tête est

réduit à l'octet contenant les drapeaux et les deux octets de la somme de contrôle de TCP.

CTCP/VJ est donc un mécanisme de compression d'en-têtes TCP/IP qui a l'avantage d'avoir une mise en œuvre très simple, tout en présentant de bonnes performances de compression. Il présente également l'avantage d'être le plus transparent possible pour les couches adjacentes. Cette compression n'offre cependant aucune fiabilité quant à la perte de paquets. En effet, le codage delta devient inutilisable dès lors que le paquet précédent, et donc son entête, est perdu.

2.3.3 IPHC/CRTP/eCRTP

Afin de combler certaines carences de CTCP, dues en majorité à l'évolution des technologies et des besoins (IPv6, RTP, ...), deux nouveaux protocoles de compression ont été définis. Le premier, IP Header Compression (IPHC) [39] se concentre sur la compression TCP/IP que ce soit en IPv4 ou en IPv6. De manière analogue et conjointe, le second mécanisme Compressed RTP [40], se charge lui des compressions RTP/UDP/IP et UDP/IP. Les deux protocoles partagent de nombreuses similitudes. enhanced CRTP (eCRTP) [41] est une extension à CRTP permettant une meilleure prise en charge des réseaux à long RTT et à réordonnancement. Ces protocoles introduisent les premiers mécanismes de fiabilisation suite à la compression d'en-têtes.

2.3.3.1 IPHC

IPHC est un protocole de compression permettant une compression des en-têtes TCP/IP jusqu'à une taille moyenne de 4 à 7 octets. Le protocole permet par ailleurs une compression seule de l'en-tête IP. Ces deux schémas sont d'ailleurs gérés en IPv4, mais aussi en IPv6. Tout comme CRTP, IPHC est un protocole de compression point-à-point ; un paquet compressé n'est donc pas transporté de bout en bout sans modifications. Celui-ci nécessite donc une régénération complète de la compression au niveau de chaque nœud réseau, mais garantit une transparence complète vis-à-vis de ce même réseau, si toute la chaîne n'est pas compatible, ou ne souhaite pas activer cette compression. Au niveau du codage, IPHC utilise un mécanisme de codage différentiel similaire à CTCP, tout en ajoutant deux mécanismes de détection/reprise sur erreur, TWICE et *Header Request* que nous détaillerons par la suite.

L'identification des paquets IPHC se fait par deux paramètres. Le premier, le CID est similaire au mécanisme d'identification de connexion de CTCP/VJ. Le second paramètre est appelé *generation*, et permet d'identifier chaque paquet IPHC d'une même connexion et d'un même contexte par un numéro de séquence. Celui-ci n'est pas utilisé pour la compression TCP/IP, la couche TCP permettant une identification grâce à son numéro de séquence propre, mais pour la compression des en-têtes IP seuls. Cette numérotation, comme nous le verrons par la suite, permet de mettre en

place les mécanismes de détection et de reprise sur erreur.

Le protocole IPHC se base sur quatre types de paquets :

- Les paquets FULL_HEADER permettent de transporter l'en-tête non compressé. De manière similaire à CTCP, le champ Packet Length d'IP étant fourni par la couche liaison, celui-ci sert ici au transport du CID et éventuellement du *generation*. Ce type de paquet est utilisé pour un établissement ou un rafraîchissement du contexte effectué de manière régulière afin d'éviter la propagation d'erreurs.
- Les paquets COMPRESSED_NON_TCP qui se chargent donc de la compression de l'en-tête IP seul. Dans ce cas, l'en-tête IP est remplacé par deux octets permettant la transmission du CID et du *generation*.
- Les paquets COMPRESSED_TCP qui sont donc les paquets les plus courants dans ce mécanisme. Ils permettent le transport de l'en-tête compressé TCP/IP à l'aide d'un codage différentiel, ainsi que l'identifiant de contexte. Le schéma de cet en-tête est très similaire à celui de CTCP, le bit inutilisé étant remplacé par R, drapeau permettant la définition d'extensions supplémentaires. Le bit C de CTCP est ici inutile car le CID est toujours transmis.
- Les paquets COMPRESSED_TCP_NODELTA qui permettent l'envoi de l'en-tête TCP non compressé sans les numéros de port. En accolant le CID, la taille de cet en-tête est de 17 octets. Ce type de paquet ne peut être envoyé qu'en réponse au mécanisme de *Header Request* détaillé par la suite.

L'une des principales nouveautés des protocoles IPHC et CRTP est la mise en place de mécanismes de fiabilisation. Deux mesures principales ont été mis en oeuvre afin de lutter contre les défauts inhérents au codage différentiel.

Dans un protocole plus classique, tel CTCP, lorsqu'un en-tête est perdu, le décompresseur est incapable de récupérer les paquets suivants, pourtant reçus, jusqu'à réception d'un en-tête non compressé qui contient le nouveau contexte. Dans le but d'éviter ce genre de désynchronisation, IPHC et CRTP implémentent un mécanisme appelé TWICE. L'idée de l'algorithme TWICE est de considérer que lorsque un en-tête est reçu, si la somme de contrôle de TCP échoue, alors cet échec est imputable à une perte de paquets. Dans ce cas, le décompresseur va considérer que le ou les paquets perdus apportaient le même codage différentiel que celui reçu, et va appliquer ce différentiel une ou plusieurs fois.

Lorsque le mécanisme TWICE échoue, IPHC permet de basculer sur le second mécanisme de reprise sur erreur : *Header Request*. Le décompresseur a la possibilité d'envoyer un paquet nommé CONTEXT_STATE. Dans ce paquet, le décompresseur précise le ou les CID pour lequel le contexte est invalidé. Lorsque le compresseur reçoit cette notification, il utilise alors un paquet COMPRESSED_TCP_NODELTA qui contient l'intégralité de l'en-tête TCP et permet donc de resynchroniser le récepteur. Par la suite, IPHC utilise un mécanisme de reprise sur erreur de type Slow Start, bien connu de TCP. Initialement chaque en-tête non compressé est séparé par l'envoi d'un

en-tête compressé. Cette séparation augmente exponentiellement jusqu'à atteindre une valeur limite. Lorsque une perte est détectée, le compresseur revient à son état initial.

Enfin, la dernière problématique majeure reste la gestion du réordonnancement. Dans le cadre de la transmission d'en-têtes TCP/IP et à cause du codage différentiel, le décompresseur n'a aucun moyen de gérer le réordonnancement. Deux solutions sont alors possibles. La première solution consiste à n'utiliser que des paquets de type COMPRESSED_TCP_NODELTA, mais celle-ci n'est pas satisfaisante car elle nécessite l'envoi d'en-têtes de 17 octets, et n'offre ainsi aucun avantage sur CTCP par exemple. Une seconde solution consiste à envoyer les numéros de séquence TCP en valeur absolue, au coût de 2 octets supplémentaires. Cette solution est proposée à titre informative dans le standard.

2.3.3.2 CRTP

Parrallèlement à la compression TCP/IP, un protocole de compression compatible a été défini pour la compression des en-têtes RTP/UDP/IP et UDP/IP : Compressed RTP (CRTP). Ainsi, CRTP partage de nombreux mécanismes communs avec IPHC. Avec la croissance des besoins multimédias, l'utilisation des protocoles RTP et UDP est devenue cruciale. Dans le cadre d'envoi de paquets de petite taille, typiquement des paquets audio, la taille de l'en-tête RTP/UDP/IP peut être similaire à celle des données transportées. Il est donc important de pouvoir définir un schéma de compression d'en-tête, ceux-ci pouvant représenter jusqu'à 50% du trafic envoyé.

Real-Time Protocol ou RTP est un protocole [42] qui se situe au dessus de la couche UDP et qui apporte une gestion particulièrement adaptée au transport de données sensibles au délai et à la gigue. Sa taille est de 12 octets. UDP quant à lui est un protocole de transport en mode non connecté, qui n'apporte ainsi aucun mécanisme de qualité de service. Ceci est cependant un avantage pour RTP et le transport de données temps-réel qui ne nécessitent pas ce type de contraintes. L'en-tête du protocole UDP est donc réduit à 8 octets, contenant les ports source et destination, la longueur du paquet ainsi que la somme de contrôle de l'en-tête.

La taille totale de l'en-tête RTP/UDP est de 20 octets, identique à celle de l'en-tête TCP. Au niveau de l'en-tête RTP, le mécanisme de codage différentiel est adapté. En effet, la plupart des champs RTP évoluent d'un delta constant d'un paquet à l'autre. L'idée est ici de coder le changement de delta, ou la différence du second ordre en d'autres termes. Dans ce cas, lorsque le décompresseur reçoit une nouvelle valeur représentant un champ, il modifie le delta à appliquer (sauf pour le numéro de séquence RTP) et non pas la valeur du champ directement. Ce mécanisme sous-entend bien entendu, que l'émetteur et le récepteur conservent en mémoire les deltas correspondants en plus des paramètres de la connexion. Par rapport à IPHC, il est important de noter que ce codage différentiel du second ordre s'applique aussi au champ IP ID de l'en-tête IP.

Au niveau du contexte, CRTP considère que le contexte est défini par les adresses et ports source et destination ainsi que l'identifiant de source SSRC de RTP. De ma-

nière similaire aux mécanismes précédemment étudiés, chaque contexte est associé à un identifiant de connexion CID. Au niveau de l'en-tête UDP/IP, seul le champ IP ID est ainsi transmis. Quant à l'en-tête RTP, le champ SSRC étant identifié par un CID unique, seuls les champs M, le numéro de séquence et le timestamp sont transmis. Les bits P, X, et le champ type de données sont considérés comme suffisamment constants pour ne pas avoir à être transmis. Contrairement aux autres champs, seuls les 4 derniers bits du numéro de séquence sont transmis, et ce en valeur absolue permettant une détection rudimentaire de pertes.

Tout comme IPHC, CRTP définit quatre types de paquets à transmettre :

- Les paquets de type FULL_HEADER sont homogènes avec leurs homologues d'IPHC. Ceux-ci transportent le CID et le numéro de séquence sur 4 bits, en utilisant, ici aussi, les champs Packet length des en-têtes UDP et IP.
- Les paquets de type COMPRESSED_NON_TCP identiques à ceux d'IPHC.

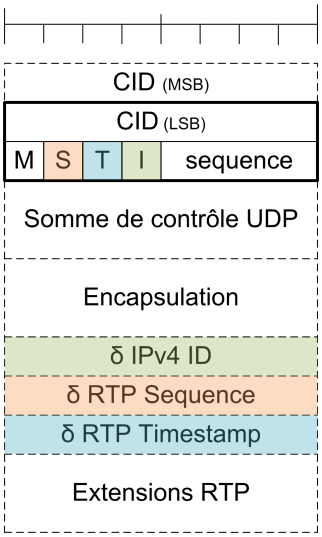

Figure 2.12 – Schéma général de l'en-tête COMPRESSED_RTP

- Les paquets COMPRESSED_RTP qui utilisent le codage différentiel du second ordre. Comme nous pouvons le remarquer sur la Figure 2.12, l'en-tête est alors réduit à 2 octets au minimum, comprenant le CID, le numéro de séquence et les drapeaux marquant la présence des deltas du second ordre dans la suite de l'en-tête.
- La compression de l'en-tête UDP/IP seul entraîne l'envoi d'un paquet de type COMPRESSED_UDP. Ce type de paquet est également utilisé lorsque l'un des champs considéré comme constant de RTP, est modifié. La partie UDP/IP compressée peut être considérée comme un cas particulier des paquets COMPRESSED_RTP (Figure 2.13).

Les mécanismes de détection et de reprise sur erreur sont similaires à ceux d'IPHC.

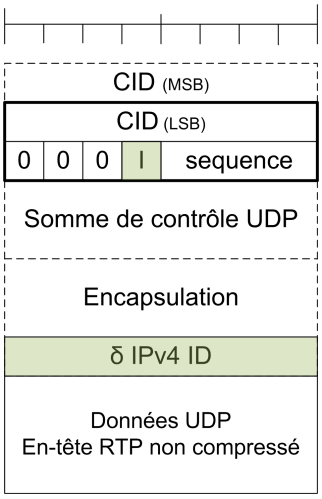

Figure 2.13 – Schéma général de l'en-tête COMPRESSED_UDP

Cependant, le numéro de séquence n'est ici pas codé en différentiel, mais en valeur absolue sur ses 4 derniers bits. Ce mécanisme semble donc permettre une détection plus pertinente qu'IPHC, sauf dans le cas où le nombre de paquets perdus est un multiple de 16. Dans ce cas, la détection des pertes se fera sur la somme de contrôle UDP si elle est présente. Dans le cas contraire, une propagation d'erreurs conséquente peut apparaître.

2.3.3.3 eCRTP

Afin d'éviter ce genre de comportements, mais pour aussi s'adapter à des transmissions multimédia sur des réseaux à long délais, où la retransmission est inutile, l'IETF a proposé une évolution de CRTP, nommée enhanced CRTP (eCRTP). Le mécanisme de récupération sur erreur n'étant pas modifié, le gain en fiabilité s'effectue grâce à une redondance inter-paquets, permettant de limiter la perte de synchronisation. Cette redondance s'accompagne naturellement d'une compression moindre. Cette redondance s'articule en trois points : un envoi multiple des paramètres du contexte, l'envoi possible de paramètres en valeurs absolues et enfin l'utilisation de paquets COMPRESSED_UDP comme type de paquet par défaut.

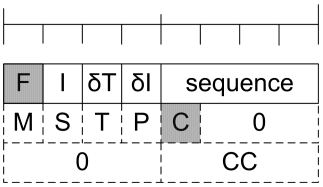

Figure 2.14 – Octet de drapeaux eCRTP

Afin de permettre une plus grande souplesse au niveau de la transmission des paramètres, eCRTP se base sur le schéma d'en-tête compressé COMPRESSED_UDP. Pour des raisons de compréhension, les drapeaux marquants des différentiels du second ordre seront représentés avec un δ. Sur la Figure 2.14, les 3 premiers bits inutilisés sont remplacés par F, I et δt. Le bit F définit la présence d'un second octet de drapeaux. I identifie la présence de l'IP ID d'IPv4 en valeur absolue, et δI, son nouveau différentiel. δt permet quant à lui de donner le nouveau différentiel du timestamp RTP. Sur le second octet de drapeaux, nous retrouvons les bits M, S, T et P utilisés de la même manière que pour les paquets COMPRESSED_RTP de CRTP. Le bit C quant à lui définit la présence d'un troisième octet dont les quatre derniers bits correspondront au nombre de sources contribuantes de RTP (CSRC).

Au niveau de la fiabilisation, eCRTP nécessite la mise en place d'un mécanisme de validation. Celui-ci peut être fourni par la somme de contrôle UDP, mais celle-ci n'est pas obligatoire en IPv4. Afin de pallier ce manque, eCRTP propose d'utiliser une somme contrôlant l'en-tête nommée Header Checksum ou HDRCKSUM. Celui-ci est géré par le protocole de compression. Il est donc transparent pour les couches supérieures, et son utilisation est signalée via un drapeau dans les champs Packet Length.

Afin d'éviter des invalidations de contexte qui entraînent une perte de synchronisation, eCRTP propose un mécanisme de renvoi multiple. Dans la version classique, la perte d'un seul en-tête peut entraîner une perte de synchronisation, si celui-ci apportait un changement de delta irrécupérable par l'algorithme TWICE. eCRTP propose de répéter chacune des modifications de delta un certain nombre de fois. En répétant chaque mise à jour N fois, eCRTP permet de supporter N pertes successives. A titre d'exemple, un émetteur émet un paquet RTP toutes les 10ms pendant 100ms. Puis cet émetteur attend 200ms avant de renvoyer des paquets tous les 10ms. Avec le codage différentiel de CRTP, si le récepteur ne reçoit pas le premier paquet émis après les 200ms, l'algorithme TWICE échoue car le delta du timestamp a été modifié. Avec eCRTP, non seulement le timestamp peut être envoyé en valeur absolue, mais ceci sur N paquets consécutifs, et permet donc de supporter la perte de ce paquet. Cependant, il est important de noter que ces améliorations de la fiabilisation se font au détriment du taux de compression. Il faut donc tenir compte à ne pas utiliser un N surdimensionné et à n'envoyer les champs en valeur absolue que lorsque ceci est nécessaire, afin de ne pas rendre la compression d'en-têtes eCRTP contre-productive.

eCRTP apporte donc des améliorations concernant la fiabilisation des en-têtes RTP/UDP/IP. L'ensemble IPHC/eCRTP permet donc une compression efficace des principaux protocoles réseaux à savoir TCP, UDP, RTP ainsi qu'IPv4 et IPv6. Grâce à l'utilisation d'algorithme de récupération, ces protocoles permettent d'éliminer en partie les conséquences de la compression sur la perte de paquets et les pertes de synchronisation successives. Cependant, du point de vue la fiabilisation, ces protocoles ne sont pas optimaux car, afin d'être très efficaces, ceux-ci nécessitent de sacrifier

une partie importante du taux de compression.

2.3.4 ROHC

RObust Header Compression (ROHC) est une plateforme pour la compression d'en-têtes standardisé par l'IETF [43]. Le principe de ce standard est de proposer une plateforme commune de compression à laquelle se greffent des profils particuliers à chaque protocole. Ainsi, le but est de standardiser le fonctionnement de l'émetteur et du récepteur, tout en conservant les particularités de chaque protocole. Dans la première version du standard, ROHC ne gère que les profils RTP/UDP/IP, UDP/IP et ESP/IP, en IPv4 et IPv6. Par la suite, la compression d'IP seule [44], d'UDP lite [45] et de TCP [46] ont été standardisées.

ROHC permet donc l'implémentation au fur et à mesure de profils de protocoles, qui suivent les mêmes règles de fonctionnement, et autorise donc une grande modularité. Cependant cette modularité se traduit par des mécanismes de fonctionnement complexes et en constante évolution comme en témoigne le nombre de standards mettant à jour la définition du framework ROHC ou de ses profils [47] [48] [49] [50].

Dans cette étude, par souci de concision, nous présenterons le mécanisme général de fonctionnement de ROHC, que nous illustrerons avec le profil RTP/UDP/IP.

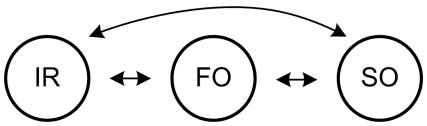

Figure 2.15 – Machine à états du compresseur ROHC

Le protocole ROHC repose sur l'utilisation de machines à états finis régissant le fonctionnement du compresseur et du décompresseur. Comme schématisé sur la Figure 2.15, le compresseur original fonctionne sur une machine à trois états :

- L'état *Initialisation & Refresh* (IR) dans lequel le compresseur envoie des en-têtes non compressés, permettant d'initialiser une connexion ou de mettre en place une récupération suite à une désynchronisation totale.
- L'état *First Order* (FO), qui correspond, comme son nom l'indique, à un premier niveau de compression dans lequel les champs dits statiques (adresses, ports,...) ne sont pas transmis, et les autres champs sont compressés partiellement. ce type de compression est utilisé pour de la reprise sur erreur n'incluant pas une perte totale du contexte.
- L'état *Second Order* (SO) qui est l'état général du compresseur dans lequel la compression est maximale.

De manière analogue, le décompresseur fonctionne aussi sur une machine à trois états, telle que représentée sur la Figure 2.16 :

- L'état *No Context* (NC) qui correspond à l'état dans lequel le décompresseur

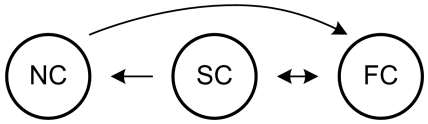

Figure 2.16 – Machine à états du décompresseur ROHC

n'a aucun contexte que ce soit les champs statiques ou dynamiques. C'est l'état initial du compresseur. Dans cet état le décompresseur attend des en-têtes non compressés.
- L'état *Static Context* (SC) dans lequel le décompresseur possède le contexte statique.
- L'état *Full Context* (FC) où le décompresseur connaît le contexte en totalité.

Même si le protocole ROHC fait en sorte que le compresseur s'adapte à l'état du décompresseur, les deux machines à états sont, dans l'absolu, décorellées.

Les transitions de la machine à états du compresseur sont dépendantes du fonctionnement souhaité. En effet, ROHC permet l'utilisation de la compression de manière pro-active ou réactive, selon que la voie de retour est disponible et/ou désirée. Pour cela, ROHC définit trois modes de fonctionnement.

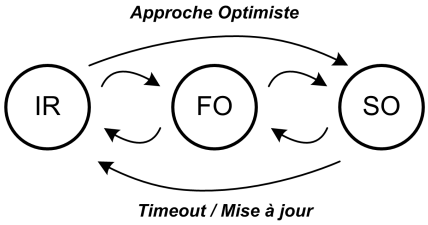

Figure 2.17 – Machine à états du compresseur en mode Unidirectionnel

Le premier mode est dit Unidirectionnel (Unidirectional, U) et n'utilise pas la voie de retour. Il est donc parfaitement adapté dans le cadre d'une fiabilisation pro-active, lorsque la voie de retour est indisponible (satellite,...) ou inapplicable de manière simple (multicast,...). Comme défini sur la Figure 2.17, les transitions entre les différents états du compresseur gérées par des timers.

Dans ce contexte, toute la difficulté est d'estimer judicieusement ces timers. En effet, dans ce mode, lorsque le décompresseur est désynchronisé, celui-ci doit attendre l'arrivée d'en-têtes non compressés afin de retrouver le contexte. Pendant ce temps, tous les paquets reçus sont donc éliminés. En conséquence, les timers régissant le comportement du compresseur doivent être suffisamment courts pour éviter des temps trop longs de désynchronisation, mais aussi, suffisamment longs pour que la compression moyenne soit efficace.

Ce mode de fonctionnement se base donc sur une connaissance à priori du canal.

Cependant, le mode U laisse la possibilité au décompresseur d'envoyer des acquittements positifs. Ceux-ci sont optionnels, mais permettent ainsi au compresseur de juger de la qualité du lien, et de modifier éventuellement en conséquence la durée des timers de retour à l'état IR.

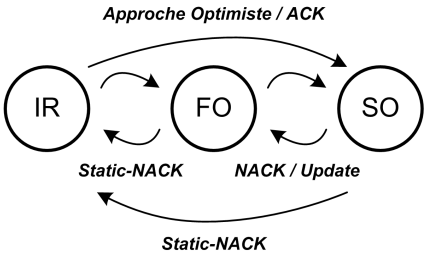

Figure 2.18 – Machine à états du compresseur en mode Optimiste

Le mode Optimiste (Optimistic, O) est un mode hybride entre le mode U et le mode Fiable défini par la suite. Dans ce mode, le compresseur se base à la fois sur des timers et sur les acquittements du décompresseur. Dans ce mode, l'utilisation de la voie retour est nécessaire. Les transitions vers les niveaux de compression supérieurs se font soit par le déclenchement d'un timer, soit par la réception d'un acquittement positif optionnel. De manière analogue, les transitions vers les niveaux de compression inférieurs se font par des timers ou sur réception d'un acquittement négatif, ceux-ci étant, par contre, obligatoires (Figure 2.18). Lorsque le compresseur reçoit un acquittement négatif de type STATIC-NACK, définissant la perte totale du contexte, il transite vers l'état IR, et s'il reçoit simplement un acquittement NACK, il transite vers l'état FO.

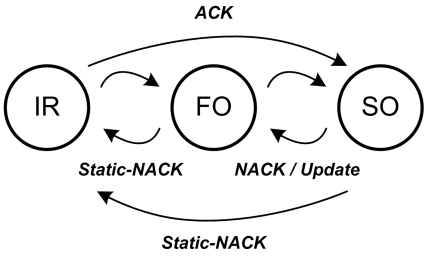

Figure 2.19 – Machine à états du compresseur en mode Fiable

Le mode Fiable (Reliable, R) quant à lui se base uniquement sur les informations de la voie retour. Les transitions montantes se font uniquement sur réception d'un acquittement positif, et les transitions descendantes à la réception d'un acquittement négatif (Figure 2.19).

Suivant le type d'application et de réseau visé, chaque mode sera plus ou moins adapté. En effet, dans le cadre de communication type réseau local, à haut débit,

voie de retour et faible latence, un mode basé sur les acquittements sera plus efficace en termes de compression qu'un mode unidirectionnel. Cependant, si le réseau utilisé comporte une forte latence, ces mécanismes seront pénalisés au profit d'un mode unidirectionnel, indépendant de la latence du réseau.

Des mécanismes permettant le basculement dynamique d'un mode de compression à un autre sont également définis dans le standard, mais ne seront pas détaillés ici.

Tout comme les mécanismes de compression précédemment étudiés, ROHC définit 5 types de champs, selon leur variabilité :

- Les champs INFERRED non communiqués car déductibles d'autres paramètres.
- Les champs STATIC qui ne varient pas pour un flux de paquet. Lorsque un de ces champs varie, un paquet IR est envoyé.
- Les champs STATIC-DEF similaires, qui permettent l'identification d'un flux de paquets (comme les adresses).
- Les champs STATIC-KNOWN qui sont des valeurs connues car caractéristiques d'un protocole
- Les champs CHANGING considérés comme variables.

Par exemple, dans le cas de RTP/UDP/IP, la répartition de ces différents champs est présentée dans la Table 2.1 :

	IPv6 (octets)	IPv4 (octets)
INFERRED	4	6
STATIC	1.75	1.875
STATIC-DEF	42.5	16
STATIC-KNOWN	0.25	2.625
CHANGING	11.5	13.5
Total	60	40

Table 2.1 – Répartition des champs RTP/UDP/IP dans ROHC

Ainsi, pour le protocole ROHC, seuls 34% de l'en-tête RTP/UDP/IP en IPv4 et 19% en IPv6 sont considérés comme variables. Tout comme les protocoles précédents, ROHC permet de discriminer parmi les champs CHANGING différents types de champs variables, notamment les champs variant d'un delta quasi-constant. Ces mécanismes permettent ainsi d'obtenir des performances de compression similaires aux autres protocoles.

Lorsque le décompresseur détecte une erreur résiduelle dans le paquet courant reçu, celui-ci rejette simplement le paquet sans envoyer d'acquittement négatif (dans les modes O et R) ou changer d'état. Le décompresseur considère alors l'échec du CRC comme une erreur binaire dans le paquet reçu. Conformément à la machine à états du décompresseur définie sur la Figure 2.16, lorsque k_1 des n_1 derniers paquets ont vu leur CRC échouer, celui-ci passe de l'état *Full Context* à l'état *Static Context*. Dans ce cas, le décompresseur rejette tous les paquets jusqu'à réception d'un paquet

transportant un contexte dynamique entier. De la même manière, si le décompresseur se trouve dans l'état *Static Context* et si k_2 des n_2 derniers paquets transportant un contexte dynamique entier échouent, le décompresseur se met dans l'état *No Context*. Dans l'état *No Context*, le décompresseur élimine tous les paquets jusqu'à réception d'un en-tête complet.

Un des points important du décompresseur ROHC est donc une bonne paramétrisation des valeurs k et n. En effet, si des valeurs trop sensibles sont utilisées, par exemple $k_1 = n_1 = k_2 = n_2 = 1$, le décompresseur réagira de manière excessive à toute erreur, et entrainera une perte conséquente de paquets suite à ce changement d'état soudain. Ceci sera d'autant plus marqué que les timeouts sont importants dans les modes U et O et le RTT long dans les modes O et R. De même, un manque de sensiblité entraînera un manque de réaction et une longue invalidation en conséquence.

Comme mécanisme de fiabilisation, ROHC propose une amélioration du mécanisme de type LSB. Dans un mécanisme de type LSB, lorsque une modification est à transmettre, on envoie les k bits ayant été modifiés, ou alors la plus petite valeur supérieure à k lorsque l'ensemble des tailles permises est limité, par exemple des champs sur un ou deux octets. Le fonctionnement de ce mécanisme est garanti lorsque le compresseur et le décompresseur utilisent la même valeur de référence. Cependant, deux scénarios peuvent empêcher le fonctionnement correct de ce codage. Dans un premier cas, des pertes de paquets peuvent apparaître. Dans ce cas, au prochain paquet reçu par le décompresseur, celui-ci et le compresseur n'ont plus forcément la même valeur de référence pour le codage LSB. Il n'est donc pas garanti ici que le décodage fonctionne ce qui entraîne un perte de synchronisation. Dans un second cas, des erreurs bits résiduelles peuvent avoir modifié le LSB porté par un paquet. En conséquence, les prochains paquets reçus ne se baseront pas sur la même valeur de référence que le compresseur et entraineront une propagation de l'erreur et une perte de synchronisation.

Afin d'éliminer les erreurs d'interprétations dues à des valeurs différentes de référence de chaque côté, ROHC utilise un schéma appelé Window-LSB (W-LSB). Avec ce mécanisme, le compresseur conserve en mémoire une fenêtre glissante de valeurs de référence. Lors de l'envoi d'un paquet original, le codage W-LSB va coder sur le nombre minimal $k^{'}$ de bits tels que l'interprétation de la valeur originale soit unique sur l'intervalle couvert par l'ensemble des valeurs de référence de la fenêtre. Ceci permet d'assurer que la décompression est correcte si tant est que le décompresseur a reçu une des valeurs de référence de la fenêtre. Ceci permet donc de garantir de manière efficace la transmission des valeurs, au prix d'une perte de compression mesurée.

Le mécanisme W-LSB est illustré sur l'exemple suivant. Prenons l'exemple de la transmission des valeurs : 124, 252, 276, 278 et supposons que la valeur 252 soit perdue lors de la transmission. Avec un codage de type LSB, le compresseur envoie les valeurs : 124, 252, 76, 8. Le codage a donc coûté ici 9 caractères. Cependant, le décompresseur décodera : 124, 176, 178. Avec un codage de type W-LSB et une taille de fenêtre de deux, le codeur enverra 124, 252, 276, 78 soit 11 caractères mais

ici le décodage des successeurs de 252 est garanti.

En plus du mécanisme W-LSB, ROHC permet un encodage particulier adapté au timestamp RTP. Celui-ci étant généralement incrémenté par un multiple d'une valeur fixée d'après les expérimentations, ROHC permet un mécanisme de compression où le compresseur envoie la valeur utilisée au décompresseur, puis envoie les valeurs successives en connaissance de cette valeur et n'a plus qu'à envoyer le multiple au décompresseur.

ROHC est une plateforme de compression évoluée permettant de compression un large panel de protocoles. Le principal intérêt de ce protocole est de se présenter comme un mécanisme indépendant de toute couche supérieure en utilisant des machines à états régissant l'émetteur et le récepteur. Ceci permet l'utilisation de modes adaptés à une transmission unidirectionnelle ou de multi-diffusion (modes U et O), mais aussi bi-directionnelle (modes O et R). Cependant, le principal point soulevé par ces modes est leur efficacité respective lorsque la voie de retour est utilisable. En effet, si le mode Fiable semble être le plus adapté à une communication bi-directionnelle, celui-ci subit intrinsèquement les délais de transmission induits par le canal. Si ce RTT (Round-Trip Time) est élevé, la détection de l'erreur sera longue et tous les paquets reçus dans cet intervalle seront perdus.

La fiabilisation de la compression ROHC est dépendante du mode utilisé et du canal de transmission. Le codage W-LSB quant à lui permet une transmission robuste des valeurs à transmettre. L'étude de la fiabilisation de ROHC sera donc portée ici sur la paramétrisation de ROHC dans différents modes et sur différents scénarios de canaux.

Plusieurs travaux ont visé à une modélisation du protocole ROHC sous certaines conditions. Les travaux de Cho et al. [51] [52] ont permis par exemple de définir un modèle de canal source pour l'étude des mécanismes de ROHC et plus particulièrement de l'intérêt de l'encodage W-LSB. Ces travaux se basent également sur un canal de type Gilbert-Elliott. De manière parallèle, les travaux de Minaburo et al. [53] [54] ont permis l'élaboration d'un modèle mathématique sur un canal à erreurs, du mode Unidirectionnel de ROHC. La complexité inhérente du protocole ROHC ne permet pas d'ailleurs une modélisation complète de tous ses mécanismes pour l'ensemble des modes de fonctionnement. Il semble donc nécessaire de procéder à une restriction de cette modélisation mathématique à des scénarios ou des mécanismes plus précis.

Chapitre 3

Modélisation d'un protocole de compression d'en-têtes

Sommaire

3.1 Introduction

Dans ce chapitre, nous présentons une modélisation mathématique d'un protocole de compression d'en-têtes protocolaire. Comme nous l'avons vu précédemment, les travaux dans ce domaine ont permis la création de mécanismes sophistiqués que ce soit avec ou sans voie de retour. Cependant, leur relative complexité rend leur étude délicate. En effet, l'évaluation de leur performance la plus simple consiste probablement à les implémenter complètement. Nous nous proposons donc de modéliser le comportement de tels mécanismes sur un canal de type Gilbert-Elliott. Des études ont déjà été menées sur la robustesse intrinsèque des mécanismes de protocole existants, mais dans ce chapitre, nous essaierons de répondre à quelques questions concernant l'utilisation de tels protocoles.

Si ces protocoles permettent, comme nous l'avons vu, d'économiser jusqu'à 40 octets par paquet, ceci s'accompagne nécessairement d'une perte de robustesse suite à ces mêmes pertes de paquets. En effet, même si elle peut s'avérer coûteuse, l'information supprimée par le protocole de compression agit, dans le cas normal, comme un mécanisme de redondance. Cette redondance est certes non optimale, mais elle

possède le mérite de pouvoir détecter et corriger rapidement un contexte non reçu, ce qui est plus délicat dès que la compression est activée. Nous pouvons comparer cette problématique à celle du codage vidéo où, comme dans le cas de H.264 par exemple, la perte de certaines *frames* peut être dommageable pour les frames suivantes.

Dans le domaine de la compression protocolaire, il n'existe pas à ce jour de mécanismes de compression permettant de favoriser un en-tête plus important qu'un autre (sauf dans le cas d'eCRTP dans l'absolu). Par ailleurs, une idée reçue sur la compression protocolaire, est d'utiliser les mécanismes de voie de retour (modes O et U de ROHC par exemple) dès lors qu'ils sont disponibles. Si ces modes permettent effectivement une compression plus efficace, nous étudierons alors les conséquences d'un tel mécanisme sur la métrique du temps moyen de déconnexion, et nous verrons l'influence certaine du champ d'application sur la pertinence d'un mécanisme de retour.

3.2 La couche protocolaire et le canal de Gilbert-Elliott

Dans le modèle d'Interconnexion de Systèmes Ouverts (OSI) classique, le protocole de compression d'en-têtes est situé entre la couche liaison et la couche réseau ou plus généralement, en remplacement de la couche réseau et de la couche transport (selon les protocoles compressés). D'une manière schématique, comme présenté sur la Figure 3.1, la couche de compression protocolaire (HC) reçoit directement des trames de la couche liaison, et elle transmet directement à la couche applicative les données dépourvues des en-têtes qu'elle aura décodé.

Comme nous l'avons évoqué rapidement dans le chapitre précédent, dans tous les mécanismes de liaison récents (Ethernet,...), lorsque que le CRC échoue, l'ensemble de la trame est généralement éliminé. Tout ce qui est transmis à la couche adjacente haute est donc considéré comme valide. Du point de vue de cette couche, et donc dans notre cas, du protocole de compression, le canal de la couche liaison peut être vu comme un canal à effacements, où tout ce qui est reçu est correct et auquel est associé un taux de perte de trames : *Frame Error Rate* (FER). Ces pertes affectent alors de manière uniforme, l'en-tête de la compression et les données transportées.

Nous avons également vu que suite à ces pertes reçues, de nouvelles pertes peuvent apparaître au niveau de la décompression, celles-ci étant imputables aux mécanismes de compression. Nous avons vu qu'ainsi, lorsque la décompression échoue, l'ensemble du paquet est inutilisable n'est donc pas transmis à la couche applicative.

Nous avons donc considéré que le protocole de compression d'en-têtes reçoit des trames d'une couche de liaison se présentant comme un canal à effacements avec un taux de perte FER, et transmet à la couche application des paquets valides avec un taux de perte paquets (PER), et par définition $PER \geq FER$. Bien entendu, dans ce taux PER, le taux FER n'est pas imputable à la couche HC mais à la couche liaison. La donnée qui va principalement nous intéresser est donc le taux de pertes imputable à la compression rapporté au taux de perte en entrée. Nous étudierons également deux autres métriques : l'efficacité de cette compression, et le temps moyen de déconnexion,

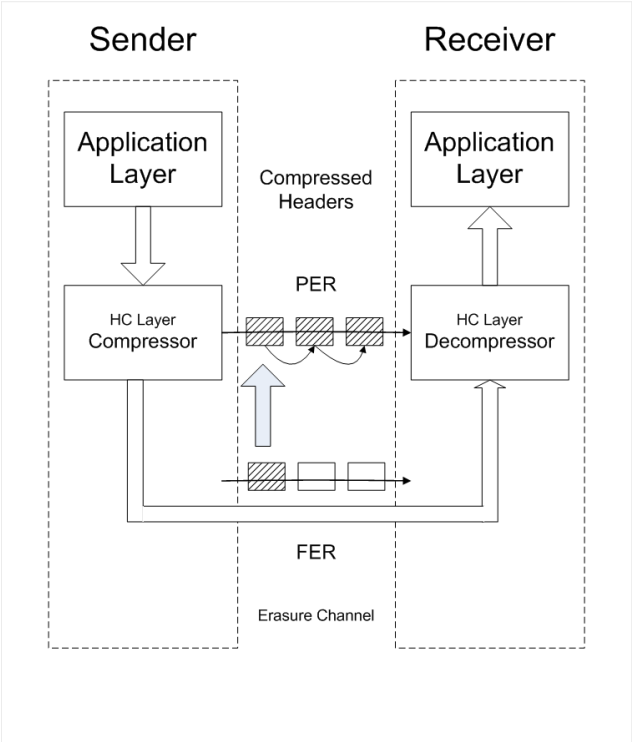

Figure 3.1 – Représentation du protocole de compression d'en-têtes dans un modèle OSI d'émetteur/récepteur simplifié

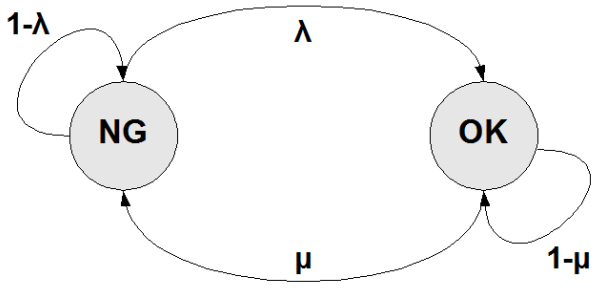

Figure 3.2 – Modélisation du canal de liaison à effacements par un modèle de Gilbert-Elliott basé sur une chaîne de Markov à deux états

qui correspond au temps pendant lequel, suite à une perte en entrée, le décompresseur se retrouve dans l'impossibilité de décoder de nouveaux paquets, et se met en attente d'un paquet moins ou non compressé.

3.2.1 Le modèle de Gilbert-Elliott

Dans les premiers travaux que nous avions menés sur ce sujet, nous avions utilisé le modèle de pertes le plus simple, à savoir celui de pertes isolées, où chaque trame est associée à la même probabilité d'effacement FER. Nous avons alors décidé, au prix d'une plus grande complexité du modèle, de considérer la couche liaison comme un canal de Gilbert-Elliot [55] [56] basé sur une chaîne de Markov à deux états (Figure 3.2).

Dans le modèle présenté par Gilbert, les deux états représentent un état Bon (Good/OK) où la probabilité de pertes P_G est nulle, et un état Mauvais (Bad/NG), où la probabilité de pertes P_B est importante. Elliott a par la suite généralisé ce modèle en proposant le fait que P_G soit non nul. Dans notre modèle et pour la suite de ce chapitre, nous avons pris l'hypothèse que $P_G = 0$ et $P_B = 1$. Ceci signifie que, lorsque le canal se trouve dans l'état Bon, toutes les trames sont correctes, et dans l'état Mauvais, toutes les trames sont invalides ou perdues. A ces états, on associe des probabilités de transition λ et μ, λ représentant la probabilité de transition vers l'état Bon depuis l'état Mauvais, et μ l'inverse.

Ce type de canal permet ainsi de modéliser le comportement d'un canal, qui en temps normal est correct, mais subit des *bursts* de pertes. Ce comportement est classique, par exemple, pour une voie satellite, où le taux d'erreur du lien est très faible ($< 10^{-5}$ défini dans les standards [57] [58]) en comparaison de la congestion aux bornes de ce lien, qui peut être de l'ordre de 10^{-2}.

Un autre avantage de ce canal est de se présenter comme un cas plus général que le canal de Bernouilli, ceux-ci pouvant être vu comme un canal GE particulier avec $\lambda = \mu = \frac{1}{2}$. Le canal de Gilbert-Elliott a également servi de référence pour modéliser les pertes de l'Internet [59]. Ce modèle de canal s'est également montré efficace pour évaluer des trafics similaires [60].

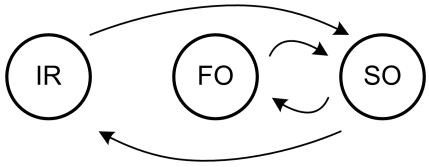

Figure 3.3 – Machine à états du modèle de compresseur

Lorsque la chaîne de Markov se trouve dans un état stationnaire, la probabilité que celle-ci se soit trouvée dans l'état Mauvais est $\frac{\mu}{\lambda+\mu}$. Cette probabilité correspond asymptotiquement au taux de perte trames du canal FER. Ceci nous permet alors de relier ce taux aux paramètres de la chaîne de Markov :

$$\mu = \lambda \times \frac{FER}{1 - FER}$$

Afin de déterminer complètement le canal, il suffit alors de déterminer λ. Supposons que la chaîne se trouve dans l'état Mauvais. La transition à l'état Bon correspond alors au tirage successif et indépendant d'un événement v de probabilité λ jusqu'à réussite. La variable λ correspond ainsi à la variable d'une loi géométrique qui régit alors la longueur d'une salve de pertes. Grâce à ce résultat, il est alors très intéressant de rattacher ce λ à la longueur moyenne des salves de pertes du canal liaison, que nous nommerons m via la relation :

$$m = E[v] = \frac{1}{\lambda}$$

Nous sommes alors capables de caractériser la chaîne de Markov régissant le comportement du canal liaison simplement en connaissant le taux de perte trames moyen FER et la longueur moyenne des salves de pertes m.

3.2.2 Le modèle de compression

Dès lors que le modèle de canal en entrée est défini, il reste à présenter le modèle de la couche de compression.

Le compresseur, tel que visible sur la Figure 3.3, est composé d'une machine à états à trois états : un état appelé *Initialisation & Refresh* ou IR, un état dit First Order ou FO et enfin un état *Second Order* ou SO. Ces états sont volontairement calqués sur les états du compresseur du protocole ROHC, ceci permettant de rester le plus fidèle possible aux protocoles existants, mais aussi de se ramener de manière simple à un système de compresseur à deux états. Les transitions possibles dans cette machine à états sont celles entre IR et SO, et FO et SO.

Dans ce modèle de compresseur, l'état IR correspond à un état de récupération ou des en-têtes non compressés sont envoyés. L'état FO correspond à un état intermédiaire, similaire à celui de ROHC, c'est à dire permettant de transmettre l'ensemble

du contexte non-statique de manière non compressée. L'état normal du compresseur est l'état SO où les en-têtes les plus petits possibles sont transmis.

Il reste alors à définir les conditions permettant au récepteur de récupérer un contexte perdu, suite à un paquet antérieur non reçu. Lorsque un paquet de type IR ou FO est reçu par le décompresseur, nous considérons que celui-ci permet de récupérer ce contexte et de délivrer de nouveaux paquets, car dans ce cas, l'ensemble du contexte non-statique est transporté.

Lorsque le contexte est désynchronisé est que le récepteur ne reçoit que de nouveaux paquets SO, la récupération n'est pas garantie, et est, comme nous l'avons vu dans le chapitre précédent, soumis aux performances des algorithmes de récupération mis en place (TWICE, W-LSB, ...). Il est également admissible que, plus la distance entre le premier paquet SO reçu et la perte initiale est grande, plus l'algorithme de récupération aura de chances d'échouer. Nous modéliserons alors ce comportement grâce à une probabilité $P(\Delta)$ décroissante en Δ, où δ représente la distance du premier paquet SO reçu à la perte initiale. Notons par ailleurs que si cette récupération sur un premier paquet SO échoue, nous considérons qu'il est alors impossible de récupérer celui-ci via un paquet SO postérieur, ce qui semble une hypothèse raisonnable, dans la mesure ou son contexte sera encore plus lointain que le contexte du premier SO reçu du contexte de la perte initiale.

Nous nous proposons d'étudier et de comparer ce modèle sur trois scénarios. Tout d'abord un premier mode nommé Unidirectionnel (U) sans voie de retour, où les transitions du compresseur sont basées sur des *timers*. Un second mode appelé Optimiste (O) qui sera composé de *timers* pour les transitions ascendantes et de réception d'acquittements négatifs pour la voie descendante du compresseur, et enfin un dernier mode Fiable (R) basé uniquement sur les acquittements du récepteur.

Nous proposons trois métriques pour évaluer et comparer ces différents modes. Tout d'abord, nous essaierons de déterminer ν, le nombre moyen de pertes imputables à la couche de compression suite à une perte de paquets en entrée. Nous tenterons également de déterminer e définit comme l'efficacité pratique de la compression, où le rapport de l'en-tête moyen à la taille de l'en-tête non compressé. Enfin la dernière métrique qui nous a semblé pertinente est celle du délai moyen de resynchronisation, lié à ν suite à une perte de paquets.

3.3 Modélisation du mode Unidirectionnel

Pour le mode Unidirectionnel, les transitions entre les différents états se font grâce à des *timers*. En considérant que le débit de transmission est constant, on peut alors, de manière équivalente, considérer que le compresseur envoie un nombre fixe de paquets d'un même état/type avant de transiter vers son prochain état. Dans la suite, nous appellerons l'ensemble de ces paquets une salve. Chaque état différent est alors associé avec une longueur propre de salve que nous appellerons respectivement I, F et S pour les états IR, FO et SO. Par ailleurs nous prenons l'hypothèse que les transitions

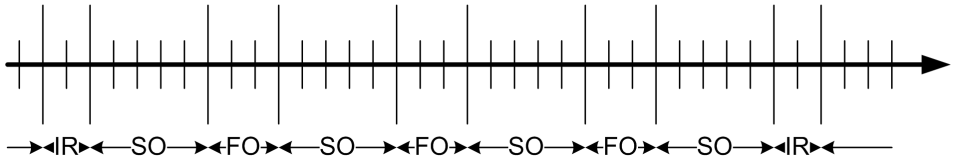

Figure 3.4 – Chronographe du modèle de compresseur en mode Unidirectionnel

descendantes de SO vers FO ou IR sont régulières et entrelacées. Ceci signifie que lorsque le compresseur a effectué k transitions descendantes vers FO successives, sa prochaine transition sera vers IR. Sur la Figure 3.4, nous illustrons le comportement du chronographe du compresseur pour $k = 3$. Le quadruplet $\{I, F, S, k\}$ correspond donc à l'entrée de notre modèle.

Lorsque nous souhaitons déterminer ν (qui correspond, rappelons-le au nombre moyen de pertes imputables à la couche de compression suite à une perte de paquets en entrée) pour ce mode Unidirectionnel, il est alors important de noter que le nombre de pertes successives à une perte initiale dépend de l'état du paquet de la perte initiale. En effet, il est nécessaire de distinguer le fait que la perte initiale appartienne à IR, FO ou SO, le schéma de récupération étant alors différent. Pour le calcul de ν, définissons tout d'abord ν_{IR}, ν_{FO} et ν_{SO} qui correspondent au calcul de ν selon l'état du paquet initial perdu.

Pour le mode Unidirectionnel, nous avons alors :

$$\nu = \frac{I \times \nu_{IR} + kF \times \nu_{FO} + (k+1)S \times \nu_{SO}}{I + kF + (k+1)S}$$

En prenant une notation générique où l'état en question est X, chaque ν_X étant la moyenne pour chacun de ses paquets :

$$\nu_X = \frac{1}{X} \times \sum_{i=0}^{X-1} \nu_X(i)$$

où $\nu_X(i)$ représente le nombre moyen de pertes suite à la perte du i^{eme} paquet d'une salve X. Pour des raisons de commodité, la numérotation des en-têtes d'une même salve sera inversée : le premier en-tête transmis sera donc d'indice $X - 1$ et le dernier sera d'indice 0.

Nous pouvons cependant remarquer que la perte de paquet IR ou FO de même position entraîne, sans perte de généralité notable, le même schéma de récupération (fin de la salve IR/FO puis salve SO puis salve IR/FO etc...), comme nous pouvons l'observer sur le chronographe. Nous pouvons alors poser : $\nu_{IR}(i) = \nu_{FO}(i)$ pour tout i de 0 à $\min(I, F) - 1$. La différence de ν entre ces deux cas étant alors la différence de longueur de leurs salves respectives. En conséquence, nous traiterons le calcul d'une perte initiale de paquet IR et de paquet FO ensemble. Si cette perte a lieu à la position i, la longueur de cette perte sera notée $\nu_{IR/FO}(i)$. Afin de déterminer ν totalement, il suffit alors de déterminer $\nu_{IR/FO}(i)$ pour $i \in (0, 1, ..., \max(I, F) - 1)$ et

$\nu_{SO} = \frac{1}{S} \times \sum_{i=0}^{S-1} \nu_{SO}(i)$.

Perte initiale d'un paquet IR ou FO Déterminons tout d'abord $\nu_{IR/FO}(i)$. Pour cela, nous avons représenté le schéma de la récupération du contexte suite à une perte d'un paquet IR ou FO sur la Figure 3.5. Les différentes valeurs évaluées dans cette section sont indicées en fonction des étapes numérotées sur cette figure.

Détaillons ce fonctionnement. Tout d'abord, suite à la perte initiale d'un paquet de type IR/FO d'indice i, il est possible de récupérer le contexte suite à la réception d'un des paquets IR ou FO restants de la même salve (1). Dans ce cas, les résultats sur la loi géométrique, nous donnent si un de ces paquets est reçu :

$$\nu_1 = \lambda \sum_{k=0}^{i} k(1-\lambda)^{k-1} = \frac{1-(1-\lambda)^{i+1}}{\lambda} - (i+1)(1-\lambda)^i$$

La probabilité qu'aucun des i éléments restants de la même salve ne soit reçu est

$$p_2 = (1-\lambda)^i$$

Dans ce cas (2), le compresseur transite vers l'état SO comprenant une salve de S éléments. Là aussi, deux cas de figures peuvent apparaître. Dans le premier cas (4), l'ensemble de cette salve est perdue. Nous détaillerons ce scénario par la suite.

Dans le second cas (3), au moins un des éléments de la salve SO est reçu. Comme nous l'avons vu, la réception d'un paquet SO ne garantit pas la récupération du contexte et dépend donc de $P(.)$.

Supposons alors que dans l'ordre nominal $(1, 2, ...S)$, le j^{eme} paquet de la salve SO soit le premier reçu. Les résultats sur la loi géométrique donnent une probabilité d'un tel événement égale à $\lambda(1-\lambda)^{j-1}$ Ce paquet se situe alors à la distance $i+j$ de la perte initiale. En conséquence, si la récupération réussit (5), le nombre moyen supplémentaires de paquets perdus (en plus des $i+1$ de la salve IR/FO initiale) est :

$$\nu_5 = \sum_{j=1}^{S} P(i+j) \times \lambda(1-\lambda)^{j-1} \times (j-1)$$

La probabilité (6) que la récupération du paquet SO échoue est alors :

$$p_6 = \sum_{j=1}^{S} (1 - P(i+j)) \times \lambda(1-\lambda)^{j-1}$$

Dans ce cas là comme nous l'avons vu, comme la réception sur SO a échoue, le décompresseur se place dans l'attente d'un paquet IR ou FO. En conséquence les S éléments de la salve SO sont perdus. Suite à cette salve, une salve IR ou une salve FO va suivre, avec les probabilités correspondantes $\frac{1}{k+1}$ et $\frac{k}{k+1}$.

Afin de déterminer les probabilités de réception des paquets de cette salve non compressée, il est important de remarquer que la dernière information connue sur la

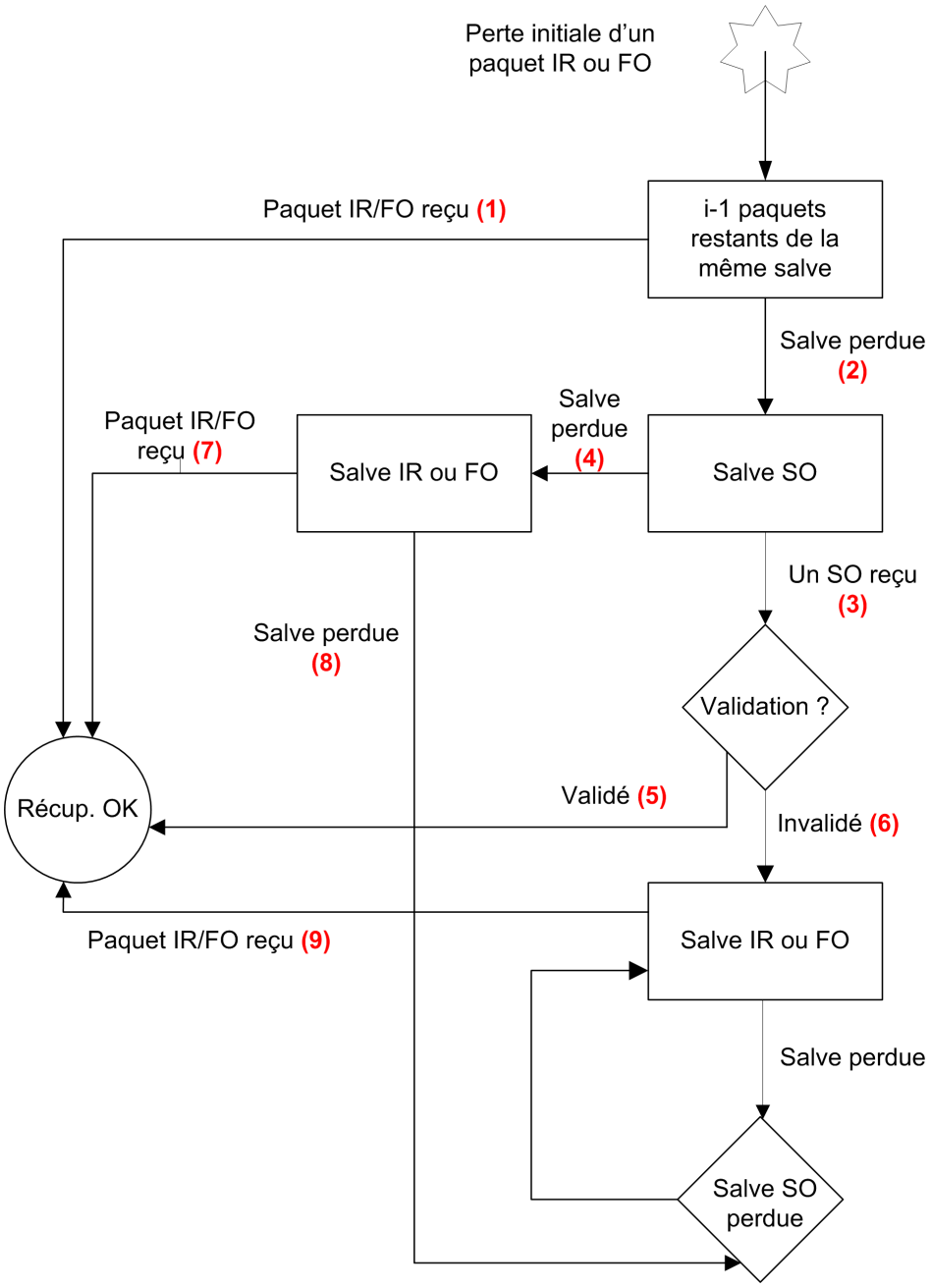

Figure 3.5 – Mécanisme de récupération du contexte suite à la perte d'un paquet IR ou FO

chaîne de Markov du compresseur est celle de la réception du paquet SO invalidé. Ainsi, la probabilité de non-réception du premier élément du burst IR/FO n'est pas $1-\lambda$ mais un terme que nous appellerons $z_0(i)$.

Soit la chaîne de Markov du canal dont l'état initial est l'état OK. Nous avons vu précédemment que la probabilité que la chaîne se retrouve dans l'état NG après un nombre de transitions infini (état stationnaire) est $\frac{\mu}{\lambda+\mu}$. Ceci vient du fait que la probabilité de se retrouver dans l'état NG après t transitions, $t \geq 0$ est en fait :

$$p_t(NG/init = OK) = \frac{\mu(1-(1-\lambda-\mu)^t)}{\lambda+\mu}$$

Dans notre cas, ceci permet le calcul de $z_0(i)$. En effet, la probabilité de non-réception du premier IR/FO est en fait équivalente à la probabilité que la chaîne de Markov soit dans l'état NG sachant qu'un des SO précédents a été reçu, et donc que l'état initial de la chaîne de Markov est OK. Comme nous avons défini la probabilité de réception de chacun de ces SO précédemment, nous avons :

$$z_0(i) = \frac{\sum_{j=1}^{S}(1-P(i+j)) \times \lambda(1-\lambda)^{j-1} \times p_{S+1-j}(NG/init = OK)}{\sum_{j=1}^{S}(1-P(i+j)) \times \lambda(1-\lambda)^{j-1}}$$

Deux cas de figure peuvent alors ainsi apparaitre. Soit un des IR/FO est reçu avec l'information initiale sur la chaîne de Markov du canal précédente, et dans ce cas, le contexte est retrouvé avec le nombre moyen de paquets supplémentaires :

$$\nu_6 = \frac{k}{k+1} z_0(i) \left(\frac{1-(1-\lambda)^F}{\lambda} - F(1-\lambda)^{F-1} \right) + \frac{1}{k+1} z_0(i) \left(\frac{1-(1-\lambda)^I}{\lambda} - I(1-\lambda)^{I-1} \right)$$

Soit toute la salve IR/FO est perdue par la canal. Cet événement a la probabilité :

$$p_{g_1} = z_0(i) \left(\frac{k}{k+1}(1-\lambda)^{F-1} + \frac{1}{k+1}(1-\lambda)^{I-1} \right)$$

Il entraîne alors la perte moyenne des $n_{gen} = \frac{k}{k+1}F + \frac{1}{k+1}I$ éléments de la salve.

Le mécanisme rentre alors dans un état générique ou la récupération n'est désormais possible que sur réception d'un paquet IR/FO et où tous les paquets SO reçus sont éliminés. A ce stade, il suffit alors de déterminer le nombre moyen supplémentaire de paquets perdus dans cet état, que nous noterons Γ, et que nous détaillerons par la suite.

Nous avons donc déterminé le scénario où suite à la perte de l'ensemble du burst IR/FO initial, un élément de la salve suivante SO a été reçu. Il reste tout d'abord à

revenir sur le cas où aucun de ces SO n'a été reçu (4). Tout d'abord la probabilité d'un tel événement est :

$$p_4 = (1-\lambda)^S$$

et les S paquets du SO sont bien évidemment perdus.

Tout comme dans le scénario précédent où un SO a été invalidé (6), le compresseur va alors transiter dans l'état IR ou FO avec les mêmes probabilités respectives que précédemment. Cela dit, dans ce cas, la dernière information reçue sur la chaîne de Markov est ici la perte du dernier paquet SO. En l'occurrence, la probabilité de non-réception du premier paquet de la salve IR/FO est bien $(1-\lambda)$. Tout comme précédemment, si un des IR/FO est reçu (7), le contexte est récupéré avec la perte supplémentaire de :

$$\nu_4 = \frac{k}{k+1}(1-\lambda)\left(\frac{1-(1-\lambda)^F}{\lambda} - F(1-\lambda)^{F-1}\right) + \frac{1}{k+1}(1-\lambda)\left(\frac{1-(1-\lambda)^I}{\lambda} - I(1-\lambda)^{I-1}\right)$$

Bien entendu, il est possible qu'aucun des paquets de la salve IR/FO ne soit reçu (8). Ceci peut apparaître avec la probabilité :

$$p_{g_2} = \frac{k}{k+1}(1-\lambda)^F + \frac{1}{k+1}(1-\lambda)^I$$

Nous retombons alors dans le cas générique présenté précédemment où les n_{gen} éléments moyens sont perdus, et où la récupération se fait alors sur réception d'un paquet IR/FO, les paquets SO étant alors systématiquement ignorés. Dans ce cas, le nombre moyen supplémentaires d'éléments perdus est également Γ. Il reste alors à déterminer cette valeur du cas générique.

Dans le cas générique où en moyenne Γ paquets supplémentaires sont perdus, l'état initial de la chaîne de Markov du canal est celui où le dernier paquet d'une salve non compressé a été perdu. Cet état peut être vu comme la succession régulière de k salves FO et d'une salve IR, séparées par la perte d'une salve de S paquets SO. Le contexte est récupéré (9) dès lors qu'un de ces en-têtes non compressé est reçu. La valeur Γ peut alors être vue comme le calcul d'une succession de salves IR ou FO perdues, ainsi que la salve SO successive, jusqu'à une salve IR ou FO dont au moins un des éléments est reçu.

Dans tous les cas, l'état initial de la chaine de Markov étant la perte du dernier élément de la dernière salve non compressée, la probabilité de non-réception du premier en-tête de la salve courante est alors équivalent à la probabilité que la chaîne de Markov du canal se trouve dans l'état NG au bout de $S+1$ transitions, sachant que l'état initial était NG. Les résultats sur la chaîne de Markov avec cet état initial donnent :

$$p_t^{'}(NG/init = NG) = \frac{\mu - \lambda(1-\lambda-\mu)^t}{\lambda+\mu}$$

La probabilité de non-réception du premier paquet du premier IR ou FO reçu est alors égale à la valeur $p_{S+1}^{'}(NG/init = NG)$ que nous noterons par la suite ζ. En conséquence, la probabilité que l'ensemble d'une salve IR ou FO soit perdue est alors : $\zeta(1-\lambda)^{X-1}$ ou X représente la taille de la salve IR ou FO selon le cas.

Lorsque le contexte est enfin re-synchronisé suite à la réception d'un paquet d'une salve IR ou FO, de nouveaux paquets sont tout de même perdus, car ce n'est pas forcément le premier paquet de la salve qui a été reçu. Nous noterons cette nouvelle valeur z_X^{S+1} où X représentera I dans le cas IR et F dans le cas d'une salve FO. Cette valeur vaut alors :

$$z_X^{S+1} = \frac{\zeta}{1-\zeta(1-\lambda)^{X-1}} \times \left(\frac{1-(1-\lambda)^X}{\lambda} - X(1-\lambda)^{X-1} \right)$$

La valeur Γ correspond alors au produit du nombre de paquets perdus $N(y)$ par la probabilité qu'une salve non compressée (un paquet ici) soit reçue, alors que les précédentes ont été perdus $p_N(y)$:

$$\Gamma = \sum_{y=0}^{\infty} p_N(y).N(y)$$

La probabilité pour la salve y s'exprimant alors par :

$$p_N(y) = \zeta^y \times \underbrace{\left((1-\lambda)^{k(F-1)+(I-1)}\right)^{\lfloor \frac{y}{k+1} \rfloor}}_{\text{Groupes de k salves FO et 1 salve IR perdues}}$$
$$\times \underbrace{\left(\frac{y'}{k+1}(1-\lambda)^{(F-1)(y'-1)+(I-1)} + \frac{(k+1-y^{'}) \mod (k+1)}{k+1}(1-\lambda)^{y'(F-1)} \right)}_{\text{Reste non groupable des salves}}$$
$$\times \underbrace{\left(\frac{k}{k+1}\left(1-\zeta(1-\lambda)^{F-1}\right) + \frac{1}{k+1}\left(1-\zeta(1-\lambda)^{I-1}\right) \right)}_{\text{Salve reçue}}$$

où $y^{'}$ représente le reste de la division entière de y par $k+1$. Ce terme peut être exprimé sous la forme :

$$y^{'} = y - \lfloor \frac{y}{k+1} \rfloor (k+1)$$

Lorsque la salve y est reçue, le nombre moyen de paquets perdus est alors :

$$N(y) = \lfloor \frac{y}{k+1} \rfloor (k(F+S) + (I+S)) + y' \left((F+S) + \frac{1}{k+1}(I-F) \right) + \left(\frac{k}{k+1} z_F^{S+1} + \frac{1}{k+1} z_I^{S+1} + S \right)$$

Nous avons donc défini tous les éléments, ce qui en résumé nous donne :

$$\nu_{IR/FO}(i) = \nu_1 + (1-\lambda)^i \times [(i+1) + \nu_5 + p_6 (S + \nu_6 + p_{g_1}(n_{gen} + \Gamma)) + p_4 (S + \nu_4 + p_{g_2}(n_{gen} + \Gamma))]$$

Perte initiale d'un paquet SO Le calcul de ν_{SO} va être effectué de la même manière. Celui-ci est légèrement plus complexe, mais ressemble néanmoins fortement au calcul précédent. L'objectif est de calculer pour chaque i, $\nu_{SO}(i)$. De la même manière que précédemment, nous avons représenté le modèle de récupération du contexte suite à une perte initiale d'un paquet SO sur la Figure 3.6.

Tout d'abord, il est possible de décoder si l'un des paquets restants de la même salve SO est reçu (1). Bien entendu, celui-ci est alors soumis à la fonction $P(.)$ et n'est pas garanti. Si la récupération réussit (2), le nombre moyen de paquets nécessaires est alors dans ce cas :

$$\nu_1 = \sum_{k=1}^{i} \lambda(1-\lambda)^{k-1} \times P(k) \times k$$

Néanmoins, la probabilité que cette récupération échoue est alors (3) :

$$p_3 = \sum_{k=1}^{i} \lambda(1-\lambda)^{k-1} \times (1 - P(k))$$

Dans ce cas là, les $i+1$ paquets restants de la salve SO sont perdus, et le décodage sur SO ayant échoué, on considère alors que le décodage n'est plus possible que sur réception d'un en-tête non compressé. Nous retombons alors sur les résultats du cas générique. Pour cela, il suffit alors de déterminer la probabilité de non-réception du premier paquet de la salve IR/FO suivante. Pour cela, la dernière information connue sur la chaîne de Markov du canal, est la réception d'un paquet parmi les i paquets restants de la salve par hypothèse. La probabilité de non-réception est alors dépendante de la position i de la perte initiale et sera alors ici notée z_0^i. Elle correspond alors à la somme des cas possibles de réception de paquet SO parmi les i restants de la salve, soit :

$$z_0^i = \frac{\sum_{j=1}^{i} (1 - P(j)) \times \lambda(1-\lambda)^{j-1} \times p_{i+1-j}(NG/init = OK)}{\sum_{j=1}^{i} (1 - P(j)) \times \lambda(1-\lambda)^{j-1}}$$

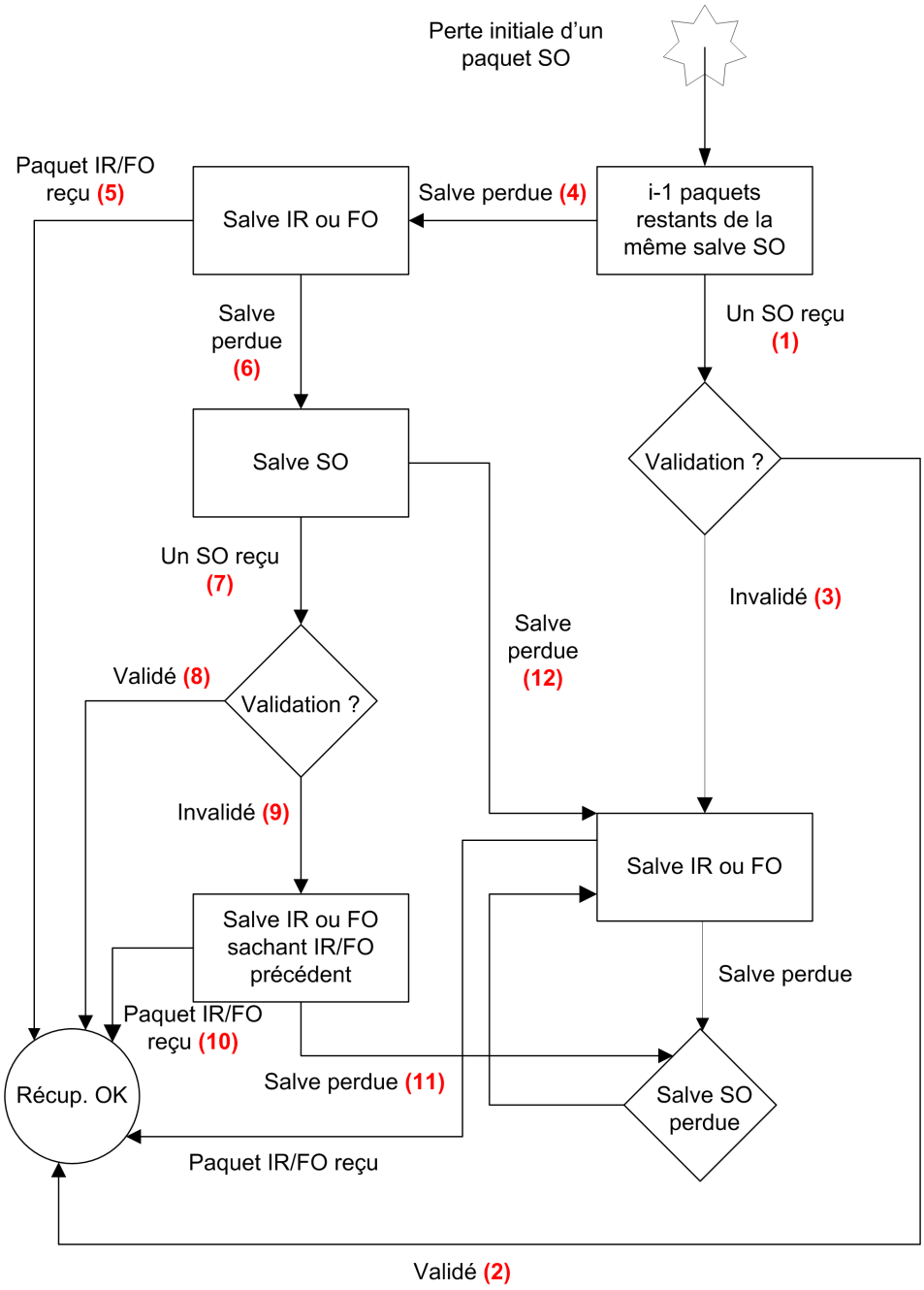

Figure 3.6 – Mécanisme de récupération du contexte suite à la perte d'un paquet SO

De manière similaire, si un des paquets de la salve IR ou FO est reçu, le nombre moyen de symboles ajoutés est alors :

$$\nu_r = \frac{k}{k+1} z_0^i \left(\frac{1-(1-\lambda)^F}{\lambda} - F(1-\lambda)^{F-1} \right) + \frac{1}{k+1} z_0^i \left(\frac{1-(1-\lambda)^l}{\lambda} - l(1-\lambda)^{l-1} \right)$$

Cependant il est possible que toute la salve soit perdue, avec la probabilité suivante :

$$p_r = z_0^i \left(\frac{k}{k+1}(1-\lambda)^{F-1} + \frac{1}{k+1}(1-\lambda)^{l-1} \right)$$

Dans ce cas nous retombons alors sur le cas générique du calcul sur IR/FO ou en moyenne $n_{gen} + \Gamma$ paquets supplémentaires seront perdus.

En résumé, le nombre moyen de paquets ajoutés en prenant l'hypothèse qu'un des paquets restants de la salve initiale SO soit reçu peut alors se résumer à :

$$\nu_{rcvd}(i) = \nu_1 + p_3 \left((i+1) + \nu_r + p_r(n_{gen} + \Gamma) \right)$$

Il reste alors à traiter le cas (4) où tous les paquets restants de la salve SO ont été perdus par la couche liaison.

Le cas (4) où tous les paquets restants de la salve SO initiale ont été perdus a donc pour probabilité $p_4 = (1-\lambda)^i$ et conduit bien entendu à au moins $i+1$ pertes. La salve suivante est alors une salve IR ou une salve FO avec les probabilités $\frac{1}{k+1}$ et $\frac{k}{k+1}$. Cependant, la dernière information sur la chaîne de Markov du canal connue est tout simplement que le dernier paquet de la salve SO a été perdu, par définition. Ceci signifie donc que la probabilité de non-réception du premier paquet de la salve IR/FO est tout simplement $(1-\lambda)$. Nous retombons alors sur un scénario connu du calcul IR/FO. Soit la récupération a lieu suite à la réception d'un paquet IR/FO (5) et dans ce cas on ajoute :

$$\nu_5 = \frac{k}{k+1}(1-\lambda) \left(\frac{1-(1-\lambda)^F}{\lambda} - F(1-\lambda)^{F-1} \right) + \frac{1}{k+1}(1-\lambda) \left(\frac{1-(1-\lambda)^l}{\lambda} - l(1-\lambda)^{l-1} \right)$$

qui, nous pouvons le remarquer, est alors identique au calcul de ν_4 dans le calcul pour IR/FO.

Soit tous les paquets de la salve sont perdus (6), ce qui représente en moyenne :

$$n_{gen2} = \frac{k}{k+1}F + \frac{1}{k+1}I$$

avec la probabilité suivante :

$$p_6 = \frac{k}{k+1}(1-\lambda)^F + \frac{1}{k+1}(1-\lambda)^I$$

qui est bien équivalente à celle de p_{g_2} du calcul précédent.

A ce stade du calcul, nous pouvons appliquer deux politiques. La plus simple consiste à considérer la perte de l'ensemble de la salve non-compressée comme rédhibitoire et de se ramener au cas générique. Nous avons jugé cette hypothèse comme trop restrictive. En effet, dans l'exemple où la perte initiale est apparue à la fin de la salve SO, et où l'ensemble de la salve non-compressée successive a été perdue, il semble tout à fait concevable, au vu des tailles de salves non-compressés utilisées, que le contexte n'ait pas changé à la fin de la salve non-compressé, ce qui rend le décodage possible via la salve SO suivante, avec bien évidemment une probabilité réduite au vu de la distance séparant cette nouvelle salve à la perte initiale. Nous avons donc retenu cette hypothèse, ce qui signifie que nous considérons la tentative de décodage sur la salve SO suivante comme valide.

Nous revenons alors sur un calcul classique. Soit un paquet d'un paquet de la salve SO est reçu (7), soit aucun n'est reçu (12). Traitons tout d'abord le premier cas. Soit le paquet en question permet de valider le contexte (8) et il apporte alors un nombre moyen de paquets supplémentaires :

$$\nu_8 = \frac{k}{k+1}\sum_{k=1}^{S}\lambda(1-\lambda)^{k-1}P(k+i+F)\times(k-1) + \frac{1}{k+1}\sum_{k=1}^{S}\lambda(1-\lambda)^{k-1}P(k+i+I)\times(k-1)$$

ou bien, celui-ci ne permet pas de valider le contexte (9) avec la probabilité associée :

$$p_9 = \frac{k}{k+1}\sum_{k=1}^{S}\lambda(1-\lambda)^{k-1}(1-P(k+i+F)) + \frac{1}{k+1}\sum_{k=1}^{S}\lambda(1-\lambda)^{k-1}(1-P(k+i+I))$$

A ce point, le contexte n'est toujours pas récupéré, et le compresseur envoie alors une salve IR ou FO. Lorsque nous souhaitons calculer la probabilité de non-réception de cette nouvelle salve IR ou FO il est judicieux de remarquer que si c'est une salve IR, la salve non-compressée précédente (la première) était automatiquement une salve FO,

car deux salves IR ne peuvent être transmises successivement. de manière similaire, si c'est une salve FO, la salve non-compressée précédente était alors soit une salve IR, avec la nouvelle probabilité $\frac{1}{k}$ ou une salve FO avec la probabilité $\frac{k-1}{k}$. Ceci nous permet donc de calculer cette probabilité de non réception, en tenant compte plus précisément de la salve précédente. Cette probabilité, notée z_1^X où X est soit I, soit F, vaut alors :

$$z_1^X = \frac{\sum_{j=1}^{S}(1-P(i+j+X)) \times \lambda(1-\lambda)^{j-1} \times p_{S+1-j}(NG/init = OK)}{\sum_{j=1}^{S}(1-P(i+j+X)) \times \lambda(1-\lambda)^{j-1}}$$

L'état initial de la chaîne de Markov étant la réception d'un des paquets de la salve SO précédente, un des paquets de la salve IR/FO reçu (10)ajoute donc :

$$\nu_{10} = \frac{k}{k+1}\left(\frac{k-1}{k}z_1^F + \frac{1}{k}z_1^I\right) \times \left(\frac{1-(1-\lambda)^F}{\lambda} - F(1-\lambda)^{F-1}\right) + \frac{1}{k+1}z_1^F\left(\frac{1-(1-\lambda)^I}{\lambda} - I(1-\lambda)^{I-1}\right)$$

Soit tout le burst IR/FO est perdu (11), avec la probabilité :

$$p_{11} = \frac{k}{k+1}\left(\frac{k-1}{k}z_1^F + \frac{1}{k}z_1^I\right)(1-\lambda)^{F-1} + \frac{1}{k+1}z_1^F(1-\lambda)^{I-1}$$

et on retombe alors dans le cas générique où $n_{gen} + \Gamma$ paquets seront perdus en moyenne.

Il reste alors à revenir sur le cas où toute la salve SO précédente a été perdue (12), qui a une probabilité de $(1-\lambda)^S$. Dans ce cas, nous considérons alors que la perte est alors suffisamment longue pour avoir vu le contexte changé depuis la perte initiale. Nous nous ramenons, alors, là aussi, au cas générique.

Nous avons donc, en résumé :

$$\nu_{SO}(i) = \nu_{rcvd}(i) + (1-\lambda)^i\left((i+1) + \nu_5 + p_6\left(n_{gen2} + \nu_8 + (1-\lambda)^S\alpha + p_9\beta\right)\right)$$

avec

$$\alpha = S + \nu_5 + p_6(n_{gen} + \Gamma)$$

et

$$\beta = S + \nu_{10} + p_{11}(n_{gen} + \Gamma)$$

Nous avons donc conclu le calcul de ν pour le mode unidirectionnel.

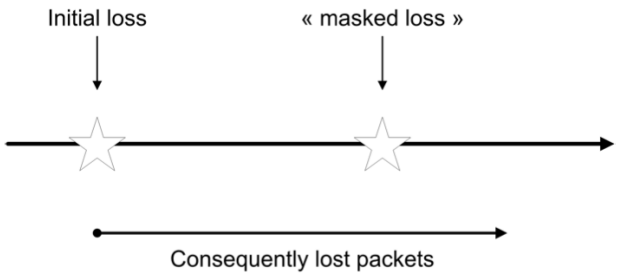

Figure 3.7 – Principe des pertes masquées

Calcul des différentes métriques Si la valeur de ν permet de nous renseigner sur le nombre moyen de pertes suite à une perte initiale de la couche liaison, il faut néanmoins faire le lien entre ν, le FER en entrée et le PER en sortie. En effet, lors de la perte de synchronisation, certaines de ces pertes sont imputables à la couche de compression, alors que d'autres sont bien des pertes réelles au niveau de la couche liaison. En fait, nous avons déterminé, que lorsque une perte apparaît au niveau de la couche liaison, celle-ci n'est pas isolée. Nous avons même déterminée que celle-ci s'accompagnait d'une salve de pertes de longueur moyenne m. Ceci signifie que les m premières pertes dans le calcul de ν sont en fait tout simplement des pertes en entrée, pour lesquelles la couche de compression n'est pas responsable. A ceci, s'ajoute un phénomène de "pertes masquées" que nous avons représenté sur la Figure 3.7. En effet, suite à ces pertes initiales, peut s'ajouter sur les éléments restants de ν d'autres pertes du canal.

En effet, sur le "reste" de ν, des pertes imputables au canal peuvent apparaître avec une probabilité FER. En somme, pour une longueur de pertes ν, il est raisonnable de dire que $m + (\nu - m)FER$ de ces pertes sont en fait des pertes du canal. le ratio entre ν est cette valeur nous donne alors le ratio entre le PER et un FER en entrée donné :

$$PER = \frac{\nu}{m + FER(\nu - m)} \times FER$$

Ce calcul est valable également pour les modes Optimiste et Fiable, dès lors que ν est donné.

Les autres calculs de métriques sont cependant spécifiques.

Pour le mode Unidirectionnel, le calcul de l'efficacité de compression est relativement simple. Soit l_{IR}, l_{FO} et l_{SO} les longueurs respectives des en-têtes IR complets, FO et SO. L'efficacité de la compression du mode U s'exprime alors ainsi :

$$e_U = \frac{I\ l_{IR} + k\ F\ l_{FO} + (k+1)\ S\ l_{SO}}{(I + k\ F + (k+1)\ S) \times l_{IR}}$$

De la même manière, si l_p représente la longueur moyen des données d'un paquet (*payload*) et que les tailles de paquets sont quasi-constantes, le temps moyen de resynchronisation suite à une perte initiale s'exprime par :

$$t_U = \nu \times \frac{l_P + e\ l_{IR}}{R}$$

où R représente le débit moyen du lien.

3.4 Modélisation du mode Optimiste

Fort heureusement, le calcul du nombre moyen de pertes ν est plus simple pour les modes Optimiste et Fiable. En effet, dans ces deux cas, il n'y a pas de mécanismes de *timers*, et dans l'état général, le compresseur n'envoie que des paquets de type SO.

Lorsque une perte apparaît sur la couche liaison, le décompresseur reçoit alors un des paquets SO suivants. Grâce aux mécanismes de numérotation, celui-ci va alors engager de manière classique des mécanismes de récupération. Deux cas peuvent alors apparaître. Soit le mécanisme de récupération fonctionne, et le contexte et récupéré avec en moyenne ν_{OK} pertes :

$$\nu_{OK} = \sum_{n=1}^{\infty} \lambda(1-\lambda)^{n-1} P(n) \times n$$

Soit le mécanisme de validation échoue et ces mêmes paquets sont perdus, ainsi que l'en-tête reçu. Le décompresseur envoie alors un acquittement négatif au compresseur, et se met en attente d'un en-tête non compressé, tout en éliminant tous les paquets SO.

Lorsque le compresseur reçoit cet acquittement négatif, en prenant l'hypothèse que le canal de retour est sans pertes, celui-ci envoie alors une salve de F paquets FO. Le décompresseur est dans la mesure de recevoir un de ces paquets après un délai correspondant au délai aller-retour du lien ou RTT. Cela signifie que l'ensemble des paquets SO transmis durant ce délai sont ignorés. Si R représente le débit du lien de transmission et l la longueur moyenne des paquets transmis, cela représente alors une perte de $n_{RTT} = \frac{R}{l} \times RTT$ paquets pour cette période.

Bien entendu, l'ensemble de cette salve FO peut, elle aussi, être perdue. Le compresseur utilisant un mécanisme de *timers* pour cette transition ascendante envoie donc par la suite de nouveaux en-têtes SO. SI la salve FO est perdue, alors que le décompresseur est dans l'attente d'un en-tête compressé, celui-ci finira alors par recevoir un paquet SO, qu'il sera bien entendu incapable de décoder. Il renverra alors un nouvel acquittement négatif, et ainsi de suite, jusqu'à réception d'un en-tête FO.

Il est donc nécessaire d'évaluer le nombre moyen de paques perdus suite à ce mécanisme que nous noterons $\nu_{lost}(n)$, où n représente la distance entre la première perte initiale et le premier paquet SO reçu qui a invalidé le contexte et qui a provoqué l'envoi du premier acquittement négatif.

En résumé, pour le mode Optimiste, nous avons :

$$\nu = \nu_{OK} + \sum_{n=1}^{\infty} \lambda(1-\lambda)^{n-1}(1-P(n)) \times \nu_{lost}(n)$$

où $\nu_{lost}(n)$ se présente comme la somme des $n+1$ éléments perdus jusqu'au premier paquet SO reçu et invalidé, des n_{RTT} éléments perdus durant le délai aller-retour et d'une composante dépendante de la bonne réception de la salve FO envoyée à la suite de l'acquittement négatif :

$$\nu_{lost}(n) = n + n_{RTT} + 1 + Z$$

Il reste alors à déterminer Z. de la même manière que pour le mode U, le décompresseur peut ne pas recevoir le premier paquet de la salve FO. Il est possible de calculer cette probabilité de non-réception, qui se base sur le fait que le paquet reçu un RTT auparavant a été reçu, car c'est un paquet SO invalidé qui a entraîné l'envoi de l'acquittement négatif, ceci signifie que, lorsque un paquet d'une salve FO est reçu par le décompresseur, le nombre moyen d'en-têtes perdus de cette salve FO est alors :

$$z_f = \frac{p_{n_{RTT}}(NG/init = OK)\left(\frac{1-(1-\lambda)^F}{\lambda} - F(1-\lambda)^{F-1}\right)}{1 - p_{n_{RTT}}(NG/init = OK)(1-\lambda)^{F-1}}$$

Ceci peut se traduire par le fait que dès lors qu'un paquet d'une salve FO est reçu et permet de récupérer le contexte, en moyenne, tout de même, z_f paquets de cette salve ont été perdus.

Il est possible malgré tout de perdre l'ensemble d'une salve FO avec la probabilité $p_{n_{RTT}}(NG/init = OK)(1-\lambda)^{F-1}$. Dans ce cas, les F paquets FO sont bien entendus perdus ainsi que les n_{RTT} envoyés pendant le traitement de l'acquittement et les z_s paquets envoyés suite à la réception de ce NACK jusqu'à détection par le décompresseur et envoi d'un nouveau NACK. z_s correspond tout simplement au résultat de la loi géométrique :

$$z_s = \sum_{n=1}^{\infty} \lambda(1-\lambda)^{n-1} \times n = \frac{1}{\lambda}$$

Hormis z_f, le calcul de Z se résume alors au calcul du nombre moyen de paquets perdus jusqu'à réception d'une salve FO, qui peut-être perdue avec la probabilité $p_{n_{RTT}}(NG/init = OK)(1-\lambda)^{F-1}$, et qui entraine alors la perte moyenne de $F + n_{RTT} + \frac{1}{\lambda}$ paquets :

$$Z = z_f + (F + \frac{1}{\lambda} + n_{RTT}) \sum_{k=1}^{\infty} k((1-\lambda)^{F-1} p_{n_{RTT}}(NG/init = OK))^k$$

Et le calcul de ν vaut alors :

$$\nu = \frac{1}{\lambda} + \lambda(n_{RTT} + 1 + Z)\left(\frac{1}{\lambda} - \sum_{n=1}^{\infty}(1-\lambda)^{n-1}P(n)\right)$$

Le calcul du temps de resynchronisation moyen suite à une perte initiale est identique à celui du mode Unidirectionnel.

Il reste alors à déterminer l'efficacité moyenne de la compression en mode Optimiste. Par définition, la couche de compression protocolaire implique un taux de perte en sortie égal à PER, ce qui en d'autres termes se traduit par le fait, qu'en moyenne le protocole de compression se trouve dans un état nominal avec la probabilité $(1-PER)$ et dans un état d'échec avec la probabilité PER. Durant cet état nominal, seuls des paquets SO sont envoyés, et lorsque le protocole est en échec, le mécanisme présenté ci-dessus est appliqué. Dans cet état, des paquets SO et des paquets FO sont envoyés. Dans le mode O, ainsi que dans le mode R, des paquets FO sont envoyés si et seulement si le contexte n'est pas récupéré suite à la réception du premier paquet SO suivant la perte initiale. En effet, si le contexte est récupéré, le compresseur ne reçoit pas d'acquittement négatif et continue alors d'envoyer de manière transparente des en-têtes compressés. La probabilité de non-récupération p_{NOK} étant alors :

$$p_{NOK} = \sum_{n=1}^{\infty} \lambda(1-\lambda)^{n-1}\,(1-P(n))$$

Il reste alors à déterminer le nombre moyen n_{FO} de paquets FO envoyés lorsque le contexte n'est pas récupéré. En suivant le mécanisme présenté précédemment, si le contexte est invalidé, alors au moins une salve FO est envoyée ainsi que de nouvelles salves jusqu'à réception d'une salve FO, chaque salve ayant une probabilité de perte égale à $p_{n_{RTT}}(NG/init = OK)(1-\lambda)^{F-1}$. Dans ce cas, nous avons :

$$n_{FO} = F + \sum_{k=1}^{\infty} kF \times \left((1-\lambda)^{F-1} p_{n_{RTT}}(NG/init = OK)\right)^k$$

L'efficacité moyenne de la compression du mode Optimiste est alors définie par :

$$e = \frac{(1-PER)l_{SO} + PER \times p_{NOK} \times n_{FO} \times l_{FO}}{(1-PER + PER \times p_{NOK} \times n_{FO}) \times l_{IR}}$$

3.5 Modélisation du mode Fiable

Le comportement du mode Fiable est très proche de celui du mode Optimiste, la seule différence étant le traitement d'un acquittement négatif. Là où le mode Optimiste, suite à la réception d'un NACK, envoie une salve FO puis repart sur des en-têtes SO, le mode Fiable attend la réception d'un acquittement positif pour retourner dans l'état SO.

Les seules différences entre ces deux modes concernent alors le calcul de Z et de n_{FO}, les autres élément étant alors identiques à ceux du mode Optimiste.

Tout comme le mode Optimiste, la probabilité de non-réception par le décompresseur du premier paquet FO est $p_{n_{RTT}}(NG/init = OK)$. La valeur de Z suit alors la loi géométrique :

$$Z = \sum_{i=1}^{\infty} \lambda(1-\lambda)^{i-1} p_{n_{RTT}}(NG/init = OK) \times i = \frac{p_{n_{RTT}}(NG/init = OK)}{\lambda}$$

Le nombre moyen n_{FO} de paquets FO envoyés lorsque le contexte n'a pu être récupéré est ici très simple à déterminer. En effet, le compresseur va envoyer des paquets FO dès qu'il reçoit un acquittement négatif, et ce, jusqu'à ce qu'il reçoive un acquittement positif. Ceci correspond alors au nombre de paquets transmis durant un RTT, ainsi que le nombre moyen de paquets Z avant que le décompresseur ne reçoive un de ces paquets FO :

$$n_{FO} = n_{RTT} + Z = n_{RTT} + \frac{p_{n_{RTT}}(NG/init = OK)}{\lambda}$$

En remplaçant ces deux valeurs dans les résultats du mode O, on obtient les résultats sur les métriques souhaités.

3.6 Instantiations du modèle

3.6.1 Instantiation par le protocole ROHC

Nous nous proposons de paramétrer ce modèle en utilisant certains mécanismes de ROHC. Pour cela, afin de correspondre au protocole, nous supposons que les paquets de type UOR-2 de ROHC font ici partie des paquets SO et non des FO. Nous prenons cette hypothèse car le comportement de ce type de paquets est similaire à celui des paquets SO de notre modèle.

Nous proposons d'implémenter le mécanisme W-LSB de ROHC par la fonction de récupération $P(\Delta)$ suivante :

$$\begin{cases} P(\Delta) = 1 & \text{si } \Delta \leq W \\ P(\Delta) = a^{\Delta - W} & \text{si } \Delta > W, a \leq 1 \end{cases}$$

Le premier cas se justifie par le fait que le mécanisme W-LSB garantit une récupération du contexte si tant est que le paquet reçu est à une distance moindre que la taille de la fenêtre glissante W, de la perte initiale. Lorsque la distance dépasse la taille de la fenêtre, le décodage n'est plus garanti. Nous avons pris l'hypothèse de modéliser ce cas par une fonction géométrique décroissante par rapport à cette distance, qui permet de modéliser le taux de réussite de plus en plus faible du W-LSB lorsque la distance à la perte initiale augmente.

Ceci nous permet notamment de déterminer les valeurs des efficacités et de ν pour les modes O et R du modèle. Ainsi p_{NOK} devient :

$$p_{NOK} = (1-\lambda)^W - \frac{\lambda a (1-\lambda)^W}{1 - a(1-\lambda)}$$

La valeur de ν dans le cas Optimiste étant ainsi :

$$\nu_U = \frac{1}{\lambda} + (1-\lambda)^W \left(\frac{(1-a)(n_{RTT}+1+Z)}{1-a(1-\lambda)} \right)$$

et pour le mode Fiable :

$$\nu_R = \frac{1}{\lambda} + (1-\lambda)^W \left(\frac{(1-a)(n_{RTT}+1+\frac{p_{n_{RTT}}(NG/init=OK)}{\lambda})}{1-a(1-\lambda)} \right)$$

3.6.2 Instantiation par un modèle à deux états

De par la structure du modèle présenté, il est aisé de se ramener à un comportement d'un protocole à deux états : un état non-compressé et un état compressé. Pour cela, il suffit dans le modèle, de fusionner les états IR et FO, en prenant leurs paramètres égaux : $I = F$ et $l_{IR} = l_{FO}$. Ce cas précis permet de modéliser le comportement de protocoles à deux états, tels IPHC/eCRTP ou ROHCv2.

Dans le cas de IPHC/eCRTP, il est alors possible de modéliser la fonction $P(\Delta)$ par le mécanisme TWICE, tout comme le W-LSB dans le paragraphe précédent. [61] et [39] ont montré que le taux de réussite de l'algorithme TWICE dépassait 83% suite à une unique perte. L'algorithme échoue si et seulement si une modification non prédictible est apparue ente temps. Cette condition pouvant, par nature, être considérée comme indépendante d'un paquet à l'autre du flux, il est raisonnable de modéliser la fonction de récupération du modèle par $P(\Delta) = \alpha^\Delta$ où α représente alors le taux de réussite de l'algorithme TWICE suite à une perte unique.

Pour le protocole ROHCv2, le modèle de récupération et de transitions est similaire à celui de ROHC. La taille des en-têtes du mode compressé du modèle étant alors initialisé avec la taille moyenne des en-têtes du mode CO observés.

3.7 Résultats de simulation

Dans un premier temps, nous effectuons une comparaison des trois modes du modèle, instancié par ROHC. Les longueurs des salves IR, FO et SO sont respectivement prises à 2, 5 et 100. La longueur respective de leurs en-têtes est prise à 40, 20 et 5 octets. Le débit du lien est de 100 ko/s et le ratio de transitions IR/FO pour le mode U est de $k = 5$. Pour modéliser le W-LSB, nous avons pris les paramètres $W = 14$ et $a = 0,9$. Afin de déterminer un comportement significativement différent entre les modes Optimiste et Fiable, le taux de pertes de la couche liaison FER est fixé à 5.10^{-2}.

Les Figures 3.8, 3.9 et 3.10 montrent le comportement des trois modes de la couche protocolaire de compression sur le ratio $\frac{PER}{FER}$, l'efficacité moyenne de la compression et le temps moyen de resynchronisation, en fonction de la longueur moyenne des salves de perte m et le délai aller-retour du lien (RTT).

Le premier résultat est que le mode Unidirectionnel est bien indépendant du RTT, à cause du manque de voie retour. Par conséquent, le mode Unidirectionnel devient le

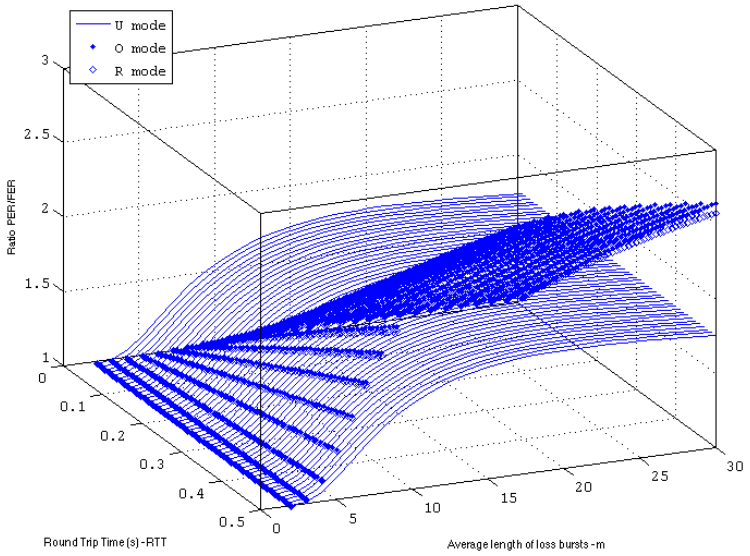

Figure 3.8 – Ratio PER/FER pour les trois modes de transmission, en fonction de m et du RTT

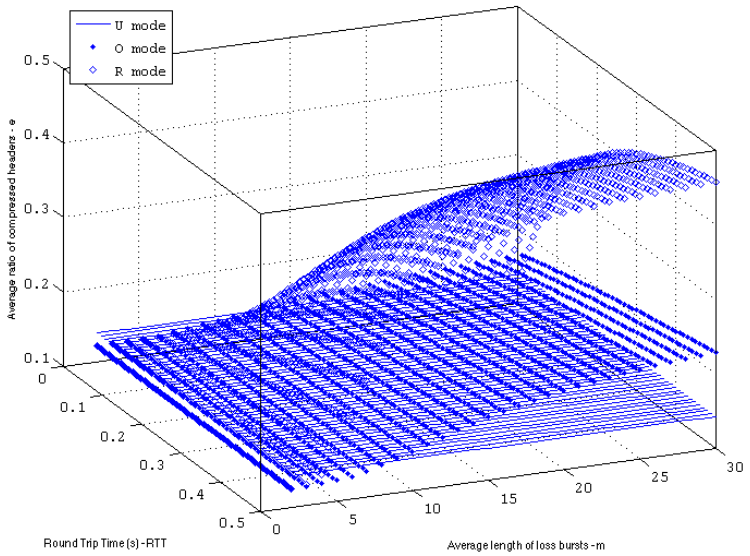

Figure 3.9 – Efficacité de la compression des trois modes de transmission, en fonction de m et du RTT

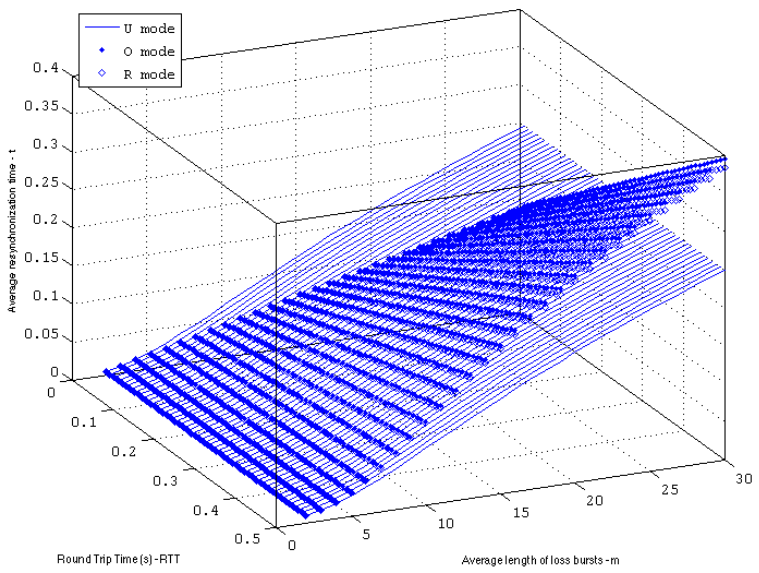

Figure 3.10 – Temps moyen de resynchronisation pour les trois modes de transmission, en fonction de m et du RTT

mode privilégié dès lors que le RTT dépasse 250ms, ou lorsque les longueurs de salves de perte sont importantes. Les deux autres modes sont alors pénalisés par leur manque de réactivité. La Figure 3.9 montre par ailleurs, qu'en plus d'être la plus performante sur ces longs RTT, la transmission unidirectionnelle est aussi bien plus efficace. Le mode Optimiste est pénalisé par ses envois répétés de salves FO lorsque les longueurs moyennes de pertes sont importantes. Ceci illustre le fait que des salves FO entières sont alors perdues, nécessitant l'envoi répété de nouvelles salves. Le mode Fiable est quant à lui encore plus pénalisé par le fait que lorsque ce RTT augmente, le nombre d'en-têtes FO transmis durant ce RTT devient prépondérant. La Figure 3.10 illustre le fait que, grâce à une bonne paramétrisation du protocole de compression, la durée moyenne de désynchronisation du mode Unidirectionnel peut-être inférieure à celle de mécanismes avec voie de retour.

Enfin, nous avons tenté de mettre en évidence l'intérêt d'un mécanisme de récupération efficace. Nous avons comparé le ratio entre le taux de perte en sortie et en entrée pour deux instantiations du modèle. La première est celle d'un protocole ROHC, muni d'un W-LSB de paramètres $W = 14$ et $a = 0,85$. La seconde est de type IPHC/CRTP basé sur l'algorithme TWICE de paramètre $\alpha = 0,85$. Les deux modèles ont été paramétrés pour avoir la même efficacité $e = 14,27\%$. L'influence de la longueur des salves de pertes dur le ratio $\frac{PER}{FER}$ est présenté sur la Figure 3.11.

Ces résultats mettent ici en évidence l'apport considérable du mécanisme W-LSB, qui permet grâce à sa fenêtre glissante, de contenir des salves de pertes modestes.

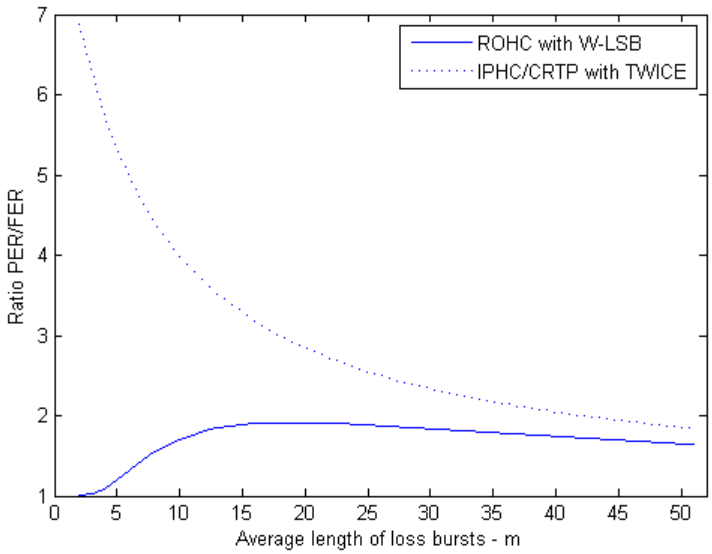

Figure 3.11 – Influence du mécanisme de récupération sur le taux d'erreur en sortie

Les deux mécanismes deviennent aussi peu efficaces dès lors que ces salves de pertes sont longues, ce qui illustre les limites des deux mécanismes.

3.8 Conclusion

Grâce à cette modélisation, nous avons pu montrer que, contrairement à une possible idée reçue, l'utilisation d'une voie de retour sur les mécanismes de compression d'en-têtes peut être contre-productif. En effet, dès lors que les temps aller-retour en jeu deviennent importants, non seulement les performances des modes O et R sont dégradées, mais aussi leur efficacité de compression, qui contrairement au mode Unidirectionnel, est asymptotiquement optimale en mode stationnaire. Ces mêmes résultats de simulation ont également mis en évidence l'impact de longues salves de pertes, qui dégradent fortement les performances de ces mêmes modes. De par sa structure, le mode U est alors plus à même de réagir rapidement à ces longues pertes.

Il est néanmoins nécessaire de ramener cette modélisation sur un plan pratique. Du point de vue des longueurs de salves de pertes, les valeurs que nous avons pu expérimenter correspondent typiquement aux longueurs des pertes dans un réseau comprenant des nœuds congestionnés.

Quant aux RTT pris en compte dans nos simulations, rappelons que celui d'un satellite géostationnaire est de l'ordre de 0.5s. Par ailleurs, dans un contexte satellitaire, la voie de retour peut être absente ou se révéler coûteuse. Des études récentes [62] ont montré que le RTT d'une liaison de type Internet peut également atteindre plus de 100ms, et dans ce cas rend caduque l'utilisation d'une voie de retour. Dans les réseaux

cellulaires, où par exemple, ROHC fait partie intégrante de la norme UMTS, les délais en jeu sont également importants [63]. Les résultats expérimentaux ont ainsi montré que le RTT des réseaux GPRS pouvait atteindre 1s et 250ms pours des réseaux de type UMTS.

Pour des scénarios de type WiFi, les délais en jeu sont faibles et les voies de retour accessibles simplement. Ainsi pour ce type de connexion, les modes O et R pourront être privilégiés. Cependant, les conditions sont différentes pour des réseaux de type WiMax, où la topologie même du réseau entraîne des latences pouvant varier de moins de 150ms à plus de 500ms [64] [65]. Avec un taux de perte trame inférieur à 1%, la voie de retour ne semble ici pas utile.

Cette modélisation permet alors de mettre en lumière les enjeux d'une paramétrisation du protocole de compression en fonction du scénario envisagé.

Chapitre 4

Codes à effacements à décodage hybride avec matrice génératrice bande

Le 14 juillet c'est la semaine du 14.

Théorème toulousain.

Sommaire

4.1 Introduction

Après le chapitre précédent consacré à la compression d'en-têtes, ce chapitre présente la première des trois contributions concernant les codes à effacements.

Les codes que nous présentons dans ce chapitre, que nous appellerons codes à matrice bande, sont des codes binaires linéaires systématiques construits pour pouvoir être décodés efficacement de deux manières. Un décodage par le maximum de vraisemblance peut être utilisé sur la matrice génératrice et un décodage de type itératif peut être utilisé sur la matrice de parité associée. Chacun des deux décodages comporte ses avantages et des inconvénients. Le décodage à maximum de vraisemblance est basé sur

un algorithme de type élimination Gaussienne. Ce décodage permet donc d'atteindre la capacité de décodage du code, mais ceci se fait au détriment d'une complexité mathématique élevée $O(n^3)$. En ce sens, le décodage gaussien n'est pas adaptable dès lors que la taille des codes est supérieure à quelques centaines de symboles. Le décodage itératif quant à lui, permet de décoder simplement avec une complexité linéaire, mais atteint des capacités de récupération bien moindres. Dans ce chapitre, nous emploierons le terme de décodage hybride dans le sens où les deux décodages sont indépendants pour ce type de codes.

Ce type décodage hybride n'est en soit pas une nouveauté. Par exemple, en utilisant des codes de type Low Density Parity-Check (LDPC) qui procèdent à un décodage itératif sur une matrice de parité creuse, il est tout à fait possible de procéder à une élimination Gaussienne sur la matrice génératrice associée. En ce sens, nous avons affaire à un décodage hybride. De la même manière, lorsque le décodage LDPC échoue à une étape intermédiaire, il est possible de basculer sur un décodage à maximum de vraisemblance sur les éléments restants.

Deux leviers permettent d'agir sur les performances d'un décodage hybride basé sur ces deux types de décodage. Le premier consiste à améliorer la capacité de correction liée au décodage itératif. Ce mécanisme est difficile à maîtriser car il s'agit d'améliorer les dépendances inter-symboles en limitant au maximum l'ajout de ces relations qui ont pour conséquence de remplir la matrice de parité, et donc de diminuer la vitesse du décodage itératif. Le remplissage de cette matrice de parité sera toujours un compromis entre la capacité et la vitesse du décodage itératif. Puisque le décodage gaussien est optimal pour un code donné, le deuxième levier consiste à restreindre l'ensemble des codes possibles à l'ensemble des codes pour lequel un décodage à maximum de vraisemblance adapté peut être utilisé. Le but ici est d'obtenir un décodage optimal avec une complexité mathématique moindre, sans affecter la capacité des codes.

Le codage présenté ici utilise ce second levier. L'idée est de restreindre la partie non-systématique de la matrice génératrice du code dans une bande de largeur b. En dehors de cette bande, les éléments de la partie non systématique sont tous nuls, ce qui signifie en outre que chaque symbole de redondance est une combinaison d'un nombre limité de symboles sources voisins. En appliquant cette restriction, il est possible d'effectuer une élimination gaussienne adaptée avec une complexité $O(b^2 \times n)$ où $b \ll n$. L'intérêt de ce décodage a été montré par Studholme et Blake [66] [67] sur une matrice génératrice aléatoire bande avec une largeur $2\sqrt{k}$.

Le principal point ici est d'obtenir une matrice génératrice avec une partie non-systématique réduite à une bande qui possède une matrice de parité associée creuse permettant un décodage itératif efficace.

Ces codes ont été développés en collaboration avec Mathieu Cunche et Vincent Roca de l'INRIA Grenoble.

$$T = \begin{pmatrix} t_0 & 0 & 0 & \dots & & 0 & 0 \\ t_1 & t_0 & 0 & & & & \\ t_2 & t_1 & t_0 & & & \vdots & \vdots \\ \vdots & t_2 & t_1 & \ddots & & & \\ t_i & \vdots & t_2 & \ddots & \ddots & 0 & \\ \vdots & t_i & \vdots & & & t_0 & 0 \\ t_{n-1} & \vdots & t_i & & & t_1 & t_0 \end{pmatrix}$$

Figure 4.1 – Matrice de Toeplitz triangulaire inférieure

4.2 Construction des codes à matrice bande

4.2.1 Définitions et propositions

Nous définissons ici quelques éléments nécessaires à la construction des codes à matrice bande.

Définition 4.1. *Soit $T = (t_{i,j},\ i,j \in \{0,...,n-1\})$ une matrice de Toeplitz carrée triangulaire inférieure de taille n. Cette matrice est donc à diagonale constante. Pour $i < j$, $t_{i,j} = 0$ et pour $i \geq j$, $t_{i,j} = t_{i-j}$. Cette matrice peut donc être définie par le n-uplet $(t_0, t_1, ..., t_{n-1})$.*

A ce n-uplet on associe le polynôme suivant $t(x) \in K[X]/(X^n)$, soit l'anneau de l'ensemble des polynômes à degré inférieur à n modulo X^n et à coefficients sur le corps K :

$$t(x) = \sum_{i=0}^{n-1} t_i x^i$$

Cette matrice est représentée comme sur la Figure 4.1 :

Proposition 4.2. *Soit $G \in \mathcal{M}_{n,m}$ une matrice à n lignes à coefficients dans $K[X]$, et soit $g_j(x)$ les polynômes associés aux colonnes de G. Ces polynômes peuvent s'écrire $\forall j \in (0,...,m-1), g_j(x) = \sum_{i=0}^{n-1} g_{i,j} x^i$.*
Soit T une matrice de Toeplitz telle que définie précédemment et t(x) son polynôme associé dans l'anneau défini précédemment. Les colonnes de la matrice du produit TG sont égales au produit des polynômes associés $t(x)g_j(x)$.

Ce résultat peut être résumé comme suit :

$$\begin{pmatrix} t_0 & & & \\ t_1 & t_0 & & \\ & & \ddots & \\ t_{n-1} & t_{n-2} & & t_0 \end{pmatrix} \times \begin{pmatrix} g_{0,0} & g_{0,1} & g_{0,2} & \cdots & g_{0,m-1} \\ g_{1,0} & g_{1,1} & g_{1,2} & \cdots & g_{1,m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ g_{n-1,0} & g_{n-1,1} & g_{n-1,2} & \cdots & g_{n-1,m-1} \end{pmatrix}$$
$$= \begin{pmatrix} (t.g_0)_0 & (t.g_1)_0 & \cdots & (t.g_{m-1})_0 \\ (t.g_0)_1 & (t.g_1)_1 & \cdots & (t.g_{m-1})_1 \\ \vdots & \vdots & \vdots & \vdots \\ (t.g_0)_{n-1} & (t.g_1)_{n-1} & \cdots & (t.g_{m-1})_{n-1} \end{pmatrix}$$

Démonstration. Le produit matriciel pour chaque élément de la matrice donne :

$$\begin{aligned} TG_{i,j} &= \sum_{k=0}^{n-1} t_{i,k} g_{k,j} \\ &= \sum_{k=0}^{i} t_{i,k} g_{k,j} + \sum_{i+1}^{n} 0 \times g_{k,j} \\ &= \sum_{k=0}^{i} t_{i-k} g_{k,j} \end{aligned}$$

D'un autre côté le produit des polynômes de chaque colonne donne :

$$t(x)g_j(x) = (\sum_{i=0}^{n-1} t_i x^i)(\sum_{i=0}^{n-1} g_{i,j} x^i) \mod x^n = \sum_{i=0}^{n-1} (\sum_{k=0}^{i} t_{i-k} g_{k,j}) x^i$$

Les coefficients des polynômes produits associés sont bien les coefficients de la matrice produit. □

4.2.2 Schéma général du code

Nous considérons un code binaire systématique de dimension k et de longueur n. La matrice génératrice associée à ce code G est de taille $k \times n$, et la matrice de parité H est de taille $(n-k) \times n$. Soit $M \in \mathcal{M}_{k,n-k}$ la partie non-systématique de la matrice génératrice. Soit $A \in \mathcal{M}_{n-k,k}$ la matrice formée des k premières colonnes de la matrice de parité et U la matrice carrée de taille $n-k$ formant les dernières colonnes de H. Les deux matrices peuvent ainsi s'écrire sous la forme suivante :

$$G = (Id_k | M) \qquad H = (A|U)$$

Nous prenons l'hypothèse que la matrice U est une matrice de Toeplitz triangulaire inférieure telle que définie dans le paragraphe 4.2.1. Ceci permet de se ramener à un cas similaire à celui des codes LDPC-Triangle tels que définis dans la RFC5170 [29].

En effet, en gardant une structure triangulaire inférieure, tout comme les codes LDPC-Triangle, un décodage itératif à complexité linéaire est possible. La matrice U est donc déterminée par le (n-k)-uplet $(1, u_1, u_2, ..., u_{n-k-1})$. La matrice U étant triangulaire avec une diagonale égale à 1, elle est donc de rang plein et inversible. Comme décrit précédemment, nous associons cette matrice avec le polynôme $u(x)$:

$$u(x) = 1 + \sum_{k=1}^{n-k-1} u_k x^k$$

En résumé, la matrice U possède la forme suivante :

$$U = \begin{pmatrix} 1 & 0 & 0 & \dots & & 0 & 0 \\ u_1 & 1 & 0 & & & & \\ u_2 & u_1 & 1 & & & \vdots & \vdots \\ \vdots & u_2 & u_1 & \ddots & & & \\ u_i & \vdots & u_2 & & \ddots & 0 & \\ \vdots & u_i & \vdots & & & 1 & 0 \\ u_{n-k-1} & \vdots & u_i & & & u_1 & 1 \end{pmatrix}$$

Pour la construction de la matrice $M \in \mathcal{M}_{k,n-k}$, l'objectif est de restreindre l'ensemble des éléments non nuls de la matrice à une bande. Nous noterons la largeur de cette bande B. Dans un premier temps, nous pouvons écrire la matrice M sous la forme suivante :

$$M = \begin{pmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,B-1} & 0 & \dots & 0 & 0 \\ 0 & m_{1,0} & \ddots & \dots & m_{1,B-1} & 0 & \dots & 0 \\ \vdots & 0 & \ddots & \ddots & \dots & \ddots & 0 & \vdots \\ \vdots & \vdots & 0 & m_{k-2,0} & \dots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & 0 & m_{k-1,0} & \ddots & \dots & m_{k-1,B-1} \end{pmatrix}$$

A chaque ligne de la matrice nous pouvons associer le polynôme $m_i(x) = \sum_{j=0}^{B-1} m_{i,j} x^j$. Dans ce cas particulier, nous avons une relation entre la dimension et la longueur du code, due à la contrainte de la bande :

$$n - k = k + B - 1$$

Nous verrons par la suite comment s'affranchir de cette contrainte afin de construire des codes avec des dimensions indépendantes de B.

Par définition, la matrice génératrice G envoie n'importe quel vecteur source binaire dans l'espace vectoriel image formé de l'ensemble des mots du code. De la même manière la matrice de parité associée H est une application linéaire dont le noyau est

$$H = \left(\begin{array}{cccc|cccccccc} a_{0,0} & & & & 1 & & & & & & & \\ a_{0,1} & a_{1,0} & & & u_1 & 1 & & & & & & \\ a_{0,2} & a_{1,1} & a_{2,0} & & u_2 & u_1 & 1 & & & & & \\ a_{0,3} & a_{1,2} & a_{2,1} & & u_3 & u_2 & u_1 & \ddots & & & & \\ \vdots & a_{1,3} & a_{2,2} & \ddots & \vdots & u_3 & u_2 & \ddots & \ddots & & & \\ & \vdots & a_{2,3} & & & \vdots & u_3 & & \ddots & & & \\ & & \vdots & & & & \vdots & & \ddots & 1 & & \\ & & & & & & & & & u_1 & 1 & \end{array}\right)$$

Figure 4.2 – Matrice de parité associée à une matrice génératrice bande, la partie de gauche est bande, la partie de droite est de Toeplitz triangulaire inférieure

l'ensemble des mots du même code. Ces deux matrices sont orthogonales. Nous avons donc la relation suivante :

$$HG^T = 0$$

Grâce aux définitions précédentes, cette relation se transforme en :

$$(A|U)(\frac{Id_k}{M^T}) = 0 \Leftrightarrow A \oplus UM^T = 0 \Leftrightarrow A = UM^T$$

Ce résultat permet de donner une relation importante entre la matrice bande et la matrice de parité. D'après la proposition 4.2, et d'après la construction de U prise en hypothèse, on en déduit la relation entre les colonnes de la matrice A et leurs polynômes associés $a_i(x)$, la matrice U et les lignes de la matrice M, $m_i(x)$ pour tout $i \in (0, 1, ...k-1)$.

$$\forall i \in (0, 1, ..., k-1), \qquad a_i(x) = u(x)m_i(x)$$

Grâce à cette relation, il va ainsi être possible de construire des matrices génératrices à structure bande en tenant compte de la matrice de parité. Une particularité de ce résultat est d'ailleurs de nous indiquer la structure de la matrice de parité associée. En effet, si U est une matrice de Toeplitz triangulaire inférieure, alors A aura également une structure restreinte à une bande, comme représenté sur la Figure 4.2.

Nous rappelons que le poids de Hamming d'une séquence est le nombre d'éléments non nuls de celle-ci. De manière similaire, le poids de Hamming d'un polynôme $p(x)$ est défini par le nombre de monômes non nuls et sera noté $W(p)$.

4.2.3 Construction des matrices

Afin de supporter un décodage itératif sur la matrice de parité efficace, il est nécessaire que celle-ci soit creuse. En d'autres termes, le nombre d'éléments non nuls par colonne de la matrice devra être faible. En utilisant la représentation polynômiale

détaillée dans le paragraphe précédent, cela revient à dire que le poids de Hamming des polynômes $a_i(x)$ et $u(x)$ doit être faible. En pratique, nous définissons une distribution de petits degrés sur les a_i permettant un décodage efficace. Nous verrons par la suite comment le poids des lignes peut être géré lorsque les polynômes a_i sont sélectionnés.

Du point de vue du décodage à maximum de vraisemblance sur la matrice génératrice, la seule contrainte consiste à conserver l'ensemble des éléments non nuls dans une bande de largeur B. D'un point de vue polynomial, cela se traduit par une limitation des polynômes $m_i(x)$ à des degrés strictement inférieurs à B. Cependant, il est nécessaire d'utiliser les polynômes de degrés aussi proches que possible de B afin de permettre à chaque bit du mot encodé d'être la somme d'un ensemble de bits sources le plus varié possible, et ainsi se rapprocher des codes bandes aléatoires.

Le point crucial de la construction de ce code est donc de déterminer un ensemble de polynômes $\{a_i(x), m_i(x), u(x)\}$ tel que pour tout $i \in (0, 1, ...k-1)$, $a_i(x) = u(x)m_i(x)$ et tel que $u(x)$ et chaque $a_i(x)$ soit de faible poids de Hamming, et que chaque $m_i(x)$ soit de degré strictement inférieur à B. L'ensemble de ces polynômes pour un $u(x)$ donné sera nommé l'ensemble des *polynômes candidats*. Ces polynômes peuvent être recherchés de manière exhaustive, et dans le cadre de nos expérimentations, l'algorithme implémenté en MAPLE^{TM}n'excédait pas quelques secondes pour des largeurs de bande de 200.

Les résultats expérimentaux ont montré que choisir un seul et unique polynôme $m(x)$ pour l'ensemble des lignes de M dégradait fortement la capacité de correction du code. C'est pour cela qu'à partir de ce point, nous choisirons un ensemble de polynômes $m_i(x)$ et donc un ensemble de polynômes $a_i(x)$. Ceci peut s'expliquer par la régularité ainsi engendrée par l'utilisation d'un seul polynôme. Cependant, de la même manière, les résultats expérimentaux ont aussi montré que la disposition des dits polynômes dans la manière n'avait que peu d'influence sur les performances du code en comparaison de la sélection de l'ensemble de ces polynômes. Nous avons donc pris une approche aléatoire pour la disposition des polynômes candidats.

Néanmoins, cette approche ne permet pas de liberté sur la distribution des degrés des lignes de la matrice de parité. Il est donc possible d'ajuster cette distribution en effectuant des permutations sur les colonnes et les polynômes de la matrice A. En conséquence, il sera possible de s'approcher d'une distribution des lignes donnée sans pour autant modifier celle des colonnes ou modifier la capacité de correction du code. Il est cependant important de noter que malgré ce degré de liberté, il n'est pas garanti qu'une distribution cible de degrés lignes puisse être atteinte grâce à ce mécanisme.

4.2.4 Adaptation et optimisations

4.2.4.1 Indépendance par rapport à la largeur de la bande

La structure présentée précédemment apporte deux principaux problèmes. Le premier est que la dimension et la longueur du code sont liées par la taille de la bande : $n = 2k + B$. Le second, qui découle en partie du premier est que les premiers et derniers symboles de redondance non systématiques ne protègent qu'un ensemble très réduit

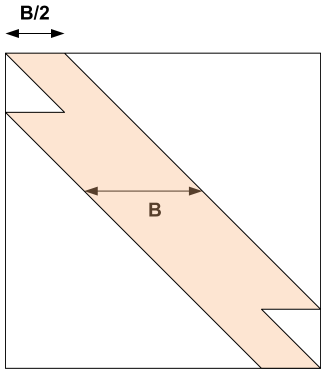

Figure 4.3 – Matrice M de largeur B dont les premières et dernières lignes ont été remplacées par des lignes de largeur de bande B/2

de symboles d'entrée. A l'extrême, le premier et le dernier symbole de redondance ne protègent qu'un seul symbole source.

Nous avons résolu ce problème en utilisant un second ensemble de polynômes, de degré maximal $B/2$. En utilisant ce second ensemble sur les $B/2$ premières et dernières lignes de M, la bande est conservée. Le nouveau ratio de codage est alors indépendant de B : $n = 2k$, et les symboles aux limites de M protègent désormais plus de symboles sources. La représentation de la matrice M avec ce mécanisme se trouve sur la Figure 4.3.

Nous avons également exploré la possibilité d'utiliser de nombreux ensembles de polynômes de degrés maximaux respectifs $B/2, B/2+1, B/2+2, \ldots$ afin de compléter chaque ligne par des polynômes de degrés adaptés. L'effet escompté était de "combler" la partie triangulaire laissée vide par le mécanisme précédent sur les symboles extrêmes (tel que nous pouvons le voir sur la Figure 4.3. Les résultats expérimentaux ont cependant montré que l'impact sur les performances du code est minime.

Nous avons également étudié la possibilité d'éliminer purement et simplement les $B/2$ premières et dernières colonnes de M, mais ceci posait un problème, notamment lors de la génération de la matrice de parité. En effet, dans ce cas, il n'est plus possible de générer directement celle-ci à partir de la matrice génératrice. Par ailleurs, l'élimination des lignes (dues à la transposée de M) dans la matrice de parité n'est pas envisageable car les colonnes de celle-ci sont peu remplies et peuvent potentiellement devenir vides.

4.2.4.2 Adaptation à des différents taux de codage

Grâce au mécanisme précédent, nous avons pu rendre le ratio de codage indépendant de la largeur de bande. Cependant, en l'état, celui-ci reste fixé à $\frac{1}{2}$. Une seconde adaptation consiste donc à permettre l'utilisation d'un rendement quelconque. Pour cela, il suffit d'ajuster le décalage de positions entre une ligne de la matrice M et la suivante. Par défaut celui-ci étant fixé à 1, il permet un rendement de $\frac{1}{2}$. Soit f ce

décalage et R le ratio du code. En posant $f = \frac{1}{R} - 1$, nous pouvons prendre en charge des rendements différents tout en conservant la largeur de bande désirée. Si la valeur f n'est pas entière, il suffit d'utiliser une famille $\{f_i\}$ d'entiers tels que le décalage moyen soit f. Par ailleurs, il est nécessaire d'adapter le mécanisme précédent en remplaçant les les $\frac{B}{2f}$ premières et dernières lignes de M par des lignes construites à partir de polynômes de degré $B/2$.

4.2.4.3 Vers une analyse de l'évolution de densité ?

La faisabilité d'une étude de type évolution de densité [68] pour le décodage itératif de la matrice de parité a été étudiée, mais a cependant fait face à de nombreux obstacles.

Le premier point est qu'afin d'effectuer cette analyse, le code étudié doit théoriquement être de longueur infinie. Comme nous le verrons par la suite, la largeur de la bande nécessaire pour effectuer un décodage efficace augmente lorsque la longueur du code augmente. Il faudrait donc une largeur de bande infinie afin d'effectuer une analyse avec un code de longueur infinie. Par ailleurs, si l'on élimine cette hypothèse, l'hypothèse des cycles de longueur infinie tombe elle aussi, car à cause de la structure bande, chaque nœud de contrôle ne peut-être connecté qu'à un ensemble délimité de nœuds sources.

Par ailleurs, si la distribution des degrés des colonnes peut-être facilement ajustée, nous avons vu que des limites apparaissaient sur la distribution des degrés des lignes de la matrice de parité.

Nous avons tenté de nous rapprocher autant que possible de schémas de répartition connus, mais sans résultat notable.

L'ensemble de ces points fait que cette approche a été étudiée, mais étant considérée comme difficilement adaptable, du fait des contraintes de la bande, nous nous sommes concentrés sur une approche empirique pour la sélection des polynômes.

4.3 Analyse théorique

4.3.1 Capacités de correction théorique

Décodage à maximum de vraisemblance

Le décodage optimal du code, à maximum de vraisemblance sur la matrice de parité est dépendant de la largeur de bande. En effet, le code utilisé sera un compromis entre une largeur de bande suffisamment grande, pour permettre une plus grande dispersion du code, et une largeur suffisamment raisonnable pour permettre un décodage rapide.

Dans la présentation de leurs codes bande aléatoires, Studholme et Blake ont étudié de manière théorique, l'influence de la largeur de la bande sur la capacité de correction de leur codes. En conclusion, il s'avère qu'une largeur de bande de $2\sqrt{k}$ et composée d'au moins de $2 \log k$ éléments non nuls par colonne permettait d'obtenir des capacités de correction proches de celles de matrices aléatoires classiques. Nous

pouvons considérer notre travail comme un cas particulier de ces codes avec des contraintes supplémentaires sur le remplissage de la bande et des relations inter-lignes. Nous avons ainsi considéré cette borne comme la contrainte minimale de notre code pour un décodage optimal.

Décodage itératif

De la même manière que le décodage à maximum de vraisemblance, la contrainte bande va réduire la diversité des connexions inter-symboles. En conséquence, les performances attendues seront inférieures à celles des codes Repeat-Accumulate dont ils tiennent leur structure. Il ne faut d'ailleurs pas négliger l'importance de la régularité de la matrice U sur la capacité de correction du décodage itératif.

Une code à matrice bande dont la largeur augmentera se rapprochera d'un code Repeat-Accumulate, qui possède de bonnes performances. Alors qu'une largeur de bande faible sur A se rapprochera d'un code à répétition qui possède une mauvaise capacité de correction.

4.3.2 Complexité théorique

Comme présenté précédemment, le décodeur itératif fonctionne avec une complexité linéaire $\mathcal{O}(k)$. Lorsque celui-ci échoue, un décodage Gaussien adapté aux bandes est utilisé sur la matrice génératrice. Il est important de rappeler que dans le cas général, la complexité d'un tel décodage est $\mathcal{O}(k^3)$.

La matrice génératrice globale du code étant systématique, lorsque un élément systématique est reçu, il convient d'éliminer la ligne correspondante dans le décodage de la matrice M. De la même manière, seules les colonnes de M dont le symbole a été reçu sont utilisées pour le décodage. Le décodage gaussien s'effectue donc sur une sous-matrice de la matrice bande M. Il est important de noter que si M est une matrice de bande B, alors toute sous-matrice de M est de bande M. Dans le cas général et pour un rendement R, la matrice à inverser est proche de la taille $kR \times k(1/R-1)(1-R)$ soit $(k(1-R))^2$. En utilisant un codage adapté, tel que défini dans [69], le décodage Gaussien tombe donc a une complexité équivalente à $\mathcal{O}(kB^2)$.

A titre d'exemple, pour $k = 2000$ et une bande de largeur $B = 200$, le gain théorique est de l'ordre de 100 en comparaison d'un décodage Gaussien classique.

4.4 Résultats obtenus

4.4.1 Méthodologie de test

Des simulations ont été effectuées afin d'étudier le bénéfice pratique de ces codes en termes de vitesse de décodage et l'impact sur la capacité de correction du décodage itératif de cette structure bande. L'architecture de test est un Quad-Core Intel Xeon 5120 @ 1.86Ghz / 4GB RAM sous Linux. Un décodeur hybride ML/itératif a été utilisé. Les codes à matrice bande ont été comparés à deux autres types de codes :

- Les codes LDPC-Staircase, tels que défini dans le standard. Ce type de code a été choisi car il est de la même famille que les codes à matrice bande et supporte aussi un décodage de type hybride. L'intérêt est de comparer l'impact de la bande sur le décodage itératif par rapport à un code standard possédant une très bonne capacité de correction itérative. Il s'agit aussi d'étudier le gain de vitesse pratique obtenu par rapport à un décodage Gaussien classique. Pour ce code, le nombre d'éléments par colonne de la matrice de parité a été fixé à 5.
- Les codes "Windowed Erasure" de Studholme et Blake, en utilisant les paramètres recommandés par les auteurs. Ces codes ne supportent pas de décodage itératif et ont été adaptés car il ne sont pas systématiques par défaut. Ces codes permettront d'évaluer l'impact des contraintes de la prise en charge d'un décodage hybride sur la capacité de correction par rapport à un code totalement aléatoire.

La méthodologie d'obtention des matrices bandes est celle de la section 4.2.

4.4.2 Capacités de correction

Les tables 4.1 et 4.2 présentent l'inefficacité moyenne des codes à ratio $\frac{1}{2}$pour les décodages itératif et à maximum de vraisemblance pour $k = 1000$ et $k = 2000$ symboles.

Type de décodage	IT	ML
LDPC-Staircase $N_1 = 5$	14.24%	1.21%
LDPC-Band - B = 100	18.39%	2.97%
LDPC-Band - B = 200	14.75%	1.24%
Windowed Erasure - b = 63	-	0.17%

Table 4.1 – Inefficacité moyenne en fonction du décodeur utilisé, k=1000, R=$\frac{1}{2}$

Type de décodage	IT	ML
LDPC-Staircase $N_1 = 5$	13.95%	1.15%
LDPC-Band - B = 200	16.23%	1.19%
Windowed Erasure - b = 89	-	0.79%

Table 4.2 – Inefficacité moyenne en fonction du décodeur utilisé, k=2000, R=$\frac{1}{2}$

Comme évoqué lors de l'analyse théorique, la largeur de la bande a une influence sur la capacité de correction des deux décodages. Pour ces dimensions, et avec une largeur de bande $B = 200$, les codes à matrice bande permettent d'obtenir un décodage proche de celui des codes LDPC-Staircase en itératif, tout en ayant une capacité de correction quasi-équivalente.

L'écart avec les codes Windowed Erasure montre bien que les contraintes sur les

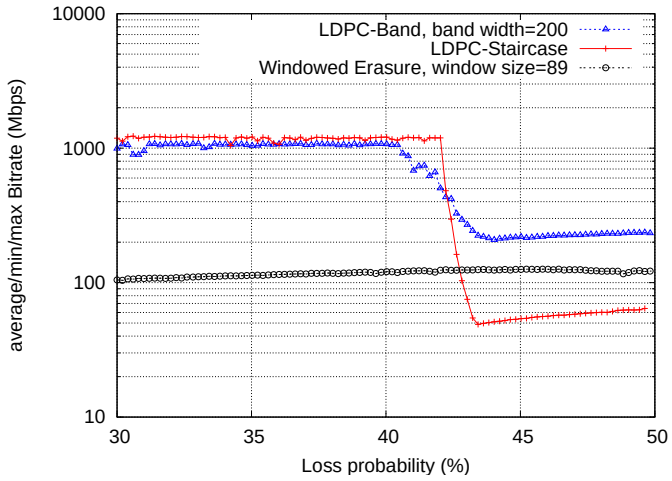

Figure 4.4 – Vitesse de décodage moyenne, k=2000 symboles, symboles de 1024 octets, rendement $\frac{1}{2}$

matrices bandes ont un impact non-négligeable sur la capacité de correction intrinsèque du code.

4.4.3 Vitesses de décodage

Sur la Figure 4.4, nous avons représenté la vitesse de décodage en fonction du taux de perte introduit dans le canal, pour un code de dimension $k = 2000$ et un rendement de $\frac{1}{2}$. Un code optimal pourra donc supporter jusqu'à 50% de pertes. Ici un décodage hybride est utilisé : lorsque le décodage itératif échoue, le décodage ML prend le relais et impacte donc la vitesse moyenne de décodage.

Nous pouvons effectivement remarquer deux zones de vitesse bien différentes pour les codes hybrides. Dans la zone autour de 1Gbps, le décodage itératif fonctionne. Dès lors que le taux de perte augmente, le décodage itératif ne suffit plus, et nous pouvons observer un second palier de vitesse correspondant au décodage ML, qui est plus élevé pour les matrices bandes que pour les LDPC-Staircase. Ceci permet de mettre en lumière l'intérêt de ce décodage adapté. Le fait que le décodage de la matrice bande "chute" avant celui du LDPC-Staircase s'explique par la capacité moindre du décodage itératif comme mis en évidence dans le paragraphe précédent. Il convient donc de relativiser les résultats du paragraphe précédent au vu de ces résultats.

Dans tous les cas, le décodage des Windowed Erasure est plus lent que celui des matrices bandes.

Le tableau 4.3 détaille les vitesses de décodage à maximum de vraisemblance pour plusieurs tailles de code. Ceci permet de mettre en évidence la vitesse sous-cubique du décodage sur la matrice bande, alors que les deux autres décodages suivent une

véritable complexité cubique. Ceci permet d'obtenir un gain de vitesse d'un facteur 2 par rapport aux codes bandes aléatoires et jusqu'à 4 par rapport aux LDPC-Staircase.

k	1000	2000	4000
LDPC-Band	326 Mbps	235 Mbps	150 Mbps
LDPC-Staircase	125 Mbps	60 Mbps	30 Mbps
Windowed Erasure	220 Mbps	120 Mbps	68 Mbps

Table 4.3 – Vitesse de décodage ML moyenne en fonction de la dimension du code, symboles de 1024 octets, rendement $\frac{1}{2}$

Les résultats équivalents pour le décodage itératif sont fournis sur la Table 4.4. On remarque ainsi que l'impact de la restriction à une bande n'a qu'un impact limité sur la vitesse de décodage itérative.

k	1000	2000	4000
LDPC-Band	1100 Mbps	1050 Mbps	900 Mbps
LDPC-Staircase	1300 Mbps	1200 Mbps	1000 Mbps

Table 4.4 – Vitesse de décodage itérative moyenne en fonction de la dimension du code, symboles de 1024 octets, rendement $\frac{1}{2}$

4.5 Conclusion

Dans ce chapitre, nous avons présenté une construction de codes permettant un décodage itératif adapté sur la matrice de parité, tout en permettant un décodage au maximum de vraisemblance adapté sur la matrice génératrice grâce à une structure bande. Ceci permet de réduire la complexité théorique du décodage ML à $\mathcal{O}(kB^2)$, où B représente la largeur de la bande.

Les résultats de simulation obtenus ont permis de confirmer le gain théorique de ce type de code. Ces codes bandes permettent d'obtenir un décodage itératif sensiblement équivalent à celui de codes LDPC classiques tout en obtenant un gain pratique de vitesse jusqu'à 4x dans le cadre de canaux fortement perturbés.

Cependant, la nature intrinsèque de ces codes rend leur adaptabilité faible, comme nous avons pu le voir lors de l'analyse de l'évolution de densité. Les travaux futurs concernant ces codes devront probablement se concentrer sur un autre aspect de ces codes, leur potentiel semblant limité par leur rigidité d'un point de vue classique.

Dans ce contexte, le fait que la partie gauche de la matrice de parité soit aussi une matrice bande n'a pas été exploité en l'état. A l'aide de quelques modifications, il semble possible d'adapter un décodage gaussien optimisé bande, directement sur la matrice de parité, ce qui aurait pour conséquence de libérer une contrainte sur ces

codes, et donc de permettre une amélioration des performances encore plus significative.

Chapitre 5

Codes de Reed-Muller pour le canal à effacements de paquets

There's a difference between knowing the path and walking the path.

Morpheus.

Sommaire

5.1 Introduction

Dans le cadre des codes à effacements, lorsque les tailles de bloc sont inférieures à quelques centaines d'éléments, deux types de codes peuvent être utilisés. Tout d'abord, les codes de Reed-Solomon, les plus couramment utilisés dans ce cas, permettent un codage optimal. Malheureusement ceux-ci peuvent être pénalisés par une complexité d'encodage/décodage relativement importante, par rapport à la taille des codes utilisés, lorsque les contraintes matérielles ou temporelles sont importantes. Des codes de type LDPC de complexité linéaire peuvent aussi être déployés. Cependant, si leur complexité est faible, ces codes ne sont pas adaptés pour des tailles de codes faibles et souffrent d'une inefficacité importante.

Dans ce chapitre, nous présentons un encodeur/décodeur basé sur les codes de Reed-Muller [70] [71]. Ceux-ci ont le double avantage d'être de faible complexité et d'avoir une capacité théorique proche de celle des codes MDS. La plupart des travaux concernant ces codes concernent le canal à erreur [72], mais leur complexité relativement faible les place également comme des candidats intéressants pour le canal à effacements [73]. Pour ces codes, il existe un décodage de complexité mathématique logarithmique basé sur une décomposition récursive du code, appelée décomposition

de Plotkin [74], cependant dans sa version classique, celle-ci souffre d'une capacité de correction médiocre rendant son application pratique caduque. Une technique largement étudiée pour le canal à erreur consiste à effectuer un décodage par permutations, qui permet d'améliorer de manière substantielle la capacité du code, tout en permettant de conserver la structure récursive du décodage.

Nous proposons donc d'adapter ce décodage récursif par permutations au canal à effacements, auquel nous ajoutons des mécanismes supplémentaires, respectivement un mécanisme de *remontée partielle* et un *décodage à blanc*. Nous étudierons l'impact de l'ensemble de ces mécanismes sur la capacité et la complexité du code, que nous comparerons à un décodage par maximum de vraisemblance (ML).

5.2 Présentation du code

Dans cette partie, nous présentons le principe des codes de Reed-Muller et les améliorations proposées pour l'adaptation au codage à effacements.

5.2.1 Présentation des codes de Reed-Muller

Les codes de Reed-Muller peuvent être définis de plusieurs manières, la méthode la plus directe étant la suivante :

Définition 5.1. *(Codes de Reed-Muller) Le code de Reed-Muller binaire de paramètres r et m où $0 \leq r \leq m$, correspond à l'espace vectoriel formé par les vecteurs des valeurs des fonction binaires à m variables et de degré strictement inférieur à r. Ce code sera noté $RM(r, m)$.*

Proposition 5.2. *La dimension d'un code de Reed-Muller $RM(r, m)$ est $k = \sum_{i=0}^{r} \binom{m}{i}$ et sa longueur est $n = 2^m$.*

Démonstration. La dimension k d'un code de Reed-Muller, par définition, correspond au cardinal de toute base génératrice de fonction binaire à m variables et de degré strictement inférieur à r. Soit $(x_0, x_1, ...x_{m-1})$ l'ensemble des m variables. Il est possible de choisir la base formée des monômes de toutes les combinaisons de x_i comportant moins de r variables.

Soit $0 \leq i \leq m$, l'ensemble des monômes distincts comprenant i variables parmi m. Par analogie combinatoire, ceci revient à dénombrer l'ensemble des choix de i éléments possibles parmi m. Cette valeur est donc égale à $\binom{m}{i}$.

La base des fonctions à m variables de degré inférieur à r étant formée de l'ensemble des monômes à i variables parmi m et ceci pour $0 \leq i \leq m$, la dimension du code est donc $k = \sum_{i=0}^{r} \binom{m}{i}$.

La longueur du code étant définie par la longueur de chaque élément de la base, elle correspond à l'ensemble des valeurs binaires prises par les x_i et est donc de taille 2^m. □

Une méthode sensiblement identique permet de construire de manière plus simple la matrice génératrice d'un code de Reed-Muller :

Définition 5.3. *(Version alternative) Un code de Reed-Muller binaire $RM(r,m)$ correspond à l'espace vectoriel des combinaisons linéaires des monômes $1, x_0, x_1, ...x_{m-1}, x_0x_1, x_0x_2, ...$ de degré inférieur à r où les x_j sont tels que, disposés en ligne d'une matrice, la colonne i contienne l'écriture binaire de l'entier i.*

La définition ci-dessus est strictement identique, mais permet une vision plus simple de la matrice génératrice du code. Par exemple, pour $m = 3$ les x_j définis ci-dessus nous donnent les vecteurs suivants :

$$\begin{pmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \end{pmatrix} : \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Ce qui donne par exemple pour le code $RM(2,3)$ la matrice génératrice suivante :

$$RM(2,3) = \begin{pmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \\ x_0x_1 \\ x_0x_2 \\ x_1x_2 \end{pmatrix} : \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Proposition 5.4. *(Plotkin) Soit $0 < r < m$, tout mot d'un code de Reed-Muller $RM(r,m)$ peut-être décomposé de manière unique par un mot de $RM(r, m-1)$ et un mot de $RM(r-1, m-1)$ suivant la construction suivante :*

$$RM(r,m) = \{(u|u \oplus v), u \in RM(r, m-1), v \in RM(r-1, m-1)\}$$

Cette construction du code s'appelle construction de Plotkin.

Démonstration. Par définition, un mot du code $RM(r,m)$ correspond à l'évaluation d'un polynôme binaire à m variables de degré inférieur à r. Soit $(x_0, x_1, ...x_{m-1})$ ces m variables et $f(x_0, x_1, ...x_{m-1})$ le polynôme évalué.

Il est possible de factoriser le polynôme f de la manière suivante :

$$f(x_0, x_1, ...x_{m-1}) = u(x_0, x_1, ...x_{m-2}) + x_{m-1} \times v(x_0, x_1, ...x_{m-2})$$

Ceci peut être fait tout simplement en séparant les monômes contenant x_{m-1} et ceux ne le contenant pas. Il est aussi clair que cette décomposition est unique. Le polynôme u est donc un polynôme binaire à $m-1$ variables de degré strictement inférieur à r et le polynôme v est polynôme binaire à $m-1$ variables de degré strictement inférieur

à $r-1$. D'après la définition d'un code de Reed-Muller, en évaluant le polynôme f sur $\{(0,0,...,0,0),(1,0,...,0,0),...,(1,1,...,1,0)\}$, ceci est équivalent à l'évaluation de u car $x_{m-1}=0$, puis en évaluant sur $\{(0,0,...,0,1),(1,0,...,0,1),...,(1,1,...,1,1)\}$, nous avons bien ici l'évaluation de $u+v$. En concaténant l'évaluation totale, nous obtenons bien le résultat souhaité. □

Il est alors important de noter que le code $RM(0,m)$ correspond au code à répétition car $k=1$ et que le code $RM(m,m)$ peut être ramené au code à identité, c'est-à-dire au code dont la matrice génératrice est la matrice identité. Cela revient dans ce cas à évaluer le polynôme source sur sa base propre et non plus sur la base classique telle que celle de la définition 5.3, ce qui ne modifie en rien les propriétés du code. A partir de ce point, pour des raisons de commodité, nous utiliserons la construction de Plotkin pour les codes de Reed-Muller, en gardant à l'esprit que les résultats sur la base classique sont équivalents.

Théorème 5.5. *Soit un code de Reed-Muller $RM(r,m)$, $0 \leq r \leq m$, et $n=2^m$ la taille du code associé. Chaque élément de ce code peut être construit et décomposé en éléments de code à répétition et de code identité. La complexité d'une telle opération est $\mathcal{O}(n \log n)$.*

Démonstration. La première partie s'obtient directement en appliquant la proposition 5.4 de manière récursive.

Soit $T(n)$ le coût de la construction/décomposition d'un élément de $RM(r,m)$. Grâce à la construction de Plotkin, ce coût peut être ramené au coût de la construction/décomposition de taille $\frac{n}{2}$ et un coût linéaire an, où $a \in \mathbb{R}^+$.

De manière récursive nous avons donc :

$$T(n) = an + 2T(\frac{n}{2}) = an + an + 4T(\frac{n}{4}) = ... = \underbrace{an + an + ... + an}_{\log_2 n \text{ itérations}} = an \log_2 n$$
$$= \mathcal{O}(n \log_2 n).$$

D'où la complexité annoncée. □

La décomposition récursive est illustrée sur la Figure 5.1.

A ce stade, nous venons donc de prouver qu'il est possible d'utiliser les codes de Reed-Muller en tant que code à effacements, avec un encodage et un décodage de complexité logarithmique.

Cependant, en pratique, les résultats de simulation ont montré que la capacité de correction de ce code en utilisant ce simple décodage est mauvaise. La construction de Plotkin permet donc de poser les bases d'un codage efficace mais nécessite quelques adaptations afin de proposer un décodage pertinent.

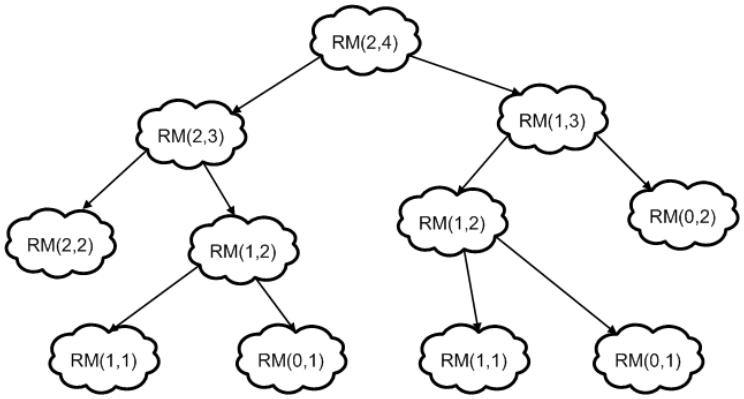

Figure 5.1 – Représentation de la décomposition récursive du code $RM(2,4)$

5.2.2 Utilisation du décodage par permutations

Les codes de Reed-Muller ont vu une grande partie de leurs applications concerner le canal à erreur. Sur ce type de canaux, la problématique du décodage ne concerne pas tant la mise en place du décodage récursif, que le type de décision pris lorsque des erreurs sont détectées. Sur le canal à effacements, la problématique est inverse. Ici nous sommes assurés que les symboles reçus sont corrects, cependant les autres symboles sont totalement inconnus. Ce qui signifie que le problème majeur du décodage récursif sur le canal à effacements est de pouvoir transmettre le maximum d'information possible aux vecteurs de récursions inférieurs.

Illustrons cette problématique par un exemple. Soit un code $RM(1,3)$ qui a donc pour dimension $k = 4$ et pour longueur $n = 8$. Soit le vecteur $a = (a_0, a_1, ..., a_7)$ de ce code. Prenons l'hypothèse que le récepteur n'a reçu que les positions suivantes $a_r = (u|u \oplus v) = (a_0, \times, \times, \times, \times, a_5, a_6, a_7)$. En effectuant une première décomposition récursive, le vecteur $v \in RM(0,2)$ est totalement inconnu, ce qui rend le décodage impossible. Comme nous le voyons sur cet exemple, le point bloquant du décodage récursif est l'impossibilité de décoder des éléments récursifs de bas niveau, qui sont assimilables à des *"stopping sets"*. Par ailleurs, nous voyons que la réussite du décodage n'est pas uniquement dépendante du nombre de symboles reçus, mais également de leurs positions.

Il est important de rappeler à ce moment que tout code de Reed-Muller peut-être décomposé récursivement en codes à répétitions, décodables si au moins une des positions est connue, et en codes identités, nécessitant la connaissance entière du vecteur.

Les codes de Reed-Muller possèdent néanmoins une propriété intéressante :

Propriété 5.6. *Le code de Reed-Muller binaire $RM(r,m)$ avec $n = 2^m$ admet comme groupe de permutations le groupe affine $GA(m)$ [17]. Ceci revient à dire que si $a = (a_0, a_1, ..., a_{n-1}) \in RM(r,m)$ alors $a' = (a_{\Pi(0)}, a_{\Pi(1)}, ..., a_{\Pi(n-1)}) \in RM(r,m)$ où $\Pi(x)$ est une permutation de $GA(m)$.*

Nous rappelons que le groupe affine $GA(m)$ est l'ensemble des applications linéaires de $(\mathbb{F}_2)^m$ sur lui-même de la forme $x \to Ax + b$, où A est une matrice carrée binaire de taille m inversible, et b un vecteur binaire de $(\mathbb{F}_2)^m$.

Cette propriété des codes de Reed-Muller est importante dans la mesure où elle permet de casser les *stopping sets* du décodage récursif sur le canal à effacements. En effet, lorsque un vecteur n'est pas décodable récursivement, une permutation de ce vecteur modifie les positions reçues et permet éventuellement de casser ces *stopping sets*.

Reprenons l'exemple du vecteur a_r reçu précédemment. Nous avons vu qu'en l'état actuel, le décodage récursif échouait. Nous effectuons alors la permutation des positions $\Pi(x) = Ax + b$ avec $A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ et $b = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$, nous obtenons le nouveau mot du code suivant $(a_0, \times, \times, \times, a_6, a_7, \times, a_5)$. Le nouveau vecteur v a donc une position connue et est donc désormais décodable car le $RM(0, 2)$ est le code à répétition de longueur 4. Ce décodage entraine par la suite celui de u et donc du vecteur reçu.

Le point crucial qui va déterminer la capacité de correction du code est alors la capacité à trouver pour chaque niveau de récursion, une permutation permettant à chaque vecteur d'être décodé.

Les effets de telles permutations sur le décodage récursif sur le canal à effacements ont été étudiés dans [75] pour les codes cycliques et les codes cycliques étendus, mais pas dans le cas général des codes de Reed-Muller.

Lorsque la décomposition récursive est effectuée, il est possible de démarrer le décodage indifféremment à partir de u ou de v. Les simulations que nous avons mené ont cependant montré que le décodage était plus efficace dès lors qu'à chaque niveau de récursion, celui-ci débutait par le décodage de v. Nous n'avons pas réussi à démontrer de manière claire ce résultat, mais une explication possible est que, même si v reçoit en moyenne, à chaque étape, deux fois moins d'information que u, v évolue dans un code de Reed-Muller d'ordre r inférieur à u et qui permet de contre-balancer ce manque d'information.

A partir de ce point, nous prenons ainsi l'hypothèse qu'à chaque étape de récursion, le décodage débute toujours par v. L'idée consiste donc à effectuer une permutation du mot reçu qui maximise le nombre de positions connues de v afin de favoriser son décodage de manière récursive. Tout d'abord il est judicieux de remarquer que dans ce cas, le vecteur b des permutations est inutile. En effet, pour le vecteur $a = (u|u \oplus v)$ de taille n, les positions connues de v sont les positions où les a_i et $a_{i \oplus \frac{n}{2}}$ sont connues : $\forall i \in (0, 1, ..., \frac{n}{2} - 1),\ v_i = a_i \oplus a_{i \oplus \frac{n}{2}}$. Le vecteur b permet seulement d'effecteur un décalage identique pour les positions de gauche ou de droite, ou de les inverser lorsque son premier bit est à un, ce qui ne modifie pas le nombre de positions de v connues. Sans perte de généralités, nous pouvons donc nous restreindre aux permutations de la forme $\Pi(x) = Ax$ qui correspond communément au groupe linéaire $GL(m, 2)$.

Nous avons donc défini l'ensemble des permutations admissibles et utiles pour un vecteur de taille n donné. D'après [17], qui donne le nombre de matrices binaires

inversibles de taille m, la complexité associée au test de toutes les permutations possibles pour un vecteur de taille n est $O(0.29 \times n^{\log(n)+1})$. En n'oubliant pas que pour un décodage entier, ce test de permutations doit être effectué à chaque étape de la récursion, nous obtenons une complexité prohibitive.

Nous avons donc choisi de restreindre l'ensemble des permutations testées. Pour cela, pour chaque vecteur de taille n, nous avons sélectionné uniquement les permutations conservant les $\frac{n}{2}$ positions de gauche (le vecteur u) invariantes. De manière matricielle, cela se traduit par le fait que les $m-1$ premières (ou dernières selon la représentation) colonnes de la matrice A sont les colonnes de la matrice identité. Le seul degré de liberté concernant cette matrice correspond aux $m-1$ positions de la dernière colonne. Nous avons donc réduit le nombre de matrices candidates différentes à $2^{m-1} = \frac{n}{2}$. Pour chacune de ces matrices de permutation, l'image des $\frac{n}{2}$ positions de droite du vecteur doivent être calculées, celles de gauche étant invariantes. Avec cette restriction, la complexité associée au test de toutes les permutations candidates pour un vecteur de taille n est donc de $\mathcal{O}(\frac{n^2}{4})$, à comparer avec la complexité du cas général.

La complexité du mécanisme de test pour le code $RM(r,m)$, $n = 2^m$, décodé récursivement est donc $\mathcal{O}(\frac{n^2 \log n}{4})$ avec nos hypothèses.

5.2.3 Remontée partielle de l'information

Indépendamment du décodage par permutations, nous proposons un second mécanisme visant à améliorer la capacité de correction du code. Une des faiblesses constatées avec l'algorithme récursif est que, lorsque le décodage à un niveau de récursion échoue, et ce même avec le décodage par permutations, l'ensemble du décodage échoue. Cependant lorsque un niveau de récursion échoue, celui-ci peut quand même avoir reçu de nouveaux symboles provenant de récursions de niveaux inférieurs parmi celles qui ont réussi. Nous proposons alors un mécanisme tel que, même si la récursion échoue à un niveau, si celle-ci contient de nouveaux symboles, ceux-ci sont transmis à la récursion de niveau supérieur.

Nous pouvons traduire l'intérêt de ce mécanisme sur l'exemple suivant. Soit a un vecteur d'un code RM décomposable en u et v, qui est lui même décomposé en v_1 et v_2. Supposons que le décodage de v_2 testé en premier réussisse et apporte de nouveaux symboles à v, mais que le décodage de v_1 échoue malgré cette information complémentaire. Sans remontée partielle, ce mécanisme échoue. Supposons alors que ces nouveaux symboles de v permettent de décoder de nouveaux symboles de a et donc de u. Le décodage de u tenté par la suite profite donc de ces nouveaux symboles et voit ses chances de réussite augmenter. En retour, si celui-ci réussit, il pourra apporter de nouveaux symboles à a et donc en conséquence potentiellement à v et même v_1 qui aura alors plus de chances d'être décodé, permettant alors un décodage total de a, et ainsi de suite. Nous utilisons donc un mécanisme de "ping-pong" qui brise malheureusement la complexité logarithmique théorique du décodage récursif.

Les résultats de simulation ont cependant montré que dans plus de 90% des cas,

et notamment sur les hauts niveaux, le nombre d'appels récursifs par vecteur était de deux, signifiant que l'utilisation de ce mécanisme est relativement limité, comme nous le verrons sur les résultats de simulation. Bien évidemment, d'un autre côté cela signifie que la complexité pratique du code reste proche de la complexité logarithmique théorique.

5.2.4 Décodage à blanc du code

Contrairement aux précédents mécanismes qui visent à améliorer la capacité de correction du code, le mécanisme de décodage à blanc permet d'augmenter drastiquement la vitesse de décodage du code lors de l'utilisation de symboles.

Pour cela, nous rappelons que lorsque un code à effacements génère des symboles encodés, ce mécanisme peut-être vu comme l'encodage parallèle de z mots de codes. ceci a pour conséquence que lorsque ces symboles sont transmis, les pertes de symboles se traduisent par des positions de pertes identiques pour les z vecteurs de mots de code. L'intérêt est alors de mutualiser toutes les opérations identiques pour l'ensemble des mots d'un même symbole dans une phase antérieure au décodage réel des mots, ce qui est fait pour la plupart des codes à effacements actuels.

Nous pouvons dès lors remarquer que la phase de recherche des permutations évoquée antérieurement est identique pour l'ensemble des mots d'un symbole, les positions reçues étant les mêmes. C'est ceci que nous appelons décodage à blanc. Dans cette phase, nous injectons des symboles de taille nulle dans le décodeur, ceux-ci correspondant aux positions reçus des symboles réels. La recherche des permutations permettant le décodage est donc effectuée sur ce vecteur blanc.

Si le décodage a blanc réussit, les permutations utilisées pour ce décodage sont alors mises en mémoire. Le vrai décodage peut alors s'effectuer en utilisant le chemin et les permutations tracées par le décodage à blanc. Si le décodage à blanc échoue, celui-ci peut cependant avoir apporté de nouvelles positions décodées. Il est alors possible de basculer sur une élimination Gaussienne qui sera effectuée après, ou à la place de ce décodage partiel.

L'intérêt du décodage à blanc est donc de mutualiser le calcul des permutations, qui est coûteux : $\mathcal{O}(\frac{n^2 \log n}{4})$, afin de ne l'effecteur qu'une seule fois par paquet et ainsi réduire son impact sur la complexité du décodage. L'impact pratique du décodage à blanc sur une implémentation paquets/symboles est discuté dans la partie suivante.

5.3 Résultats de simulation

Nous avons comparé le code récursif proposé dans ce chapitre avec l'implémentation de l'élimination Gaussienne sur le même code. Les tests ont été menés sur un Intel Core 2 Extreme @3,06GHz/4Go RAM sur Mac OS X 10.6 en 64 bits.

La Figure 5.2 présente le taux d'échec de chaque code (GE,Optimized) pour le code $RM(3,7)$, qui correspond à un code $(k,n) = (64,128)$, en fonction du nombre de symboles supplémentaires reçus. L'élimination Gaussienne permet ici d'atteindre

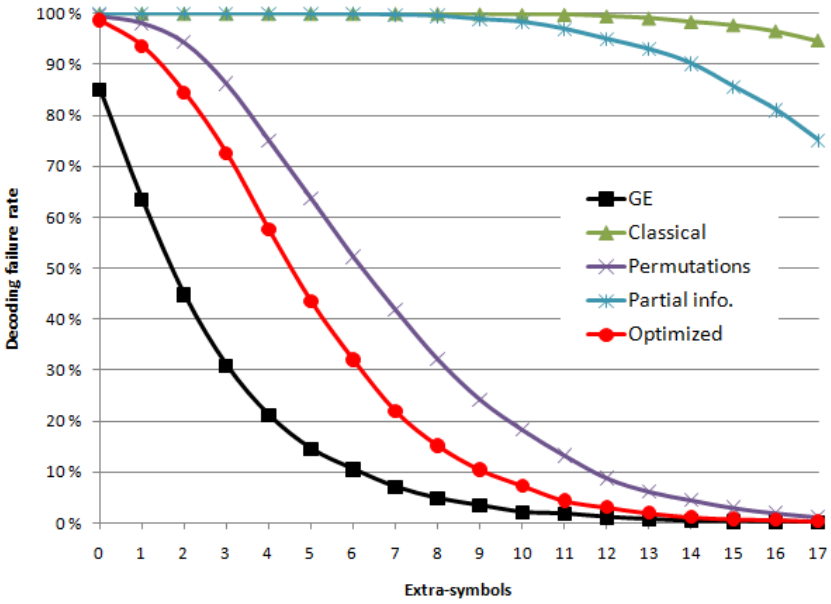

Figure 5.2 – Taux d'échec des différents algorithmes pour le code $RM(3,7)$ en fonction du nombre d'extra-symboles

la capacité théorique du code. Nous avons également ajouté sur ce schéma le taux d'échec du décodage récursif sans optimisation (la version classique) ainsi que les taux d'échec avec uniquement le décodage par permutation et la remontée partielle.

Tout d'abord, ces résultats permettent de mettre en évidence l'impossibilité d'utiliser de manière pratique le décodage récursif classique. Par ailleurs, l'importance du décodage par permutations est mis en lumière. En marge de ces résultats, nous avons également modifié le mécanisme de permutation afin de ne pas nous restreindre aux permutations conservant les positions de gauche. La capacité de correction s'en est trouvée légèrement améliorée, mais ceci s'est fait au détriment d'une chute importante de la vitesse. Le mécanisme que nous avons proposé constitue un compromis entre performance et vitesse.

Enfin, ces résultats nous permettent de discerner le bon comportement de l'algorithme récursif proposé, qui dans ce cas, ne nécessite en moyenne, que trois symboles supplémentaires pour atteindre les taux de réussite du décodage par maximum de vraisemblance.

Les vitesses des deux algorithmes pour ce code sont représentées sur le tableau 5.1. Pour ces paramètres, l'encodeur récursif obtient une vitesse de l'ordre de 4Gbps. Enfin, comme nous le verrons de manière plus précise dans le chapitre suivant, un code MDS de Reed-Solomon de mêmes paramètres fonctionne sur la même machine à environ 400Mbps.

Le tableau 5.1 donne une comparaison de capacité et de vitesse du décodage récursif optimisé et du décodage Gaussien pour différents paramètres du code. Nous

n'avons pas représenté de tailles inférieures, les différences de capacité de correction entre les deux algorithmes étant insignifiantes.

	Algorithme récursif vitesse	Elim. Gaussienne vitesse	Algorithme récursif extra-symboles	ML extra-symboles
RM(3,6)	2021 Mbps	842 Mbps	5.41%	5.06%
RM(3,7)	1073 Mbps	544 Mbps	8.59%	4.75%
RM(4,7)	2393 Mbps	381 Mbps	3.45%	2.79%
RM(4,8)	1363 Mbps	215 Mbps	9.08%	1.44%
RM(5,8)	2774 Mbps	181 Mbps	2.44%	1.17%
RM(5,9)	1783 Mbps	85 Mbps	9.23%	0.45%
RM(6,9)	3291 Mbps	80 Mbps	1.90%	0.47%
RM(6,10)	3486 Mbps	44 Mbps	8.05%	0.18%

Table 5.1 – Vitesse de décodage et extra-symboles moyens pour différents codes RM

Nous avons également étudié l'influence de la taille des symboles sur le gain de vitesse. Sur le tableau 5.2, nous avons représenté la vitesse de chaque algorithme sur un code $RM(6,9)$ de paramètres $(n,k) = (466,512)$ pour des tailles de symboles de 50, 500 et 1500 octets correspondant respectivement aux tailles de paquets VoIP, médianes de l'Internet et de PDU Ethernet.

Taille de paquet	Algorithme récursif	Élimination Gaussienne	Rapport
50 octets	546 Mbps	8 Mbps	66x
500 octets	2463 Mbps	51 Mbps	49x
1500 octets	3291 Mbps	80 Mbps	41x

Table 5.2 – Vitesses de décodage pour le code $RM(6,9)$ en fonction de la taille des paquets

Pour ce code, le rapport de vitesse varie en 40 et 60 en faveur du décodage récursif en fonction de la taille des paquets. Ceci peut s'expliquer par le fait que le décodage à maximum de vraisemblance nécessite une inversion de matrice carrée de taille k. La phase de pré-décodage de l'algorithme Gaussien est donc associée à la complexité $\mathcal{O}(k^3)$. En comparaison, la complexité de prédécodage de l'algorithme récursif est $\mathcal{O}(n^2 \log n)$. Ceci permet d'expliquer que l'écart de vitesse augmente lorsque la taille des paquets diminue. Il est cependant important de nuancer ceci en notant que, pour ce code, la phase de décodage à blanc peut représenter jusqu'à 86% du temps total lorsque la taille des paquets est de 50 octets. Par ailleurs, pour le décodage récursif, le décodage permet de retrouver l'ensemble des symboles d'encodage. Si l'on souhaite retrouver les symboles source par la suite, il suffit d'effectuer un décodage RM, que nous avons négligé ici, celui-ci étant de l'ordre de 8Gbps pour ce code.

De manière générale, l'algorithme que nous avons présenté, pour les hauts ratios, sacrifie 1% à 2% de capacité du code en permettant d'obtenir un gain de vitesse allant de 5 à 20 fois celle du décodage ML.

5.4 Conclusion

Nous avons donc présenté un algorithme de décodage pour les codes de Reed-Muller basé sur la construction de Plotkin. Les améliorations présentées ont permis d'obtenir une très bonne capacité de correction, notamment pour les petites longueurs, en sacrifiant une légère partie de la simplicité du code RM.

Pour les petits codes, et encore plus pour les hauts ratios de codage, la capacité de correction du code présenté est proche de la capacité optimale du code.

Ce code se présente par ailleurs comme une alternative intéressante aux codes de Reed-Solomon pour les petites tailles, lorsque la complexité de décodage est un point crucial, comme par exemple sur des appareils peu puissants, ou alors lorsque les contraintes temporelles sont fortes. En effet, le ratio de vitesse par rapport à un code RS peut atteindre 5x à 20x lorsque les tailles en jeu sont de l'ordre de quelques centaines.

Une implémentation de ce code est disponible sur [76].

Chapitre 6

Codes à effacements MDS basés sur les FNT

Le train de votre mépris roule sur les rails de mon indifférence.

Devise du DMIA.

Sommaire

6.1 Introduction

Dans ce chapitre, nous présentons un code à effacements à Maximum Distance Séparable (MDS). Dans de nombreuses applications, il peut-être important voire nécessaire de travailler sur un code comportant une capacité de correction optimale, ou en d'autres termes, tel que n'importe sous-ensemble codé de taille égale à la taille d'origine permette de retrouver l'ensemble original. Ce type de codes prend tout son sens dans le cadre de stockage distribué localement, tels les systèmes RAID [13] [14], mais aussi la distribution de contenu distant, où la question de la minimisation de la bande passante est cruciale. Les codes MDS permettent aussi de répondre à la problématique du partage de clés [4] et du partage optimal de l'information [6].

Si les codes MDS se présentent comme la solution idéale pour cet ensemble de solutions, ils sont néanmoins pénalisés par une complexité théorique quadratique $\mathcal{O}(n^2)$ qui en pratique se traduit par l'impossibilité d'utiliser ce genre de solutions pour des

tailles supérieures à quelques centaines d'éléments ($\ll$ 1Mbps). Dans ce cas, l'utilisateur préférera utiliser des codes de type LDPC ou Fountain de complexité plus faible, au prix de la perte de cette optimalité.

Des travaux récents [12] ont permis de définir des codes MDS à complexité sous-quadratique, en l'occurrence de l'ordre de $\mathcal{O}(n \log^2 n)$, basé sur des transformées de Walsh. Les codes MDS que nous présentons dans ce chapitre peuvent être encodés et décodés avec un algorithme de complexité pratique logarithmique $\mathcal{O}(n \log n)$. Ils peuvent également être utilisés en tant que code systématique, sans perte notable de performance au niveau du décodeur. Ces codes sont basés sur une application de la transformée de Fourier rapide (FFT) dans un corps de Galois dont la taille est un nombre de Fermat premier. Dans ce type de corps, la FFT est souvent appelée Transformée en Nombres de Fermat (Fermat Number Transform-FNT). Ces codes seront dénommés par la suite RS-FNT.

Ces codes sont les premiers codes publiés permettant d'atteindre la limite théorique de la complexité d'un code MDS [77]. Par ailleurs, ces codes étant équivalents aux codes de Reed-Solomon classiques, ils sont également encodables et décodables avec des algorithmes quadratiques plus rapides pour des codes de petite taille.

6.2 Construction des codes RS-FNT

6.2.1 La transformée de Fourier sur $\mathbb{F}_q$

Les définitions suivantes permettent de définir le contexte dans lequel est défini ce code.

Définition 6.1. *Soit q un nombre premier. L'ensemble $\{0, 1, 2, ..., q-1\}$ muni de l'addition et de la multiplication modulo q forme un corps fini [11]. Ce type de corps, qui fait partie des corps de Galois, est noté $\mathbb{F}_q$ ou parfois GF(q). Par la suite, on appellera un tel corps un corps fini premier.*

Définition 6.2. *Dans le corps fini $\mathbb{F}_q$, l'ordre d'un élément est la plus petite puissance de l'élément égale à 1 modulo q. Lorsque cet ordre est égal à $q-1$, l'élément du corps est appelé une* **racine primitive** *du corps. En ce sens, chaque élément du corps peut être défini comme une puissance d'une racine primitive du corps.*

Il est possible d'étendre la définition de la transformée de Fourier discrète (DFT) aux corps de Galois [78]. Dans ce corps, la définition de la DFT devient :

Définition 6.3. *Soit ω un élément d'ordre $n-1$ de $\mathbb{F}_q$ et $n \leq q$.Soit $a = (a_0, a_1, ..., a_{n-1})$ un vecteur de $\mathbb{F}_q^n$. La transformée de Fourier discrète (DFT) dans ce corps est :*

$$A_j = \sum_{i=0}^{n-1} a_i \omega^{ij} \ , \ 0 \leq j \leq n-1 \tag{6.1}$$

où $A = (A_0, A_1, ..., A_{n-1})$ est un vecteur de $\mathbb{F}_q^n$.

De la même manière, la transformée inverse (DFT^{-1}) est définie comme suit :

$$a_j = \frac{1}{n} \times \sum_{i=0}^{n-1} A_i \omega^{-ij} \ , \ 0 \leq j \leq n-1$$

La transformée de Fourier rapide (FFT) est un algorithme développé par Cooley et Tukey [79] en 1965 permettant le calcul de la DFT dans le cas général en complexité logarithmique $\mathcal{O}(n \log n)$. Comme mis en évidence par Pollard [78], l'algorithme de la FFT peut être adapté de manière similaire dans les corps de Galois, la racine n^{ime} de l'unité des complexes étant ici remplacée par un élément d'ordre n du corps. Ceci est possible car les mécanismes fondamentaux de la FFT sont indépendants du corps utilisé.

Le principe général de la FFT est une approche de type *diviser pour régner*. Cette approche permet d'atteindre la complexité théorique annoncée en divisant basiquement la résolution de la FFT de taille n en la résolution de deux FFT de taille $\frac{n}{2}$.

La transformée de Fourier rapide (FFT) Dans ce paragraphe, nous détaillons le principe de la FFT que nous illustrerons par un exemple dans le corps $\mathbb{F}_q$.

Soit $\mathbb{K}$ un corps fini de taille q, n la taille de la transformée de Fourier, $n < q$, soit $a = (a_0, a_1, ..., a_{n-1})$ et $A = (A_0, A_1, ..., A_{n-1})$ deux vecteurs de $\mathbb{K}^n$. Soit ω un élément d'ordre n du corps. Si A est la DFT de a, la DFT peut s'écrire de façon matricielle (la matrice est symétrique) :

$$A = \underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{pmatrix}}_{DFT_\omega(n)} \times a \qquad (6.2)$$

On suppose dès lors que n est une puissance de deux. Si l'existence d'une racine primitive a été prouvée sur les corps de Galois, la FFT utilise une propriété intéressante des éléments d'ordre :

Proposition 6.4. *Soit ω un élément d'ordre n pair, dans un corps de Galois $\mathbb{F}_q$, $n < q$. Alors ω^2 est une élément d'ordre $\frac{n}{2}$ du corps.*

Démonstration. Par définition, $\omega \equiv 1 \mod q$, et n étant pair nous avons donc $\omega^{2 \times \frac{n}{2}} \equiv 1 \mod q$ et donc $(\omega^2)^{\frac{n}{2}} \equiv 1 \mod q$. L'ordre de ω^2 est donc inférieur ou égal à $\frac{n}{2}$. Supposons qu'il existe m tel que $\omega^{2^m} \equiv 1 \mod q$ et $m < \frac{n}{2}$. Le résultat suivant

nous donne donc $\omega^{2m} \equiv 1 \mod q$. ω étant d'ordre n, nous avons donc $n \leq 2m$, ce qui contredit l'hypothèse initiale. □

Pour la FFT, les indices pairs et impairs de a sont séparés en deux vecteurs a_p et a_i. L'algorithme de la FFT réduit ainsi le calcul de la DFT de taille n en deux DFT de taille $\frac{n}{2}$, dont l'élément d'ordre est ω^2 et une multiplication par un vecteur de taille n de la manière suivante :

$$A = DFT_\omega(n) \times a = \begin{pmatrix} DFT_{\omega^2}(\frac{n}{2}) \cdot a_p \\ DFT_{\omega^2}(\frac{n}{2}) \cdot a_p \end{pmatrix} + \begin{pmatrix} 1 \\ \omega \\ \vdots \\ \omega^{n-1} \end{pmatrix} \cdot \begin{pmatrix} DFT_{\omega^2}(\frac{n}{2}) \cdot a_i \\ DFT_{\omega^2}(\frac{n}{2}) \cdot a_i \end{pmatrix} \quad (6.3)$$

Le principe de l'algorithme de la FFT est d'appliquer de mécanisme de manière récursive. Ce qui permet d'introduire la proposition suivante :

Proposition 6.5. *L'algorithme de la transformée de Fourier rapide (FFT) est de complexité $\mathcal{O}(n \log n)$.*

Démonstration. Soit $T(n)$ la complexité de l'algorithme de taille n. D'après l'équation 6.3, cette complexité peut être ramenée en la complexité de l'opération $T(\frac{n}{2})$ et la complexité linéaire du produit et de la somme $f(n)$. Nous avons donc :

$$T(n) = 2T(\frac{n}{2}) + f(n)$$

En développant nous obtenons :

$$T(n) = f(n) + 2(f(\frac{n}{2}) + 2T(\frac{n}{4})) = ... = f(n) + 2f(\frac{n}{2}) + 4f(\frac{n}{4}) + 8f(\frac{n}{8}) + ...$$

$f(n)$ étant de complexité linéaire alors $\exists a \in \mathbb{R} \backslash f(n) = an$. En conséquence nous avons :

$$T(n) = \underbrace{an + an + ... + an}_{\log_2 n \text{ itérations}} = an \log_2 n = \mathcal{O}(n \log_2 n).$$

□

Illustrons cet algorithme dans le corps fini premier $\mathbb{F}_{17}$. Dans ce corps, l'élément 4 est d'ordre 4. Soit $a = (3, 7, 5, 14)$ le vecteur défini sur ce corps. L'algorithme de la FFT nous donne le résultat suivant :

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 16 & 13 \\ 1 & 16 & 1 & 16 \\ 1 & 13 & 16 & 4 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 7 \\ 5 \\ 14 \end{pmatrix} = \begin{pmatrix} DFT_{\omega^2} \cdot \begin{pmatrix} 3 \\ 5 \end{pmatrix} \\ DFT_{\omega^2} \cdot \begin{pmatrix} 3 \\ 5 \end{pmatrix} \end{pmatrix} + \begin{pmatrix} 1 \\ 4 \\ 16 \\ 13 \end{pmatrix} \begin{pmatrix} DFT_{\omega^2} \cdot \begin{pmatrix} 7 \\ 14 \end{pmatrix} \\ DFT_{\omega^2} \cdot \begin{pmatrix} 7 \\ 14 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 12 \\ 4 \\ 4 \\ 9 \end{pmatrix}$$

où $DFT_{\omega^2} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 16 \end{smallmatrix}\right)$.

Les résultats précédents peuvent s'appliquer *mutatis mutandis* dans le cadre de la transformée de Fourier inverse.

Proposition 6.6. *Pour tout $k \leq n$, les k premières lignes de la matrice de la transformée de Fourier discrète (DFT) forment une matrice génératrice d'un code de Reed-Solomon. La DFT, qui peut être obtenue via la FFT, est donc un code MDS.*

Démonstration. L'équation 6.2 a mis en lumière l'écriture matricielle de la transformée de Fourier. Ainsi, il est possible de remarquer que la matrice de la DFT est une matrice particulière de Vandermonde avec le vecteur $(1, \omega, \omega^2, ...\omega^{n-1})$. ω étant d'ordre n, la matrice est de rang plein ce qui termine la démonstration. □

Une autre façon de considérer la transformée de Fourier discrète vient de la représentation polynômiale.

Proposition 6.7. *Soit $\mathbb{K}$ un corps fini de taille q. Soient ω un élément d'ordre n, $n < q$, $a = (a_0, a_1, ..., a_{n-1})$ et $A = (A_0, A_1, ..., A_{n-1})$ deux vecteurs de $\mathbb{K}^n$. La transformée de Fourier discrète A de a est équivalente à l'évaluation du polynôme $a(x) = \sum_{i=0}^{n-1} a_i x^i$ aux points $(1, \omega, \omega^2, ..., \omega^{n-1})$ en progression géométrique.*

Démonstration. En reprenant l'écriture originale de la DFT, nous avons pour $j \in (0, 1, ..., n-1)$:

$$a(\omega^j) = \sum_{i=0}^{n-1} a_i \omega^{j^i} = \sum_{i=0}^{n-1} a_i \omega^{ij} = A_j.$$

□

6.2.2 La FNT sur $\mathbb{F}_{65537}$

Si l'ensemble des propriétés sur la FFT sont valables sur $\mathbb{F}_q$ dès lors que q est premier, en pratique, il est intéressant de travailler sur des corps permettant une adaptation pratique aisée. Les terminaux fonctionnant généralement avec des unités de 8, 16, 32 voire 64 bits, c'est-à-dire de 2^r bits, l'idéal est donc d'utiliser des corps dont les éléments peuvent être représentés par des vecteurs de bits ayant ces longueurs. D'autre part, pour réaliser une implémentation efficace de la FFT, celle-ci doit être de taille 2^m, ce qui implique l'existence d'un élément d'ordre 2^m dans le corps fini. Malheureusement, tous les éléments des corps de taille 2^{2^r} sont d'ordre impair.

Une solution par rapport à ce problème consiste à utiliser des corps fini dont le cardinal est un nombre de Fermat de type $F_p = 2^{2^p} + 1$. Ces corps contiennent des éléments ayant comme ordre toutes les puissances de 2 comprises entre 0 et 2^{2^p}. Il a été prouvé que les nombres de Fermat $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$ et $F_4 = 65537$ sont premiers, F_4 étant à ce jour le plus grand nombre de Fermat

premier connu. Dans la pratique, nous avons donc développé notre code sur le corps de Galois $\mathbb{F}_{65537}$ qui permet de coder des valeurs sur des mots de 16 bits hormis la valeur "65536".

Il est possible de résoudre ce problème, car en pratique, le code est limité à une taille $n = 32768$. Ce qui signifie que pour chaque paquet de symboles, il existe une plus petite valeur du corps qui n'est pas présente. Dans ce cas, la valeur 65536 sera codée par cette valeur absente, et sera signalé tel quel par l'encodeur via un champ d'en-tête de 16 bits.

L'application de la DFT/FFT sur ce type de corps se définit ainsi :

Définition 6.8. *La Transformée en Nombres de Fermat (FNT) est l'application de la transformée de Fourier rapide (FFT) sur un corps $\mathbb{F}_q$ où q est un nombre de Fermat premier.*

Dans le corps $\mathbb{F}_{65537}$, nous avons utilisé 3 comme racine primitive. Pour des codes de taille $n = 2^i$, l'élément d'ordre utilisé pour la FNT sera $3^{2^{16-i}}$.

6.2.3 L'algorithme de décodage de la FNT

Dans la suite de cette partie, nous prenons l'hypothèse que la taille n du code est une puissance de deux. Lors d'une implémentation pratique, si n n'est pas une puissance de deux, alors la FNT travaillera sur la plus petite puissance de deux supérieure à n et le code sera poinçonné. De la même manière, nous prenons aussi k comme une puissance de deux, en remplissant éventuellement le vecteur source de la FNT de zéros jusqu'à atteindre une puissance de deux. Au niveau du décodage, le décodeur considérera ainsi ce *padding* comme faisant partie de l'ensemble des éléments reçus.

Soit un vecteur de symboles sources $s = (s_0, s_1, ..., s_{k-1})$ de dimension k et à valeurs dans $\mathbb{F}_q$, et soit $s(x) = \sum_{i=0}^{k-1} s_i x^i$ le polynôme associé. Nous complétons ce vecteur avec des 0 afin de former un vecteur de taille n. Le polynôme associé n'est ainsi pas modifié. Il est alors possible de calculer la FNT de ce vecteur, qui donnera un vecteur de symboles codés $c = (c_0, c_1, ..., c_{n-1})$. La proposition 6.6 nous indique ainsi que ce mécanisme forme un code MDS, à savoir qu'il est possible de retrouver l'ensemble des symboles sources à partir de n'importe quel sous-ensemble de c comportant k symboles. Le point principal est donc de définir un décodage efficace associé à cette opération.

Dans le cas particulier où l'ensemble du vecteur c a été reçu, le décodage est immédiat. Il correspond simplement à l'application de la transformée de Fourier inverse (FNT^{-1}) sur le vecteur c. Dans le cas général cependant, au moins un des symboles codés a pu être perdu, rendant impossible l'application du mécanisme précédent. Il est donc nécessaire de définir un algorithme de décodage de la FNT lorsque au moins k symboles codés ont été reçus, mais pas tous.

La proposition 6.7 a permis de mettre en lumière la relation entre la FNT/DFT et l'évaluation polynômiale. Soit ω l'élément d'ordre n utilisé pour la FNT, alors les symboles codés c_i correspondent à l'évaluation du polynôme $s(x)$ sur les ω^i :

$$\forall i \in \{0, 1, ..., n-1\}, \qquad c_i = s(\omega^i)$$

Le décodage peut donc être vu comme un problème général d'interpolation polynômiale. Il s'agit ici, de déterminer le polynôme de degré $k-1$, s(x) à partir de k points d'évaluations reçus faisant partie d'une suite en progression géométrique. Cette vue nous permet d'ailleurs de démontrer d'une autre manière le caractère MDS de la FNT.

L'interpolation polynômiale est un vaste domaine étudié depuis de très nombreuses années et qui possède un large éventail de solutions. Certaines solutions de complexité $\mathcal{O}(n \log n)$ ont été proposées. Toutefois, elles sont relativement difficiles à implémenter en l'état et la plupart des solutions et implémentations classiques possèdent une complexité quadratique. L'algorithme de décodage que nous présentons permet quant à lui d'effectuer cette interpolation et de décoder la FNT avec une complexité pratique de $\mathcal{O}(n \log n)$.

Prérequis mathématiques

Tout d'abord, nous présentons quelques résultats utiles à l'algorithme de décodage.

Dans le cas général, la multiplication de deux polynômes de degré strictement inférieur à n est de complexité $\mathcal{O}(n^2)$ en utilisant une méthode naïve où chaque produit de monômes est calculé. Dans les années 1960, Karatsuba a présenté un algorithme [80] permettant de réduire la complexité théorique de cette multiplication à $O(n^{log_2 3}) \simeq O(n^{1.59})$. Soient les polynômes $a(x) = a_1 x + a_0$ et $b(x) = b_1 x + b_0$, l'algorithme de Karatsuba utilise la décomposition de $a(x)b(x)$ suivante :

$$a(x)b(x) = a_1 b_1 x^2 + (a_1 b_1 + a_0 b_0 - (a_1 - a_0)(b_1 - b_0))x + a_0 b_0$$

Cette décomposition ne requiert ainsi que 3 multiplications et permet d'atteindre la complexité proposée. Toom et Cook [81][82] ont généralisé cette méthode pour des schémas de polynômes de degré k qui ont donné les algorithmes de Toom-k, et qui sont de complexité $\mathcal{O}(n^{\frac{\log(2k-1)}{\log k}})$. Cependant ces algorithmes trainent une constante prohibitive dès lors que k dépasse 5. L'algorithme de multiplication polynômiale que nous allons utiliser ici est une adaptation de l'algorithme de Schönhage-Strassen [83] aux corps finis.

Proposition 6.9. *(Schönhage-Strassen) Soient $a(x) = \sum_{i=0}^{n-1} a_i x^i$ et $b(x) = \sum_{i=0}^{n-1} b_i x^i$ deux polynômes à valeurs dans $(\mathbb{F}_q)^n$ avec $q > 2n$. Il est possible d'en calculer le produit $a(x)b(x)$ avec la complexité $\mathcal{M}(n) = \mathcal{O}(n \log n)$.*

Démonstration. Soit ω un élément d'ordre $2n$ dans le corps considéré. La DFT-2n des polynômes $a(x)$ et $b(x)$ peut être vue grâce à la proposition 6.7 comme l'évaluation de ces polynômes sur les différents ω^i. Soient A et B ces DFT, nous avons :

$$\forall i \in \{0, 1, ..., 2n-1\}, \quad A_i = a(\omega^i), B_i = b(\omega^i)$$

Il est possible d'effectuer le produit terme à terme de ces deux vecteurs ce qui donne :

$$A_i B_i = a(\omega^i) b_(\omega^i) = ab(\omega^i).$$

Le vecteur produit contient donc l'évaluation du polynôme produit $a(x)b(x)$ de degré strictement inférieur à $2n$, sur $2n$ points. Tous les points d'interpolation étant connus, il est donc possible d'appliquer directement la transformée de Fourier inverse afin de retrouver le polynôme produit. En utilisant la FFT, la complexité de cette opération est donc $\mathcal{O}(n \log n)$. En d'autres termes nous avons :

$$a(x)b(x) = FFT^{-1}(FFT(a) \cdot FFT(b))$$

□

De ce résultat, nous obtenons la proposition suivante :

Proposition 6.10. *Soit $(j_0, j_1, ..., j_{n-1})$ un vecteur de $\mathbb{F}_q^n$ où n est une puissance de 2. Le calcul du polynôme $j(x) = \prod_{i=0}^{n-1} (x - j_i)$ peut être effectué avec la complexité $\mathcal{O}(\mathcal{M}(n) \log n) = \mathcal{O}(n \log^2 n)$.*

Démonstration. n étant une puissance de deux, il est possible ici d'appliquer une méthode de type *diviser pour régner*. En l'occurrence, on divise le polynôme $j(x)$ en deux polynômes $j_1(x) = \prod_{i=0}^{\frac{n}{2}-1} (x - j_i)$ et $j_2(x) = \prod_{i=\frac{n}{2}}^{n-1} (x - j_i)$. La complexité $T(n)$ de calcul du polynôme $j(x)$ peut donc s'écrire :

$$T(n) = 2T(\frac{n}{2}) + \mathcal{M}(\frac{n}{2})$$

En appliquant récursivement ce mécanisme, nous obtenons :

$$T(n) = \mathcal{M}(\frac{n}{2}) + 2(\mathcal{M}(\frac{n}{4}) + 2T(\frac{n}{4}) + ...).$$

En majorant les $\mathcal{M}(\frac{n}{2^i})$ par $a\frac{n}{2^i} \log n$, grâce à la proposition 6.9, nous obtenons :

$$T(n) \leq \underbrace{an \log n + an \log n + ... + an \log n}_{\log_2 n \text{ itérations}} = an \log_2^2 n = \mathcal{O}(n \log_2^2 n).$$

□

Un cas particulier permet cependant de réduire la complexité de ce calcul.

Proposition 6.11. *Lorsque les $(j_0, j_1, ..., j_{n-1})$ sont en progression géométrique, $j_i = a^i$, la complexité de calcul de $j(x)$ est réduite à $\mathcal{O}(\mathcal{M}(n)) = \mathcal{O}(n \log n)$.*

Démonstration. Lorsque les points sont en progression géométrique, et lors de l'application de la méthode *diviser pour régner*, il suffit de remarquer que le second polynôme du produit peut se déduire du premier grâce à une homothétie de complexité $\mathcal{O}(n)$. La complexité $T(n)$ du calcul du produit devient donc :

$$T(n) = T(\frac{n}{2}) + \mathcal{M}(\frac{n}{2}) = \frac{n}{2}\log(\frac{n}{2}) \times (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ...) = n\log(\frac{n}{2})$$

□

Un dernier résultat est nécessaire avant de présenter l'algorithme de décodage. Introduisons tout d'abord le lemme suivant :

Lemme 6.12. *Soit ω un élément d'ordre n du corps de Galois $\mathbb{F}_q$. Alors ω^{-1} est aussi un élément d'ordre n de ce même corps.*

Démonstration. Directe. □

Proposition 6.13. *Soient $(b_0, b_1, ..., b_{k-1})$ et $(x_0, x_1, ..., x_{k-1})$ deux vecteurs de $\mathbb{F}_q^k$, où k est une puissance de deux, et tel que les x_i forment une sous-suite d'une suite géométrique de paramètre ω, élément d'ordre $n < q$ du corps, et deux à deux distincts. Chaque x_i peut s'écrire sous la forme $x_i = \omega^{z_i}$. Supposons par ailleurs, $\forall i, z_i < n$. Enfin, supposons que le polynôme de degré k, $D(x) = \prod_{i=0}^{k-1}(x - x_i)$ connu. Alors le numérateur $N(x)$ de la somme fractionnelle $S(x) = \sum_{i=0}^{k-1} \frac{b_i}{x - x_i} = \frac{N(x)}{D(x)}$ peut être évalué avec la complexité $\mathcal{O}(n\log n + 2k\log 2k)$.*

Démonstration. Par définition, $N(x)$ est un polynôme de degré $k-1$. Nous pouvons donc déterminer sa valeur en travaillant modulo x^k. Nous avons :

$$N(x) = D(x)S(x) \mod x^k$$

En effectuant le développement en série de $S(x)$ à l'ordre k, et en utilisant la formule $\frac{1}{a-x} = \frac{1}{a}\frac{1}{1-\frac{x}{a}} = \sum_i a^{-(i+1)}x^i$, nous obtenons :

$$N(x) = -D(x) \times \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} b_i(x_i)^{-(j+1)}x^j = -D(x) \times \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} b_i(\omega^{-(j+1)})^{z_i}x^j$$

Posons le polynôme $B(x) = \sum_{i=0}^{k-1} b_i x^{z_i}$ de degré strictement inférieur à n. Alors le numérateur $N(x)$ peut s'écrire :

$$N(x) = -D(x) \times \sum_{j=0}^{k-1} B(\omega^{-j-1})x^j$$

D'après le lemme 6.12, ω étant d'ordre n, c'est aussi le cas de son inverse. En conséquence, il est donc possible d'évaluer le polynôme $B(x)$ sur $(1, \omega^{-1}, \omega^{-2}, ...\omega^{-(n-1)})$ grâce à la FFT en $\mathcal{O}(n\log n)$.

Le calcul du numérateur se résume donc au produit d'un polynôme de degré k par un polynôme de degré $k-1$, k étant par définition une puissance de 2, grâce à la proposition 6.9, il est possible d'effectuer ce produit en $\mathcal{O}(2k\log 2k)$ [1]. □

Description de l'algorithme

Nous avons désormais détaillé l'ensemble des outils nécessaires à l'algorithme de décodage de la FNT.

Définition 6.14. *Soient $(x_0, x_1, ...x_{n-1})$ et $(y_0, y_1, ..., y_{n-1})$ deux vecteurs de $\mathbb{F}_q^n$. Le polynôme $L(x)$ défini par :*

$$L(x) = \sum_{i=0}^{n-1} y_i \times \prod_{j\neq i} \frac{x - x_j}{x_i - x_j}$$

est l'unique polynôme de degré inférieur à n tel que $\forall i \in (0, 1, ...n-1),\ L(x_i) = y_i$. Ce polynôme est appelé le **polynôme interpolateur de Lagrange**.

Il est ainsi possible de transposer cette définition au décodage de la FNT.

Proposition 6.15. *Soit un vecteur d'éléments sources $(s_0, s_1, ...s_{k-1})$ de k éléments et $s(x) = \sum_{i=0}^{k-1} s_i x^i$ le polynôme associé. La FNT de taille n produit un ensemble d'éléments codés $(c_0, c_1, ..., c_{n-1})$ qui forment un code MDS. Ces valeurs correspondent à l'évaluation du polynôme $s(x)$ aux positions $(1, \omega, ..., \omega^{n-1})$ où ω est un élément d'ordre n du corps. Soient les vecteurs $c' = (c'_0, c'_1, ...c'_{k-1})$ et $\Omega = (\Omega_0, \Omega_1, ...\Omega_{k-1})$ les k premiers éléments reçus et leurs points d'évaluation respectifs. Il est ainsi possible de retrouver l'ensemble des éléments sources en utilisant le polynôme interpolateur de Lagrange :*

$$s(x) = \sum_{i=0}^{k-1} c'_i \times \prod_{j\neq i} \frac{x - \Omega_j}{\Omega_i - \Omega_j}$$

Il est notamment important de noter que le vecteur Ω forme une sous-suite d'éléments d'une suite en progression géométrique de paramètre ω.

Théorème 6.16. *Le décodage de la transformée de Fourier numérique (FNT) sur $\mathbb{F}_q$ en tant que code MDS de dimension k et de taille n, $k < n < q$, peut être effectué avec la complexité $\mathcal{O}(k\log^2 k + n\log n)$. Cette complexité est réduite à $\mathcal{O}(k\log k + n\log n)$ lorsque les k premiers éléments codés sont reçus.*

Par ailleurs, en pratique, lors de l'implémentation réelle sur des symboles/paquets, cette complexité devient $\mathcal{O}(n\log n)$.

1. Il est par ailleurs intéressant de noter que nous sommes seulement intéressés par les k premiers éléments du produit. Pour cela, il est nécessaire de passer par une évaluation de taille $2k$. La question de ce calcul sur une évaluation de taille k reste à ce jour un problème ouvert. Des travaux ont permis de réduire effectivement la taille de l'évaluation pour des algorithmes naïfs ou de type Karatsuba[84][85], mais la question reste entière pour l'algorithme de Schönhage-Strassen

Démonstration. Soit ω un élément d'ordre n du corps, soit $s = (s_0, s_1, ...s_{k-1})$ l'ensemble des k éléments sources de $\mathbb{F}_q$ et $s(x) = \sum_{i=0}^{k-1} s_i x^i$ son polynôme associé. L'encodage via la FNT produit un ensemble d'éléments codés $(c_0, c_1, ..., c_{n-1})$ sur $\mathbb{F}_q^n$.

Soient les vecteurs $c^{'} = (c_0^{'}, c_1^{'}, ...c_{k-1}^{'})$ et $\Omega = (\Omega_0, \Omega_1, ...\Omega_{k-1})$ les k premiers éléments reçus et leurs points d'évaluation respectifs. La proposition 6.15 nous permet de retrouver l'ensemble des éléments sources s à partir de n'importe quels k éléments $c^{'}$ grâce à l'interpolation lagrangienne :

$$s(x) = \sum_{i=0}^{k-1} c_i^{'} \times \prod_{j \neq i} \frac{x - \Omega_j}{\Omega_i - \Omega_j}$$

Soit le polynôme $a(x) = \prod_{j=0}^{k-1} (x - \Omega_j)$ de degré k. Soient les polynômes $a_i(x) = \prod_{j=0, j\neq i}^{k-1} (x - \Omega_j)$de degré $k-1$ définis pour $i \in (0, 1, ..., k-1)$. Nous avons $a_i(x) = \frac{a(x)}{x - \Omega_i}$. Le polynôme $s(x)$ peut alors être réécrit comme suit :

$$s(x) = a(x) \times \sum_{i=0}^{k-1} c_i^{'} \times \frac{1}{(x - \Omega_i) a_i(\Omega_i)}$$

Calculons la dérivée de $a(x)$ en tant que polynôme. Par développement, cette dérivée s'exprime sous la forme $a^{'}(x) = \sum_{i=0}^{k-1} \prod_{j=0, j\neq i}^{k-1} (x - \Omega_j)$, ou autrement : $a^{'}(x) = \sum_{i=0}^{k-1} a_i(x)$. Il est alors important de noter que si $i \neq j$, alors $a_i(\Omega_j) = 0$. On obtient alors l'égalité suivante :

$$a^{'}(\Omega_i) = a_i(\Omega_i)$$

Ce résultat important nous permet donc de réécrire le polynôme $s(x)$:

$$s(x) = a(x) \times \sum_{i=0}^{k-1} \frac{c_i^{'} / a^{'}(\Omega_i)}{x - \Omega_i}$$

Grâce à la proposition 6.10, le calcul du polynôme $a(x)$ peut être effectué dans le cas général avec la complexité $\mathcal{O}(k \log^2 k)$. Lorsque les k premiers éléments sont reçus, nous avons $\Omega_i = \omega^i$ et alors dans ce cas, grâce à la proposition 6.11, la complexité chute à $\mathcal{O}(k \log k)$.

Le calcul de la dérivée $a^{'}(x)$ est de complexité linéaire $\mathcal{O}(k)$ et donc négligeable.

L'évaluation de la dérivée sur les Ω_i est de complexité $\mathcal{O}(n \log n)$ car elle correspond à l'évaluation sur la sous-suite des ω^i et donc à la FNT de $a^{'}(x)$ de taille n.

En posant $d_i = \frac{c_i^{'}}{a^{'}(\Omega_i)}$, le calcul de $s(x)$ revient donc au calcul de :

$$s(x) = a(x) \times \sum_{i=0}^{k-1} \frac{d_i}{x - \Omega_i}$$

Le dénominateur de la somme fractionnelle étant égal à $a(x)$, le numérateur sera alors $s(x)$. Grâce à la proposition 6.13, il est possible alors de déterminer $s(x)$ avec la complexité $\mathcal{O}(n \log n + 2k \log 2k)$.

Dans le cas général, la complexité de l'algorithme de décodage de la FNT est donc $\mathcal{O}(k \log^2 k + n \log n)$. Dans le cas où les k premiers éléments sont reçus, cette complexité devient $\mathcal{O}(k \log k + n \log n)$.

Il est important de remarquer que le calcul de $a(x)$ et l'évaluation de sa dérivée sur les Ω_i ne dépendent seulement que des positions reçues. Elle est indépendante des valeurs reçues. Dans le cadre de l'adaptation aux symboles/paquets, composé d'un nombre sz d'éléments de $\mathbb{F}_q$, les positions reçues sont identiques pour l'ensemble des sz bi-octets d'un paquet/symbole.

A titre d'exemple, une trame Ethernet de 1500 octets possède donc $sz = 750$ bi-octets. Lorsque ces trames sont reçues, les positions reçues sont donc identiques pour ces 750 éléments de $\mathbb{F}_q$.

Le calcul de $a(x)$ et des évaluations de sa dérivée n'est donc effectué qu'une fois par symbole/paquet. En pratique, la complexité du décodage de la FNT est donc $\mathcal{O}(\frac{k \log^2 k}{sz} + n \log n + 2k \log 2k)$. La partie divisée par sz devient ainsi négligeable, et pour un taux de codage fixé, k étant une proportion de n, en pratique, la complexité du décodage de la FNT sur des paquets/symboles devient donc $\mathcal{O}(n \log n)$. □

6.2.4 Construction d'un code systématique

Dans le paragraphe précédent, nous avons vu que la FNT était encodable et décodable en tant que code MDS(n,k) avec une complexité logarithmique. Cependant la FNT n'est pas un code systématique. Afin de rendre l'encodage et le décodage systématique, nous introduisons un encodage/décodage FNT supplémentaire, permettant de conserver la complexité logarithmique.

Soit $s = (s_0, s_1, ..., s_{k-1})$ le vecteur d'éléments sources. Au niveau de l'encodeur, à partir de ce vecteur de taille k, nous effectuons un décodage FNT de taille n. Ceci nous permet donc d'obtenir un vecteur $i = (i_0, i_1, ..., i_{k-1})$ de k éléments intermédiaires. Il suffit alors d'appliquer la FNT sur les éléments intermédiaires, qui nous permet ainsi d'obtenir une ensemble de n éléments codés, les k premiers correspondant alors aux éléments systématiques, $c = (s_0, s_1, ...s_{k-1}, c_k, c_{k+1}, ..., c_{n-1})$. Ce mécanisme est schématisé sur la Figure 6.1.

Au niveau du décodeur, le mécanisme est similaire. Un décodage FNT effectué sur les symboles codés permet de retrouver l'ensemble des symboles intermédiaires. En appliquant alors la FNT, on retrouve les k symboles sources sur les k premières positions (ainsi que les autres symboles codés par ailleurs). Ce comportement est schématisé sur la Figure 6.2.

L'ensemble encodeur/décodeur ainsi créé forme ce que nous appelons le code RS-FNT.

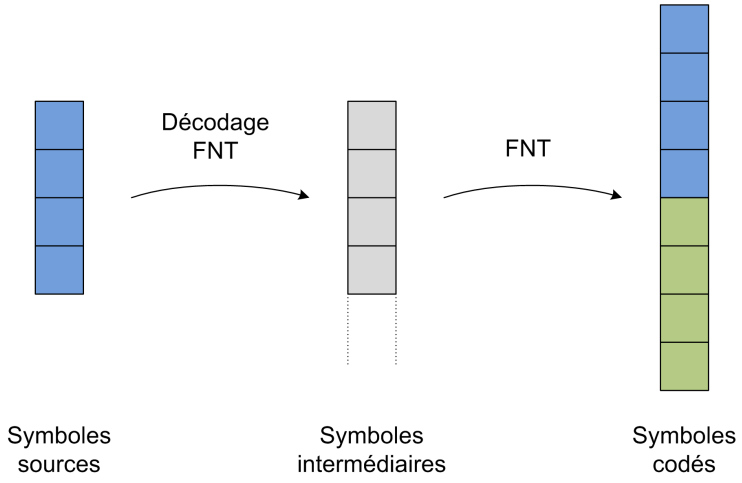

Figure 6.1 – Schéma de l'encodeur systématique RS-FNT

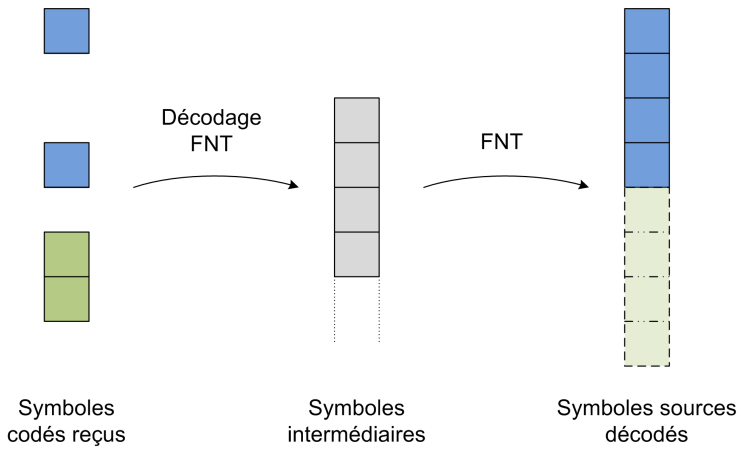

Figure 6.2 – Schéma du décodeur systématique RS-FNT

6.2.5 Encodage/décodage quadratique en tant que code de Reed-Solomon

Lorsque les tailles de codes sont inférieure à quelques centaines d'éléments, il peut s'avérer plus efficace d'utiliser un algorithme classique de complexité quadratique par rapport à l'encodeur/décodeur logarithmique. De par sa structure, le code RS-FNT est un code de Reed-Solomon et peut donc être encodé et décodé comme tel. Dans ce cas, les symboles de redondance sont donc construits directement par la matrice génératrice du code, et les symboles sources sont décodés grâce à la matrice de passage des symboles de redondance aux symboles sources.

Dans le cadre de l'implémentation du code, deux encodeurs quadratiques ont été retenus.

- L'algorithme le plus classique consistant à calculer les $n-k$ symboles de redondance à partir des k symboles sources et de la matrice génératrice du code. Ce code a donc une complexité de $\mathcal{O}(k \times (n-k))$. Ce code ressemble très fortement aux algorithmes d'encodage des codes de Reed-Solomon classiques.
- Un algorithme adapté aux petits taux de codage qui tire parti de la FNT. Dans cet algorithme, les symboles intermédiaires sont obtenus par la matrice de taille $k \times k$ à partir des symboles sources. Par la suite, ces symboles intermédiaires sont directement codés par la FNT, de la même manière que l'algorithme de complexité quadratique. La complexité associée à cet encodeur est donc $\mathcal{O}(k^2 + n \log n)$.

6.3 Etude de la complexité théorique

Si nous avons pu déterminer une estimation de la complexité théorique des codes FNT, il peut s'avérer utile de pouvoir exprimer la complexité du code en évaluant le nombre réel de FNT effectuées. Dans ce paragraphe nous noterons $FNT(x)$ la complexité de la FNT de taille x.

Tout d'abord, il est nécessaire d'établir la complexité du décodage FNT. Comme nous l'avons vu précédemment, le calcul de $a(x)$ et l'évaluation de sa dérivée ont un coût négligeable lors de l'adaptation aux paquets/symboles. En conséquence, seule la complexité du calcul de la somme fractionnaire est utile. Comme nous avons pu le voir grâce à la proposition 6.13, ce calcul consiste en l'évaluation du polynôme $B(x)$ de degré strictement inférieur à n et au produit de deux polynômes de degrés k et $k-1$.

L'évaluation du polynôme $B(x)$ correspond directement au calcul de la FNT de taille n de celui-ci. Sa complexité est donc $FNT(n)$. En utilisant l'algorithme de Schönhage-Strassen de la proposition 6.9, le calcul du produit sur une taille $2k$ correspond à celui de 3 FNT. La complexité de ce produit est donc de l'ordre de $3FNT(2k)$. En règle générale, le coût du décodage de la FNT est donc de l'ordre de $FNT(n) + 3FNT(2k)$.

A titre d'exemple, lorsque le taux de codage est de $\frac{1}{2}$, le coût du décodage de la FNT est donc de $4FNT(n)$. Nous pouvons dès lors résumer cette complexité sur les Tableaux 6.1 et 6.2.

	Non systématique	Systématique
Encodeur	FNT(n)	2FNT(n)+3FNT(2k)
Décodeur	FNT(n)+3FNT(2k)	2FNT(n)+3FNT(2k)

Table 6.1 – Complexité théorique de l'encodeur et du décodeur RS-FNT

	Non systématique	Systématique
Encodeur	FNT(n)	5FNT(n)
Décodeur	4FNT(n)	5FNT(n)

Table 6.2 – Complexité théorique de l'encodeur et du décodeur RS-FNT, taux de codage $\frac{1}{2}$

Ces résultats sur la complexité mettent en lumière le fait que le décodeur n'est que peu affecté par le codage systématique, contrairement à l'encodeur. Ils mettent également en valeur l'importance prise par le produit des deux polynômes de degrés k et $k-1$ dont finalement, seuls les k premiers monômes sont utiles. Comme nous l'avons par ailleurs discuté, ceci met en évidence l'importance de la résolution de ce produit, connu sous le nom de *short product* sur des FNT réduites à une taille k. Si cette question ouverte était résolue, le gain théorique de vitesse pourrait atteindre 67% lorsque le taux de codage est de $\frac{1}{2}$.

6.4 Résultats de simulation

Nous avons mené des simulations sur le code RS-FNT que nous avons comparé à différents codes de Reed-Solomon classiques :

- L'implémentation classique des codes de Reed-Solomon par Luigi Rizzo [15], sur des matrices de Vandermonde, reconnue pour son efficacité sur des petites tailles de codes. Ce code a été testé en utilisant des tailles de corps adaptées.
- La construction XOR-based [18] des codes de Reed-Solomon grâce à des matrices de Cauchy. De par sa construction, ce code permet de rester proche de la complexité quadratique intrinsèque des codes de Reed-Solomon. Ce code a également été testé en ajustant au mieux la taille du corps en fonction de la taille du code.
- Les codes MDS basés sur des matrices de Cauchy [19] sur le corps $\mathbb{F}_{65537}$ développés à l'ISAE. Le corps fini utilisé est donc identique à celui des RS-FNT. La différence avec les codes précédents vient donc de l'arithmétique effectuée sur ce code. Ce code est par ailleurs particulièrement adapté à une architecture 64-bit.

Pour le code RS-FNT, nous utilisons un encodeur/décodeur de complexité quadratique tel que présenté dans la partie 6.2.5 pour les petites tailles de codes. L'algorithme est alors très similaire aux autres codes. L'encodeur/décodeur de complexité logarithme prend alors le relais lorsque les tailles impliquées dépassent la centaine

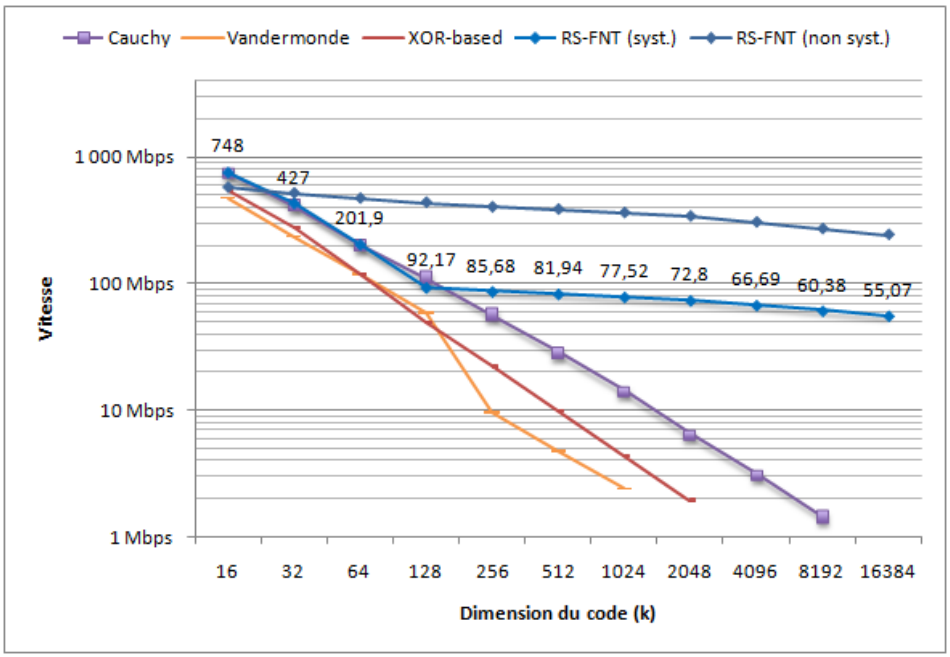

Figure 6.3 – Vitesse d'encodage des différents codes de Reed-Solomon et du code RS-FNT, en fonction de la dimension du code, taux de codage $\frac{1}{2}$, symboles de 1500 octets

d'éléments.

Les simulations ont été réalisées sur une architecture Intel Core 2 Extreme @3.06GHz / 4Go RAM sur Mac OS X 10.6 en 64-bit.

Sur la Figure 6.3, nous avons représenté la vitesse d'encodage des différents codes de Reed-Solomon en fonction de la dimension du code k, pris comme une puissance de deux. Le rendement du code est égal à $\frac{1}{2}$et la taille des symboles est de 1500 octets. Pour le code RS-FNT, nous avons représenté l'encodeur en mode systématique, mais aussi en mode non-systématique.

Lorsque les dimensions de codes en jeu sont inférieures à $k = 128$, l'encodeur RS-FNT obtient des performances similaires aux codes de Reed-Solomon. Lorsque cette dimension augmente, les encodeurs classiques sont pénalisés par leur complexité quadratique, et deviennent inutilisables en pratique, alors que le code RS-FNT systématique permet de soutenir des vitesses supérieures à 50Mbps, y compris pour des dimensions de codes supérieures à 10.000 éléments. Ces résultats permettent également de valider la complexité pratique en $\mathcal{O}(n \log n)$ du code, et confirme la négligibilité du facteur en $\mathcal{O}(k \log^2 k)$ lors du passage au mode paquet/symbole.

Ces résultats mettent, par ailleurs, en lumière l'impact de l'encodage systématique qui réduit la vitesse d'encodage d'un facteur d'environ 4.5. Ce résultat expérimental permet de valider le facteur 5 attendu grâce à l'analyse théorique du Tableau 6.2. En

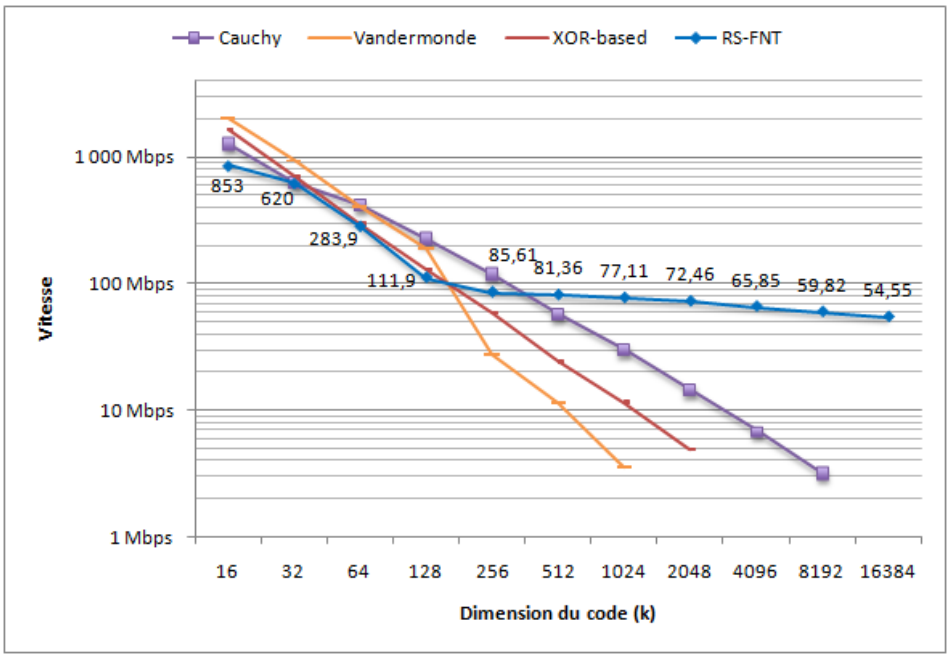

Figure 6.4 – Vitesse de décodage des différents codes de Reed-Solomon et du code RS-FNT, en fonction de la dimension du code, taux de codage $\frac{1}{2}$, symboles de 1500 octets

mode non-systématique, l'encodeur RS-FNT est capable de fonctionner à des vitesses de l'ordre de 200-300Mbps, ce qui peut être un atout considérable dans le cadre, par exemple, de stockage distribué, où le débit en écriture est crucial.

De manière similaire, sur la Figure 6.4, nous avons représenté la vitesse de décodage correspondante. Nous n'avons pas ici représenté le décodeur RS-FNT non systématique, celui-ci n'ayant une vitesse que légèrement supérieure à celle de la version systématique.

Les résultats obtenus sont similaires à ceux de l'encodeur, le code RS-FNT prenant l'avantage sur les autres codes dès que la dimension du code dépasse 256. Comme attendu lors de l'analyse de complexité, les vitesses de l'encodeur et du décodeur du code RS-FNT sont sensiblement équivalentes et toujours supérieures à 50Mbps.

Les opérations de poinçonnage/padding n'ayant que peu d'influence sur les performances du code RS-FNT, les résultats d'expérimentation ont montré une tendance similaire pour des dimensions et tailles de codes différentes d'une puissance de deux.

6.5 Conclusion

Dans ce chapitre, nous avons présenté un code de Reed-Solomon sur un corps fini dont la dimension est un nombre de Fermat premier. Sur ce corps, nous avons pu développer un encodeur et un décodeur systématique de complexité logarithmique, et qui permet donc d'atteindre la capacité des codes MDS. Nous avons également mené une analyse de complexité afin de déterminer le coût exact du code exprimés en nombre de FNT.

Les résultats expérimentaux ont permis de valider cette analyse de complexité, tout en mettant en lumière l'intérêt d'un tel code MDS lorsque les dimensions des codes dépassent la centaine de symboles. Les simulations ont également montré que les codes RS-FNT restent utilisables en pratique jusqu'à des tailles supérieures à 10.000 symboles, avec des vitesses supérieures à 50Mbps. Ceci ouvre donc tout un champ de possibilités concernant le codage optimal, et le partage de l'information et du secret sur un ensemble comportant un très grand nombre de systèmes. Les vitesses atteintes par l'encodeur non-systématique ($> 200 Mbps$) permettent également d'envisager ce code comme une réponse au besoin de stockage et de sauvegarde distribués sur un grand nombre de plate-formes. En effet, ces problématiques requièrent un coût minimal lors de la phase d'écriture, tout en étant plus souple lors de la phase d'accès.

Parmi les nombreuses pistes d'optimisation envisagées pour ce code, nous avons étudié la possibilité d'utiliser les architectures massivement parallèles des cartes graphiques (solutions de type GPU, CUDA,...) sur lesquelles des librairies de FFT optimisées sont disponibles. Nous nous sommes malheureusement heurtés aux contraintes de temps de transfert entre le CPU et le GPU, rendant caduque ce calcul via le GPU. De la même manière, nous n'avons pas pu intégrer de manière transparente les librairies de multiplication polynômiales optimisée [86], du fait du corps spécifique sur lequel nous travaillons.

Cependant quelques pistes d'optimisation sont encore envisageables. Tout d'abord au niveau théorique, comme nous l'avons vu précédemment, la question du *short product* restant une question ouverte, sa résolution pourrait conditionner un gain de vitesse théorique pouvant atteindre 67%. Du point de vue de l'implémentation, de récents résultats [87] concernant l'implémentation de la FNT au niveau matériel, tout comme la FFT, permettent d'espérer un gain considérable sur les vitesses attendues, qui permettrait alors aux codes RS-FNT de concurrencer les solutions non-MDS en termes de vitesse, tout en proposant un codage optimal.

Ce code est publié librement sous licence LGPL sur [88]. Il est disponible dans la librairie OpenFEC [89].

Chapitre 7

Conclusion et perspectives

7.1 Résumé des contributions et perspectives futures

Dans ce chapitre, nous résumons les principales contributions de ce manuscrit et les pistes d'étude envisagées.

Tout d'abord, nous avons présenté un modèle mathématique pour les mécanismes de compression d'en-têtes protocolaires. Nous sommes parvenus à modéliser le comportement de mécanismes unidirectionnels et également à voie de retour. Nous avons également pu appliquer ce modèle au protocole ROHC qui est à ce jour un mécanisme sophistiqué et largement déployé, mais pénalisé par sa relative complexité. Au prix de quelques restrictions, nous sommes parvenus à comparer les modes avec et sans voie de retour de ROHC et ce, dans différents scénarios d'exploitation. A ce jour, le modèle présenté reste le seul outil mathématique permettant une comparaison complète des mécanismes de compression unidirectionnels avec et sans voie de retour.

Les perspectives concernant ce modèle consistent principalement à adapter de manière plus précise les mécanismes spécifiques aux protocoles visés. Une autre approche complémentaire serait aussi de permettre une modélisation efficace des nouvelles implémentations de ROHC (ROHCv2). Le but serait alors de permettre une utilisation plus adaptée de ces protocoles en fonction des cas d'exploitations souhaités.

Par la suite, nous avons présenté trois contributions sur les codes à effacements. La première est un code à effacements original basé sur un décodage hybride de la matrice génératrice et de la matrice de parité. En contraignant la partie non-systématique du code dans une bande, nous sommes parvenus à augmenter significativement la vitesse de décodage de ces codes LDPC particuliers, dans les cas difficiles, sans pour autant impacter grandement la capacité de correction du code. Ce code LDPC particulier se présente donc comme une alternative intéressante aux codes LDPC classiques.

Néanmoins, nous n'avons pas utilisé le fait que la structure de la matrice de parité de ces codes était proche de celle d'une matrice bande également. Plus précisément, elle peut être ramenée à la concaténation de deux matrices bandes. Ceci ouvre donc des perspectives intéressantes pour ces codes, notamment pour une éventuelle adaptation du décodage ML directement sur la matrice bande, ce qui permettra de diminuer largement les contraintes associées à ces codes. Il est également envisageable, suite à ces adaptations, de pouvoir travailler plus aisément sur des méthodes d'optimisation basées sur des évolutions de densité.

Nous avons également présenté un décodage optimisé pour les codes de Reed-Muller, sur le canal à effacements. Ces optimisations ont permis d'améliorer de manière conséquente la capacité de correction de ces codes, tout en conservant une complexité mathématique faible. Ainsi, sur des codes de petite taille, ces codes se présentent dès lors comme un véritable concurrent aux codes de Reed-Solomon, lorsque l'aspect MDS de ces codes n'est pas un facteur crucial, les codes de type LDPC étant alors pénalisés par une forte inefficacité.

La question de la recherche de permutations efficaces avec une complexité limitée reste cependant un point ouvert. En effet, lors de nos simulations, nous avons constaté que la complexité de cette recherche empêchait d'améliorer la capacité de ces codes. En supposant qu'un algorithme de recherche efficace, de complexité même quadratique existe, ceci permettrait d'améliorer la capacité de ces codes, sans pour autant augmenter leur complexité.

Par la suite, nous avons présenté des codes MDS basés sur l'application de la transformée de Fourier sur des corps finis dont le cardinal est un nombre de Fermat premier. Un encodage et un décodage de complexité logarithmique ont été présentés pour ces codes, appelés RS-FNT. Ceci permet à ces codes d'être les premiers codes sur le canal à effacements à atteindre la limite de complexité théorique des codes MDS. Ces codes ont montré qu'il était possible d'utiliser de manière pratique des codes MDS sur des tailles pouvant atteindre plusieurs milliers de symboles.

Si la limite théorique des codes MDS a été atteinte par ces codes, il reste cependant un travail significatif à effectuer sur la constante cachée par cette complexité. Grâce à une analyse théorique, nous sommes parvenus à évaluer la complexité de ce code en fonction du nombre de FFT à calculer. Ceci a permis de mettre en évidence les points à améliorer du code, notamment la question de l'optimisation du *short product*, qui offre potentiellement un gain important. De la même manière, un travail important reste à effectuer sur l'optimisation des FFT elles-mêmes sur ces corps, notamment grâce à l'intérêt du calcul parallélisé.

Les perspectives et les enjeux de ces codes sont probablement les plus importants parmi les contributions présentées ici. Si les codes RS-FNT se présentent comme une alternative intéressante théoriquement, par rapport aux codes de type LDPC ou Raptor pour des grandes longueurs de code, le rapport de vitesse de l'ordre de 40x en défaveur de ces codes MDS, limite malheureusement l'intérêt de ces codes. En supposant que les optimisations évoquées puissent réduire ce ratio en dessous d'un ordre de grandeur, les codes RS-FNT deviendraient alors un concurrent plus que sérieux aux codes LDPC et Raptor, surtout en prenant compte de l'évolution du calcul parallélisé et des évolutions supposées des capacités de calcul.

7.2 Contributions diverses

7.2.1 Publications

- Soro, Alexandre and Lacan, Jérôme and Chaput, Emmanuel and Donny, Christophe and Baudoin, Cédric (2007) Evaluation of a generic unidirectional header compression protocol. In : International Workshop on Satellite and Space Communications - IWSSC '07, 13-14 Sept 2007, Salzburg, Austria.
- Cunche, Mathieu and Savin, Valentin and Roca, Vincent and Kraidy, G. and Soro, Alexandre and Lacan, Jérôme (2008) Low-rate coding using incremental redundancy for GLDPC codes. In : International Workshop on Satellite and Space Communications (IWSSC'08), 01-03 Oct 2008, Toulouse, France.
- Soro, Alexandre and Cunche, Mathieu and Lacan, Jérôme and Roca, Vincent (2009) Erasure codes with a banded structure for hybrid iterative-ML decoding. In : IEEE Global Communication Conference. Globecom 2009 , 30 Nov - 04 Dec 2009, Honolulu, Hawai, United States.
- Soro, Alexandre and Lacan, Jérôme (2009) FNT-based reed-solomon erasure codes. In : 7th Annual IEEE Consumer Communications and Networking Conference , 09-12 Jan 2010, Las Vegas, United States.

En cours de soumission :

- Soro, Alexandre and Lacan, Jérôme and Chaput, Emmanuel and Baudoin, Cédric "Evaluation of a generic header compression protocol".

7.2.2 Logiciels

- Codes de Reed-Muller pour le canal à effacements de paquets : http://personnel.isae.fr/jerome-lacan/reed-muller-erasure-codes.html
- Codes à effacements MDS basés sur les FNT : http://personnel.isae.fr/jerome-lacan/article/fnt-based-reed-solomon-codes.html
- Codes Raptor (projet CNES - 2008)
- Contributions à OpenFEC.org : http://www.openfec.org/

Bibliographie

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Sys. Tech. J.*, vol. 27, pp. 379–423, 623–656, 1948. 1, 5

[2] P. Elias, "Coding for two noisy channels," *Information Theory, The 3rd London Symposium*, pp. 61–76, 1955. 7

[3] R. Singleton, "Maximum distance q-nary codes," *IEEE Transactions on Information Theory*, vol. 10, pp. 116–118, 1964. 11

[4] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979. 11, 97

[5] R. J. McEliece and D. V. Sarwate, "On sharing secrets and reed-solomon codes," *Commun. ACM*, vol. 24, no. 9, pp. 583–584, 1981. 11

[6] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, 1989. 11, 14, 97

[7] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960. 11

[8] E. R. Berlekamp, "Nonbinary BCH decoding," *IEEE Transactions on Information Theory*, vol. 14, pp. 242–242, 1968. 12

[9] ——, *Algebraic coding theory*. McGraw-Hill, 1968. 12

[10] J. Massey, "Shift-register synthesis and BCH decoding," *Information Theory, IEEE Transactions on*, vol. 15, pp. 122–127, 1969. 12

[11] R. Blahut, "Algebraic fields, signal processing, and error control," *Proceedings of the IEEE*, vol. 73, no. 5, pp. 874–893, May 1985. 12, 98

[12] F. Didier, "Efficient erasure decoding of reed-solomon codes," *CoRR*, vol. abs/0901.1886, 2009. 12, 98

[13] J. S. Plank, "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems," *Software – Practice & Experience*, vol. 27, no. 9, pp. 995–1012, September 1997. 12, 97

[14] J. S. Plank and Y. Ding, "Note : Correction to the 1997 tutorial on reed-solomon coding," University of Tennessee, Tech. Rep. CS-03-504, April 2003. 12, 97

[15] L. Rizzo, "Effective Erasure Codes For Reliable Computer Communication Protocols," *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, April 1997. 12, 111

[16] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes," RFC 5510 (Proposed Standard), Internet Engineering Task Force, Apr. 2009. [Online]. Available : http://www.ietf.org/rfc/rfc5510.txt 12

[17] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. North-Holland, 1977. 13, 89, 90

[18] J. Bloemer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme," in *Technical Report ICSI TR-95-048*, August 1995. 14, 111

[19] L. Dairaine, L. Lancérica, J. Lacan, and J. Fimes, "Content-Access QoS in Peer-to-Peer Networks Using a Fast MDS Erasure Code," *Computer Communications*, vol. 28, no. 15, pp. 1778–1790, september 2005. 14, 111

[20] R. G. Gallager, "Low Density Parity Check Codes," Ph.D. dissertation, MIT, 1963. 15

[21] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, pp. 1645–1646, 1996. 16

[22] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, pp. 56–67, 1998. 16

[23] M. Luby, "LT codes," *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pp. 271–280, 2002. 16, 22

[24] A. Shokrollahi, "Raptor codes," *IEEE/ACM Trans. Netw.*, vol. 14, no. SI, pp. 2551–2567, 2006. 16, 23

[25] V. V. Zyablov and M. S. Pinsker, "Decoding complexity of Low-Density Codes for transmission in a channel with erasures," *Probl. Peredachi Inf.*, vol. 10, pp. 15–28, 1974. 16

[26] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory,*, vol. 27, no. 5, pp. 533–547, 1981. 16

[27] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 638–656, 2001. 18

[28] D. Divsalar, H. Jin, and R. J. McEliece, "Coding theorems for 'turbo-like' codes„" *in Proc. 36th Annu. Allerton Conf. Commun., Contr., Comput.*, pp. 201–210, 1998. 20

[29] V. Roca, C. Neumann, and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes," RFC 5170 (Proposed Standard), Internet Engineering Task Force, Jun. 2008. [Online]. Available : http://www.ietf.org/rfc/rfc5170.txt 20, 74

[30] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery," RFC 5053 (Proposed Standard), Internet Engineering Task Force, Oct. 2007. [Online]. Available : http://www.ietf.org/rfc/rfc5053.txt 22, 23

[31] "3GPP. Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs. TS 26.346, 3rd Generation Partnership Project (3GPP)." 23

[32] "ETSI "IP Datacast over DVB-H : Content Delivery Protocols (CDP)". Technical Report TS 102 472 v1.2.1, ETSI, 2006." 23

[33] A. Shokrollahi, S. Lassen, and R. Karp, "Systems and processes for decoding chain reaction codes through inactivation, US Patent 6,856,263." 26

[34] M. Luby, T. Gasiba, T. Stockhammer, and M. Watson, "Reliable Multimedia Download Delivery in Cellular Broadcast Networks," *Broadcasting, IEEE Transactions on*, vol. 53, pp. 235–246, 2007. 27

[35] V. Suryavanshi and A. Nosratinia, "Convolutional coding for resilient packet header compression," *in Proc. Global Telecommunications Conference, GLOBECOM '05*, 2005. 27

[36] D. Farber, G. Delp, and T. Conte, "Thinwire protocol for connecting personal computers to the Internet," RFC 914 (Historic), Internet Engineering Task Force, Sep. 1984. [Online]. Available : http://www.ietf.org/rfc/rfc914.txt 28

[37] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links," RFC 1144 (Proposed Standard), Internet Engineering Task Force, Feb. 1990. [Online]. Available : http://www.ietf.org/rfc/rfc1144.txt 28

[38] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep. 2001. [Online]. Available : http://www.ietf.org/rfc/rfc3168.txt 30

[39] M. Degermark, B. Nordgren, and S. Pink, "IP Header Compression," RFC 2507 (Proposed Standard), Internet Engineering Task Force, Feb. 1999. [Online]. Available : http://www.ietf.org/rfc/rfc2507.txt 31, 65

[40] S. Casner and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links," RFC 2508 (Proposed Standard), Internet Engineering Task Force, Feb. 1999. [Online]. Available : http://www.ietf.org/rfc/rfc2508.txt 31

[41] T. Koren, S. Casner, J. Geevarghese, B. Thompson, and P. Ruddy, "Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering," RFC 3545 (Proposed Standard), Internet Engineering Task Force, Jul. 2003. [Online]. Available : http://www.ietf.org/rfc/rfc3545.txt 31

[42] A.-V. T. W. Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP : A Transport Protocol for Real-Time Applications," RFC 1889 (Proposed Standard), Internet Engineering Task Force, Jan. 1996, obsoleted by RFC 3550. [Online]. Available : http://www.ietf.org/rfc/rfc1889.txt 33

[43] C. Bormann, C. Burmeister, M. Degermark, H. Fukushima, H. Hannu, L.-E. Jonsson, R. Hakenberg, T. Koren, K. Le, Z. Liu, A. Martensson, A. Miyazaki, K. Svanbro, T. Wiebke, T. Yoshimura, and H. Zheng, "RObust Header Compression (ROHC) : Framework and four profiles : RTP, UDP, ESP, and uncompressed," RFC 3095 (Proposed Standard), Internet Engineering Task Force, Jul. 2001, updated by RFCs 3759, 4815. [Online]. Available : http://www.ietf.org/rfc/rfc3095.txt 37

[44] L.-E. Jonsson and G. Pelletier, "RObust Header Compression (ROHC) : A Compression Profile for IP," RFC 3843 (Proposed Standard), Internet Engineering Task Force, Jun. 2004, updated by RFC 4815. [Online]. Available : http://www.ietf.org/rfc/rfc3843.txt 37

[45] G. Pelletier, "RObust Header Compression (ROHC) : Profiles for User Datagram Protocol (UDP) Lite," RFC 4019 (Proposed Standard), Internet Engineering Task Force, Apr. 2005, updated by RFC 4815. [Online]. Available : http://www.ietf.org/rfc/rfc4019.txt 37

[46] G. Pelletier, K. Sandlund, L.-E. Jonsson, and M. West, "RObust Header Compression (ROHC) : A Profile for TCP/IP (ROHC-TCP)," RFC 4996 (Proposed Standard), Internet Engineering Task Force, Jul. 2007. [Online]. Available : http://www.ietf.org/rfc/rfc4996.txt 37

[47] L.-E. Jonsson, K. Sandlund, G. Pelletier, and P. Kremer, "RObust Header Compression (ROHC) : Corrections and Clarifications to RFC 3095," RFC 4815 (Proposed Standard), Internet Engineering Task Force, Feb. 2007. [Online]. Available : http://www.ietf.org/rfc/rfc4815.txt 37

[48] L.-E. Jonsson, G. Pelletier, and K. Sandlund, "The RObust Header Compression (ROHC) Framework," RFC 4995 (Proposed Standard), Internet Engineering Task Force, Jul. 2007, obsoleted by RFC 5795. [Online]. Available : http://www.ietf.org/rfc/rfc4995.txt 37

[49] G. Pelletier and K. Sandlund, "RObust Header Compression Version 2 (ROHCv2) : Profiles for RTP, UDP, IP, ESP and UDP-Lite," RFC 5225 (Proposed Standard), Internet Engineering Task Force, Apr. 2008. [Online]. Available : http://www.ietf.org/rfc/rfc5225.txt 37

[50] K. Sandlund, G. Pelletier, and L.-E. Jonsson, "The RObust Header Compression (ROHC) Framework," RFC 5795 (Proposed Standard), Internet Engineering Task Force, Mar. 2010. [Online]. Available : http://www.ietf.org/rfc/rfc5795.txt 37

[51] C. Y. Cho, W. K. G. Seah, and Y. H. Chew, "A framework and source model for design and evaluation of robust header compression," *Comput. Netw.*, vol. 50, no. 15, pp. 2676–2712, 2006. 42

[52] C. Y. Cho, Y. H. Chew, and W. K. G. Seah, "Modeling and analysis of robust header compression performance," in *WOWMOM '05 : Proceedings of the Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks*. Washington, DC, USA : IEEE Computer Society, 2005, pp. 192–197. 42

[53] A. Minaburo, "Compression des en-têtes dans les réseaux haut-débit," Ph.D. dissertation, Université de Rennes I, December 2003. 42

[54] A. Couvreur, L.-M. Le Ny, A. Minaburo, G. Rubino, B. Sericola, and L. Toutain, "Performance analysis of an header compression protocol : The rohc unidirectional mode," INRIA, Research Report RR-5300, 2004. [Online]. Available : http://hal.inria.fr/inria-00070700/PDF/RR-5300.pdf 42

[55] E. N. Gilbert, "Capacity of a burst-noise channel," *The Bell System Technical Journal*, vol. 39, pp. 1253–1265, Sep. 1960. 46

[56] E. O. Elliott, "Estimates of error rates for codes on burst-noise channels," *The Bell System Technical Journal*, vol. 42, pp. 1977–1997, Sep. 1963. 46

[57] "ETSI "Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services" European Standard EN 300 421 V1.1.2, ETSI, 1997." 46

[58] "ETSI "Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)" EN 302 307 V1.2.1, ETSI, 2009." 46

[59] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, "Measurement and modeling of the temporal dependence in packet loss," in *INFOCOM*, 1999, pp. 345–352. 46

[60] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley, "Adaptive fec-based error control for internet telephony," *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1453–1460 vol.3, Mar 1999. 46

[61] M. Degermark, M. Engan, B. Nordgren, and S. Pink, "Low-loss tcp/ip header compression for wireless networks," *In Proceedings from MobiCom*, 1996. 65

[62] J. Aikat, J. Kaur, F. Donelson, and S. K. Jeffay, "Variability in tcp round-trip times," *3rd ACM SIGCOMM conference on Internet measurement*, 2003. 68

[63] F. Vacirca, F. Ricciato, and R. Pilz, "Large-scale RTT measurements from an operational umts/gprs network," *First International Conference on Wireless Internet*, pp. 190–197, 2005. 69

[64] G. Sawar, E. Lochin, and R. Boreli, "Experimental performance of dccp over live satellite and long range wireless links," *International Symposium on Communications and Information Technologies (ISCIT07)*, pp. 689–694, Oct. 2007. 69

[65] J. A. Perez, B. Donnet, and O. Bonaventure, "Preliminary analysis of the tcp behavior in 802.16 networks," *1st WEIRD Workshop on WiMAX, Wireless and Mobility*, May 2007. 69

[66] C. Studholme and I. Blake, "Windowed erasure codes," *Information Theory, 2006 IEEE International Symposium on*, pp. 509–513, July 2006. 72

[67] ——, "Random matrices and codes for the erasure channel," *Algorithmica*, April 2008. 72

[68] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, 2001. 79

[69] G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed. Johns Hopkins University Press, 1996. 80

[70] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme." *IEEE Transactions on Information Theory*, vol. 4, pp. 38–49, 1954. 85

[71] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection." *IEEE Transactions on Computers*, vol. 3, pp. 6–12, 1954. 85

[72] I. Dumer and K. Shabunov, "Soft-decision decoding of reed-muller codes : recursive lists." *IEEE Transactions on Information Theory*, vol. 52, pp. 1260–1266, 2006. 85

[73] C. Hänle, "Feasibility study of erasure correction for multicast file distribution using the network simulator ns-2," *IEEE Military Communications Conference*, vol. 3, pp. 1260–1266, 1998. 85

[74] M. Plotkin, "Binary codes with specified minimum distance," *IRE Transactions on Information Theory*, vol. 6, pp. 445–450, 1960. 86

[75] T. Hehn, O. Milenkovic, S. Laendner, and J. B. Huber, "Permutation decoding and the stopping redundancy hierarchy of cyclic and extended cyclic codes." *IEEE Transactions on Information Theory*, no. 12, pp. 5308–5331, 2008. 90

[76] http://personnel.isae.fr/jerome-lacan/reed-muller-erasure-codes.html. 95

[77] V. Strassen, "Die berechnungskomplexität von elementarsymmetrischen funktionen und von interpolationskoeffizienten," *j-NUM-MATH*, vol. 20, no. 3, pp. 238–251, jun 1973. 98

[78] J. M. Pollard, "The Fast Fourier Transform in a finite field," *Mathematics of Computation*, vol. 25, no. 114, pp. 365–374, Apr. 1971. 98, 99

[79] J. M. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Math. Comp.*, vol. 19, p. 297, 1965. 99

[80] A. Karatsuba and Y. Offman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, pp. 595–596, 1963. 103

[81] A. L. Toom, "The complexity of a scheme of functional elements realizingthe multiplication of integers," *Translations of Soviet Mathematics SSSR 4*, 1963. 103

[82] S. A. Cook, "On the minimum computation time of functions," Ph.D. dissertation, Department of Mathematics, Harvard University, 1966. 103

[83] A. Schonhage and V. Strassen, "Schnelle multiplikation grosser zahlen," *Computing*, vol. 7, pp. 281–292, 1971. 103

[84] T. Mulders, "On short multiplications and divisions," *AAECC*, vol. 11, pp. 69–88, 2000. 106

[85] G. Hanrot and P. Zimmermann, "A long note on Mulders' short product," *Journal of Symbolic Computation*, vol. 37, pp. 391–401, 2004. [Online]. Available : http://hal.inria.fr/inria-00100069/en/ 106

[86] "The GNU MP Bignum Library." [Online]. Available : http://gmplib.org/ 114

[87] S. Li and J. Zhang, "Area-delay efficient parallel architecture for fermat number transform," *IEICE Electronics Express*, vol. 6, no. 8, pp. 449–455, 2009. 114

[88] "FNT-based codes." [Online]. Available : http://pagespro.isae.fr/jerome-lacan/ 114

[89] "The OpenFEC project." [Online]. Available : http://www.openfec.org/ 114