Groupe de travail Réseau
Request for Comments : 5170
Catégorie : Sur la voie de la normalisation
Traduction Claude Brière de L'Isle

V. Roca, INRIA
C. Neumann, Thomson
D. Furodet, STMicroelectronics

juin 2008

Schémas d'escalier et de triangle de vérification de parité à basse densité (LDPC) pour la correction d'erreur directe (FEC)

Statut du présent mémoire

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

Résumé

Le présent document décrit deux schémas pleinement spécifiés de correction d'erreur directe (FEC, *Forward Error Correction*) l'escalier de vérification de parité à basse densité (LDPC, *Low Density Parity Check*) et le triangle LDPC, et leur application à la livraison fiable d'objets de données sur le canal à écrasement de paquets (c'est-à-dire, un chemin de communication où les paquets sont soit reçus sans aucune corruption, soit éliminés durant la transmission). Ces codes systématiques de FEC appartiennent à la classe bien connue des codes de "vérification de parité à basse densité", et sont des codes de grand bloc de FEC au sens de la RFC 3453.

Table des Matières

1. Introduction

La [RFC3453] introduit des codes de FEC de grands blocs comme solution de remplacement aux codes de FEC de petits blocs comme Reed-Solomon. Le principal avantage de ces codes de grand bloc est la possibilité d'opérer efficacement sur les blocs de source d'une taille de plusieurs dizaines de milliers (ou plus) de symboles de source. Le présent document introduit l'identifiant 3 de codage de FEC pleinement spécifiée qui est destiné à l'utilisation avec les codes de FEC LDPC-Staircase, et l'identifiant 4 de codage de FEC pleinement spécifiée qui est destiné à l'utilisation avec les codes de FEC LDPC-Triangle [RN04], [MK03]. Les deux schémas appartiennent à la large classe des codes de grand bloc. Pour une définition du terme de schéma pleinement spécifié, voir la Section 4 de la [RFC5052].

Les codes LDPC s'appuient sur une matrice dédiée, appelée une "matrice de vérification de parité", aux extrémités de codage et de décodage. La matrice de vérification de parité définit les relations (ou contraintes) entre les divers symboles de codage (c'est-à-dire, les symboles de source et les symboles de réparation) qui sont ensuite utilisés par le décodeur pour reconstruire les k symboles de source d'origine si certains d'entre eux manquent. Ces codes sont systématiques, au sens où les symboles de codage incluent les symboles de source en plus des symboles de réparation.

Comme le codeur et le décodeur doivent opérer sur la même matrice de vérification de parité, les informations doivent être communiquées entre elles au titre des informations de transmission d'objet de FEC.

Une mise en œuvre de référence publiquement disponible de ces codes est disponible et distribuée sous licence GNU/LGPL [LDPC-codec]. À côté de cela, les extraits de code inclus dans le présent document sont offerts directement comme contribution au processus de l'IETF par les auteurs de ce document et par Radford M. Neal.

2. Notation des exigences

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

3. Définitions, notations, et abréviations

3.1 Définitions

Le présent document utilise les mêmes termes et définitions que spécifiés dans la [RFC5052]. De plus, il utilise les définitions suivantes :

Symbole de source : unité de données utilisée durant le processus de codage.

Symbole de codage : unité de données générée par le processus de codage.

Symbole de réparation : symbole de codage qui n'est pas un symbole de source.

Taux de code : c'est le ratio k/n, c'est-à-dire, le rapport entre le nombre de symboles de source et le nombre de symboles de codage. Le taux de code appartient à l'intervalle]0, 1]. Un taux de code proche de 1 indique qu'un petit nombre de symboles de réparation a été produit durant le processus de codage

Code systématique : code de FEC dans lequel les symboles de source font partie des symboles de codage.

Bloc de source : bloc de k symboles de source qui sont considérés ensemble pour le codage.

Groupe de symboles de codage : groupe de symboles de codage qui sont envoyés ensemble, au sein du même paquet, et dont les relations à l'objet de source peuvent être déduites d'un seul identifiant de symbole de codage.

Paquet de source : paquet de données contenant seulement des symboles de source.

Paquet: de réparation : paquet de données contenant seulement des symboles de réparation.

Canal à écrasement de paquet : chemin de communication où les paquets sont soit abandonnés (par exemple, par un routeur encombré ou parce que le nombre d'erreurs de transmission excède les capacités de correction des codes de la couche physique) soit reçus. Quand un paquet est reçu, il est supposé que ce paquet n'est pas corrompu.

3.2 Notations

Le présent document utilise les notations suivantes :
L note la longueur de transfert d'objet en octets.
k note la longueur de bloc de source en symboles, c'est-à-dire, le nombre de symboles de source d'un bloc de source.
n note la longueur de bloc de codage, c'est-à-dire, le nombre de symboles de codage générés pour un bloc de source.
E note la longueur de symbole de codage en octets.
B note la longueur maximum de bloc de source en symboles, c'est-à-dire, le nombre maximum de symboles de source par bloc de source.
N note le nombre de blocs de source dans lesquels l'objet devra être partagé.
G note le nombre de symboles de codage par groupe, c'est-à-dire, le nombre de symboles envoyés dans le même paquet.
CR note le "taux de code", c'est-à-dire, le ratio k/n.
max_n note le nombre maximum de symboles de codage générés pour tout bloc de source. C'est en particulier le nombre de symboles de codage générés pour un bloc de source de taille B.
H note la matrice de vérification de parité.
N1 note le nombre cible de "1" par colonne dans le côté gauche de la matrice de vérification de parité.
N1m3 note la valeur N1 - 3, où N1 est le nombre cible de "1" par colonne du côté gauche de la matrice de vérification de parité.
pmms_rand(m) note le générateur de nombres pseudo-aléatoires défini au paragraphe 5.7 qui retourne un nouvel entier aléatoire dans l'intervalle [0, m-1] chaque fois qu'il est invoqué.

3.3 Abréviations

Le présent document utilise les abréviations suivantes :
ESI *(Encoding Symbol Identifier)* : identifiant de symbole de codage
FEC OTI *(FEC Object Transmission Information)* : informations de transmission d'objet de FEC
FPI *(FEC Payload Identifier)* : identifiant de charge utile de FEC
LDPC *(Low Density Parity Check)* : vérification de parité à faible densité
PRNG *(Pseudo-Random Number Generator)* : générateur de nombre pseudo aléatoire

4. Formats et codes

4.1 Identifiants de charge utile de FEC

L'identifiant de charge utile de FEC est composé du numéro de bloc de source et de l'identifiant de symbole de codage.

Le numéro de bloc de source (champ de 12 bits) identifie à partir de quel bloc de source bloc de l'objet le ou les symboles de codage dans la charge utile du paquet sont générés. Il y a un maximum de 2^^12 blocs par objet. Le numérotage des blocs de source commence à 0.

L'identifiant de symbole de codage (champ de 20 bits) identifie quels symboles de codage générés à partir du bloc de source sont portés dans la charge utile de paquet. Il y a un maximum de 2^^20 symboles de codage par bloc. Les k premières valeurs (0 à k-1) identifient les symboles de source, les n-k valeurs restantes (de k à n-k-1) identifient les symboles de réparation.

Il DOIT y avoir exactement un identifiant de charge utile de FEC par paquet. Dans le cas d'un groupe de symboles de codage, quand plusieurs symboles de codage sont envoyés dans le même paquet, l'identifiant de charge utile de FEC se réfère au premier symbole du paquet. Les autres symboles peuvent être déduits de l'ESI du premier symbole grâce à une fonction dédiée, comme expliqué au paragraphe 5.6.

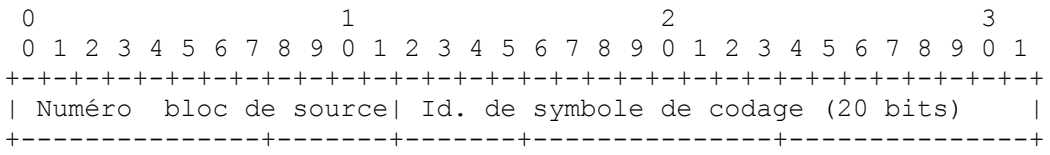


Figure 1 : Identifiant de charge utile de format de codage de FEC pour les Id de codage de FEC 3 et 4

4.2 Informations de transmission d'objet de FEC

4.2.1 Élément obligatoire

Identifiant de codage de FEC : les schémas de FEC pleinement spécifiés LDPC-Staircase et LDPC-Triangle utilisent l'identifiant de codage de FEC 3 (escalier) et 4 (triangle), respectivement.

4.2.2 Éléments communs

Les éléments suivants DOIVENT être définis avec les présents schémas de FEC :

- Longueur de transfert (L) : entier non négatif qui indique la longueur de l'objet en octets. Il y a des restrictions sur la longueur maximum de transfert qui peut être prise en charge :

  longueur maximum de transfert = 2^^12 * B * E

  Par exemple, si B=2^^19 (à cause d'un taux de code de 1/2, paragraphe 5.2) et si E=1024 octets, alors la longueur maximum de transfert est 2^^41 octets (ou 2 TB). La limite supérieure, avec des symboles de taille 2^^16 -1 octets et un taux de code supérieur ou égal à 1/2, donne 2^^47 octets (ou 128 TB).

- Longueur de symbole de codage (E) : entier non négatif indiquant la longueur de chaque symbole de codage en octets.

- Longueur maximum de bloc de source (B) : entier non négatif indiquant le nombre maximum de symboles de source dans un bloc de source. Il y a des restrictions sur la valeur maximum de B, comme expliqué au paragraphe 5.2.

- Nombre maximum de symboles de codage (max_n) : entier non négatif indiquant le nombre maximum de symboles de codage généré pour tout bloc de source. Il y a des restrictions sur la valeur maximum de max_n. En particulier max_n est au plus égal à 2^^20.

La Section 5 explique comment définir les valeurs de chacun de ces éléments.

4.2.3 Éléments spécifiques de schéma

Les éléments suivants DOIVENT être définis avec le présent schéma de FEC :

N1m3 : entier entre 0 (par défaut) et 7, inclus. Le nombre cible de "1" par colonne dans le côté gauche de la matrice de vérification de parité, N1, est alors égal à N1m3 + 3 (voir les paragraphes 6.2 et 7.2). Utiliser la valeur par défaut de 0 pour N1m3 est recommandé quand l'envoyeur n'a pas d'information sur le schéma de décodage utilisé par les receveurs. Une valeur supérieure à 0 pour N1m3 peut être un bon choix dans des situations spécifiques, par exemple, avec des codes LDPC-escalier quand l'envoyeur sait que tous les receveurs utilisent un schéma de décodage à élimination gaussienne. Néanmoins, le présent document ne rend obligatoire aucune valeur spécifique. Ce choix est laissé au développeur du codec.

G : entier entre 1 (par défaut) et 31, inclus, indiquant le nombre de symboles de codage par groupe (c'est-à-dire, par paquet). La valeur par défaut est 1, ce qui signifie que chaque paquet contient exactement un symbole. Des valeurs supérieures à 1 peuvent aussi être définies, comme expliqué au paragraphe 5.3.

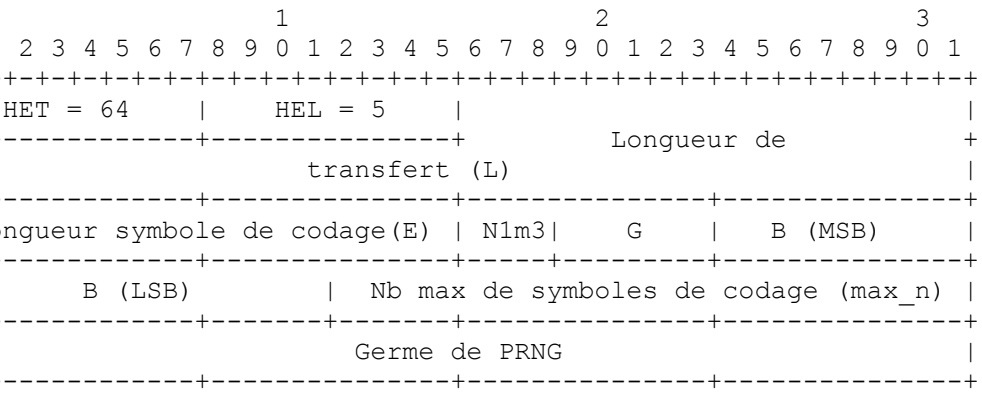
germe de PRNG : le germe est un entier non signé de 32 bits entre 1 et 0x7FFFFFFE (c'est-à-dire, 2^^31 -2) inclus. Cette valeur est utilisée pour initialiser le générateur de nombres pseudo aléatoires (paragraphe 5.7).

4.2.4 Format de codage

Ce paragraphe montre deux formats de codage possibles des FEC OTI ci-dessus. Le présent document ne spécifie pas quand ou comment ces formats de codage devraient être utilisés.

4.2.4.1 Utilisation du format général EXT_FTI

Le format binaire des FEC OTI est le suivant quand le mécanisme EXT_FTI est utilisé (par exemple, au sein des protocoles de codage de couche asynchrone (ALC, *Asynchronous Layer Coding*) [RFC5775] ou de diffusion groupée fiable en mode NACK (NORM, *NACK-Oriented Reliable Multicast*) [RFC5740]).



(HET = type d'en-tête d'extension ; HEL = Longueur d'en-tête d'extension)

Figure 2 : En-tête EXT_FTI pour les identifiants de codage de FEC 3 et 4

En particulier :

- La taille du champ Longueur de transfert (L) (48 bits) est supérieure à la taille requise pour mémoriser la longueur maximum de transfert (paragraphe 4.2.2) pour des raisons d'alignement de champ.

- Le champ Longueur maximum de bloc de source (B) (20 bits) est partagé en deux : les 8 bits de poids fort (MSB, *most significant bits*) sont dans le troisième mot de 32 bits des EXT_FTI, et les 12 bits restants de moindre poids (LSB, *least significant bits*) sont dans le quatrième mot de 32 bits.

4.2.4.2 Utilisation de l'instance FDT (spécifique de FLUTE)

Quand on désire que les FEC OTI soient portées dans l'instance de tableau de livraison de fichier (FDI, *File Delivery Table*) d'une session de livraison de fichier sur transport unidirectionnel (FLUTE, *File Delivery over Unidirectional Transport*) [RFC6726], les attributs XML suivants doivent être décrits pour l'objet associé :

- FEC-OTI-FEC-Encoding-ID *(identifiant de codage de FEC des FEC-OTI)*
- FEC-OTI-Transfer-length *(longueur de transfert des FEC-OTI)*
- FEC-OTI-Encoding-Symbol-Length *(longueur de symbole de codage des FEC-OTI)*
- FEC-OTI-Maximum-Source-Block-Length *(longueur maximum de bloc de source des FEC-OTI)*
- FEC-OTI-Max-Number-of-Encoding-Symbols *(nombre maximum de symboles de codage des FEC-OTI)*
- FEC-OTI-Scheme-Specific-Info *(informations spécifiques du schéma des FEC-OTI)*

Les informations spécifiques du schéma des FEC-OTI contiennent la chaîne résultant du codage Base64 [RFC4648] de la valeur suivante :

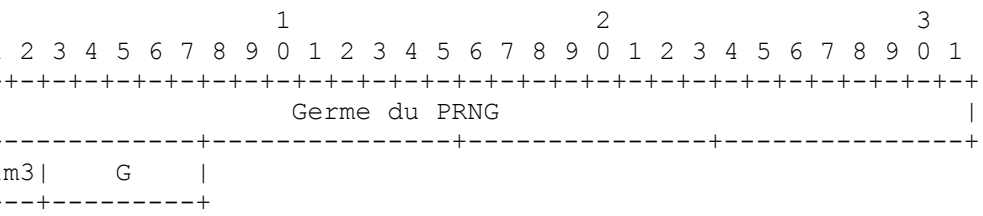


Figure 3 : Informations spécifiques de schéma FEC OTI à inclure dans l'instance de FDT pour les Id de codage de FEC 3 et 4

Durant le codage Base64, les cinq octets des informations spécifiques de schéma FEC OTI sont transformés en une chaîne de 8 caractères imprimables (dans l'alphabet de 64 caractères) ajouté à l'attribut FEC-OTI-Scheme-Specific-Info.

5. Procédures

Cette Section définit les procédures communes aux identifiants de codage de FEC 3 et 4.

5.1 Généralités

Le paramètres B (longueur maximum de bloc de source en symboles) E (longueur de symbole de codage en octets) et G (nombre de symboles de codage par groupe) sont d'abord déterminés. Les algorithmes des paragraphes 5.2 et 5.3 PEUVENT être utilisés à cette fin. L'utilisation d'autres algorithmes est possible sans compromettre l'interopérabilité car les paramètres B, E, et G sont communiqués au receveur au moyen des FEC OTI.

Ensuite l'objet de source DOIT être partagé en utilisant l'algorithme de partage de blocs spécifié dans la [RFC5052]. À cette fin, les arguments B, L (longueur de transfert d'objet en octets) et E sont fournis. Par suite, l'objet est partagé en N blocs de source. Ces blocs sont numérotés consécutivement de 0 à N-1. Les I premiers blocs de source consistent en A_large symboles de source, les N-I blocs de source restants consistent en A_small symboles de source. Chaque symbole de source est long de E octets, sauf peut-être le dernier symbole qui peut être plus court.

Ensuite, le paramètre max_n (nombre maximum de symboles de codage générés pour tout bloc de source) est déterminé. L'algorithme du paragraphe 5.4 PEUT être utilisé à cette fin. L'utilisation d'un autre algorithme est possible sans compromettre l'interopérabilité car le paramètre max_n est communiqué au receveur au moyen des FEC OTI.

Pour chaque bloc, le nombre réel de symboles de codage, n, DOIT alors être déterminé en utilisant le "n-algorithm" détaillé au paragraphe 5.5.

Ensuite, le codage et le décodage de FEC peuvent être faits indépendamment bloc par bloc. À cette fin, une matrice de vérification de parité est créée, qui forme un système d'équations linéaires entre les symboles de source et de réparation d'un bloc donné, où l'opérateur de base est OUX.

Cette matrice de vérification de parité est divisée logiquement en deux parties : le côté gauche (colonnes 0 à k-1) décrit les occurrences de chaque symbole de source dans le système d'équations linéaires ; le côté droit (colonnes k à n-1) décrit les occurrences de chaque symbole de réparation dans le système d'équations linéaires. La seule différence entre les schémas LDPC-Staircase et LDPC-Triangle est la construction de cette sous matrice de droite. Une entrée (un "1") dans la matrice à la position (i,j) (c'est-à-dire, à la rangée i et la colonne j) signifie que le symbole avec l'ESI j apparaît dans l'équation i du système.

Quand les symboles de parité ont été créés, l'envoyeur transmet les symboles de source et de parité. La façon dont cette transmission se produit peut largement impacter les capacités de récupération d'écrasement de la FEC LDPC-*. En particulier, l'envoi de symboles de parité à la suite est sous optimal. À la place, il est généralement recommandé de brassser ces symboles. Le lecteur intéressé trouvera plus de détails dans [NRFF05].

Les paragraphes qui suivent précisent comment les paramètre B, E, G, max_n, et n sont déterminés (paragraphes 5.2, 5.3, 5.4 et 5.5, respectivement). Le paragraphe 5.6 détaille comment sont créés les groupes de symboles de codage, et finalement, le paragraphe 5.7 traite du PRNG.

5.2 Détermination de la longueur maximum de bloc source (B)

Le paramètre B (longueur maximum de bloc de source en symboles) dépend de plusieurs paramètres : le taux de code (CR, *code rate*) la longueur du champ d'identifiant de symbole de codage de l'identifiant de charge utile de FEC (20 bits), ainsi que de possibles limitations internes du codec.

Le paramètre B ne peut pas être plus grand que les valeurs suivantes, dérivées des limitations de l'identifiant de charge utile de FEC, pour un taux de code donné : max1_B = 2^^(20 - sup(Log2(1/CR)))

Des valeurs courantes de max1_B sont :
o CR == 1 (pas de symbole de réparation) : max1_B = 2^^20 = 1 048 576

- o 1/2 ≤ CR < 1: max1_B = 2^^19 = 524 288 symboles
- o 1/4 ≤ CR < 1/2: max1_B = 2^^18 = 262 144 symboles
- o 1/8 ≤ CR < 1/4: max1_B = 2^^17 = 131 072 symboles

De plus, un codec PEUT imposer d'autres limitations à la taille maximum de bloc. Par exemple, c'est le cas quand le codec utilise en interne des entiers non signés de 16 bits pour mémoriser l'identifiant de symbole de codage, car cela ne permet pas de mémoriser toutes les valeurs possibles d'un champ de 20 bits. Dans ce cas, si par exemple, 1/2 ≤ CR < 1, alors la longueur maximum de bloc de source est 2^^15. D'autres limitations peuvent aussi s'appliquer, par exemple, à cause d'une taille limitée de mémoire de travail. Cette décision DOIT être précisée au moment de la mise en œuvre, quand le cas d'utilisation cible est connu. Il en résulte une limitation max2_B.

Ensuite, B est donné par : B = min(max1_B, max2_B)

Noter que ce calcul n'est exigé qu'au codeur, car le paramètre B est communiqué au décodeur par les FEC OTI.

5.3 Détermination de la longueur de symbole de codage (E) et du nombre de symboles de codage par groupe (G)

Le paramètre E dépend généralement de l'unité maximum de transmission sur le chemin (PMTU) de la source à chaque receveur. Afin de minimiser les frais généraux de l'en-tête de protocole (par exemple, des en-têtes de transport de codage en couches (LCT, *Layered Coding Transport*), UDP, IPv4, ou IPv6 dans le cas d'ALC) E est choisi aussi grand que possible. Dans ce cas, E est choisi de telle sorte que la taille de paquet composé d'un seul symbole (G=1) reste en dessous, mais proche de la PMTU.

Cependant, d'autres considérations peuvent exister. Par exemple, le paramètre E peut être une fonction de la longueur de transfert d'objet. Bien sûr, les codes LDPC sont connus pour offrir une meilleure protection pour les grands blocs. Dans le cas de petits objets, il peut être avantageux de réduire la longueur des symboles de codage (E) afin d'augmenter artificiellement le nombre de symboles et donc la taille de bloc.

Afin de minimiser les frais généraux d'en-tête de protocole, plusieurs symboles peuvent être groupés dans le même groupe de symboles de codage (c'est-à-dire, G > 1). Selon le nombre de symboles groupés (G) et le taux de perte de paquets (G symboles sont perdus pour chaque écrasement de paquet) cette stratégie pourrait ou non être appropriée. Un équilibre doit donc être trouvé.

La spécification actuelle n'impose aucune valeur pour E ou G. Elle fournit seulement un exemple des choix possibles pour E et G. Noter que ce choix est fait par l'envoyeur, et les paramètres E et G sont alors communiqués au receveur grâce aux FEC OTI. Noter aussi que l'algorithme de décodage utilisé influence le choix des paramètres E et G. Bien sûr, augmenter le nombre de symboles va impacter négativement la charge de traitement quand le décodage se fonde (totalement ou en partie) sur l'élimination gaussienne, tandis que l'impact sera assez faible quand le décodage se fonde sur l'algorithme trivial montré au paragraphe 6.4.

Exemple : supposons que l'algorithme trivial de décodage montré au paragraphe 6.4 soit utilisé. D'abord, définir la taille de la charge utile de paquet cible, pkt_sz (au plus égal à la PMTU moins la taille des divers en-têtes de protocole). La pkt_sz doit être choisie de telle façon que la taille de symbole soit un entier. Cela peut exiger que pkt_sz soit un multiple de 4, 8, ou 16 (voir le tableau ci-dessous). Ensuite on calcule le nombre de paquets dans l'objet : nb_pkts = sup(L / pkt_sz). Finalement, grâce à nb_pkts, on utilise le tableau suivant pour trouver la valeur possible de G.

Nombre de paquets	**G**	**Taille de symbole**	**k**
4000 ≤ nb_pkts	1	pkt_sz	4000 ≤ k
1000 ≤ nb_pkts < 4000	4	pkt_sz / 4	4000 ≤ k < 16000
500 ≤ nb_pkts < 1000	8	pkt_sz / 8	4000 ≤ k < 8000
1 ≤ nb_pkts < 500	16	pkt_sz / 16	16 ≤ k < 8000

5.4. Détermination du nombre maximum de symboles de codage générés par un bloc de toute source (max_n)

L'algorithme suivant PEUT être utilisé par un envoyeur pour déterminer le nombre maximum de symboles de codage générés pour tout bloc de source (max_n) comme fonction de B et du taux de code cible. Comme le paramètre max_n est communiqué au décodeur au moyen des FEC OTI, une autre méthode PEUT être utilisée pour déterminer max_n.

Entrées :

B : longueur maximum de bloc de source, pour tout bloc de source. Le paragraphe 5.2 PEUT être utilisé pour déterminer sa valeur.

CR : taux de code de FEC, qui est fourni par l'utilisateur (par exemple, quand il commence une application d'envoi FLUTE). Il est exprimé comme une valeur à virgule flottante. La valeur de CR doit être telle que le nombre de symboles de codage par bloc résultant soit au plus égal à 2^^20 (paragraphe 4.1).

Résultat : max_n, nombre maximum de symboles de codage généré pour tout bloc de source.

Algorithme :
max_n = sup(B / CR) ;
si (max_n > 2^^20), retourner une erreur ("taux de code invalide");

(Note : si B a été défini comme expliqué au paragraphe 5.2, cette erreur ne devrait jamais arriver.)

5.5 Détermination du nombre de symboles de codage par bloc (n)

L'algorithme suivant, aussi appelé "algorithme n", DOIT être utilisé par l'envoyeur et le receveur pour déterminer le nombre de symboles de codage par bloc (n) donné comme fonction de B, k, et max_n.

Entrées :
B : longueur maximum de bloc de source, pour tout bloc de source. Chez l'envoyeur, le paragraphe 5.2 PEUT être utilisé pour déterminer sa valeur. Chez un receveur, cette valeur DOIT être extraite des FEC OTI reçues.
k : longueur acuelle du bloc de source. Chez un envoyeur ou receveur, l'algorithme de partage de bloc DOIT être utilisé pour déterminer sa valeur.
max_n : nombre maximum de symboles de codage générés pour tout bloc de source. Chez un envoyeur, le paragraphe 5.4 PEUT être utilisé pour déterminer sa valeur. Chez un receveur, cette valeur DOIT être extraite des FEC OTI reçues.

Résultat :
n : nombre de symboles de codage générés pour ce bloc de source.

Algorithme : n = inf(k * max_n / B) ;

5.6 Identification des G symboles d'un groupe de symboles de codage

Quand plusieurs symboles de codage sont envoyés dans le même paquet, les informations d'identifiant de charge utile de FEC du paquet DOIVENT se référer au premier symbole de codage. Il DOIT alors être possible d'identifier chaque symbole à partir de ce seul identifiant de charge utile de FEC. À cette fin, les symboles d'un groupe de symboles de codage (c'est-à-dire, le paquet) :

- DOIVENT tous être soit des symboles de source, soit des symboles de réparation. Donc, seuls les paquets de source et les paquets de réparation sont permis, pas les paquets mixtes.
- sont identifiés par une fonction, envoyeur(respectivement, receveur)_find_ESI_de_group(), qui prend pour argument :
  * pour un envoyeur, l'indice du groupe de symboles de codage (c'est-à-dire, le paquet) que l'application veut créer,
  * pour un receveur, les informations d'ESI contenues dans l'identifiant de charge utile de FEC,
  et retourne une liste de G identifiants de symbole de codage. Dans le cas d'un paquet de source, les G identifiants de symboles de codage sont choisis consécutivement en incrémentant l'ESI. Dans le cas d'un paquet de réparation, les G symboles de réparation sont choisis au hasard, comme expliqué ci-dessous.
- sont mémorisés en séquence dans le paquet, sans bourrage. En d'autres termes, le dernier octet du ième symbole est immédiatement suivi par le premier octet du (i+1)ème symbole.

Le système doit d'abord être initialisé en créant une permutation aléatoire des n-k indices. Cette fonction d'initialisation DOIT être invoquée immédiatement après la création de la matrice de vérification de parité. Plus précisément, comme le germe de PRNG n'est pas réinitialisé, il ne doit pas y avoir d'invocation de la fonction de PRNG entre le moment où la matrice de vérification de parité a été initialisée et le moment où la fonction d'initialisation suivante est invoquée. Ceci est vrai chez l'envoyeur et chez un receveur.

```
int *txseqToID;
int *IDtoTxseq;

/*
* Fonction d'initialisation.
* Attention : utiliser seulement quand G > 1.
*/
  void
```

```
  initialiser_tableaux ()
  {
     int i;
     int randInd;
     int backup;

     txseqToID = malloc((n-k) * sizeof(int));
     IDtoTxseq = malloc((n-k) * sizeof(int));
     si (txseqToID == NULL || IDtoTxseq == NULL)
        traiter les défaillances de malloc comme approprié...
/* initialise les deux tableaux qui transposent l'ID (c'est-à-dire, ESI-k) de/en TxSequence. */
     pour (i = 0; i < n - k; i++) {
        IDtoTxseq[i] = i;
        txseqToID[i] = i;
     }
/* on rend maintenant tout aléatoire */
     pour (i = 0; i < n - k; i++) {
        randInd = pmms_rand(n - k);
        backup  = IDtoTxseq[i];
        IDtoTxseq[i] = IDtoTxseq[randInd];
        IDtoTxseq[randInd] = backup;
        txseqToID[IDtoTxseq[i]] =  i;
        txseqToID[IDtoTxseq[randInd]] = randInd;
     }
     retour ;
  }
```

Il est alors possible, chez l'envoyeur, de déterminer la séquence des G identifiants de symboles de codage qui vont faire partie du groupe.

```
/*
* Déterminer la séquence des ESI pour le paquet en construction chez envoyeur.0
* Attention : utiliser seulement quand G > 1.
* PktIdx (IN) : indice du paquet, dans {0..sup(k/G)+sup((n-k)/G)} gamme des ESI[] (OUT) : liste des ESI pour le paquet */
void
envoyeur_trouve_ESI_du_groupe (int PktIdx, ESI_t ESIs[])
{
   int i;
  si (PktIdx < nbSourcePkts) {                    /* c'est un paquet de source */
      ESIs[0] = PktIdx * G;
      pour (i = 1; i < G; i++) {
            ESIs[i] = (ESIs[0] + i) % k;
      }
   } autrement {                                  /* c'est un paquet de réparation */
      pour (i = 0; i < G; i++) {
         ESIs[i] = k + txseqToID[(i + (PktIdx - nbSourcePkts) * G) % (n - k)];
      }
   }
   retour ;
}
```

De même, à réception d'un groupe de symboles de codage (c'est-à-dire, un paquet) un receveur peut déterminer la séquence des G identifiants de symboles de codage à partir du premier ESI, esi0, qui est contenu dans l'identifiant de charge utile de FEC.

```
/*
* Déterminer la séquence des ESI pour le paquet reçu.
* Attention : utiliser seulement quand G > 1.
* esi0 (IN) : ESI contenu dans l'identifiant de charge utile de FEC
* ESIs[] (OUT) : liste des ESI pour le paquet. */
void
receveur_trouve_ESI_du_groupe (ESI_t    esi0, ESI_t    ESIs[])
```

```
{
   int i;
  si (esi0 < k) {                                  /* c'est un paquet de source */
      ESIs[0] = esi0;
      pour (i = 1; i < G; i++) {
         ESIs[i] = (esi0 + i) % k;
      }
   } autrement {                                   /* c'est un paquet de réparation */
      pour (i = 0; i < G; i++) {
         ESIs[i] =  k + txseqToID[(i + IDtoTxseq[esi0 - k]) % (n - k)];
      }
   }
}
```

5.7 Générateur de nombre pseudo aléatoire

Les identifiants de codage de FEC 3 et 4 s'appuient sur un générateur de nombres pseudo-aléatoires (PRNG) qui doit être pleinement spécifié, en particulier afin de permettre aux receveurs et aux envoyeurs de construire la même matrice de vérification de parité.

Le PRNG Park-Miler "minimal standard" [PM88] DOIT être utilisé. Il définit un simple algorithme multiplicatif congruent : Ij+1 = A * Ij (modulo M) avec les choix suivants : A = 7^^5 = 16 807 et M = 2^^31 - 1 = 2 147 483 647. Un critère de validation de ce PRNG est le suivant : si germe = 1, alors la 10 000 ième valeur retournée DOIT être égale à 1 043 618 065.

Plusieurs mises en œuvre de ce PRNG sont connues et discutées dans la littérature. Une mise en œuvre optimisée de cet algorithme, utilisant seulement une mathématique de 32 bits, et qui n'exige aucune division, peut être trouvée dans [rand31pmc]. Elle utilise l'algorithme de Park et Miller [PM88] avec l'optimisation suggérée par D. Carta dans [CA90]. L'historique de cet algorithme est détaillé dans [WI08]. Cependant, toute autre mise en œuvre de l'algorithme de PRNG qui satisfait les critères de validation ci-dessus, comme celles détaillées dans [PM88], est appropriée.

Ce PRNG produit, naturellement, une valeur de 31 bits entre 1 et 0x7FFFFFFE (2^^31-2) inclus. Comme on désire adapter le nombre pseudo-aléatoire entre 0 et maxv-1 inclus, on doit garder les bits de poids fort de la valeur retournée par le PRNG (il est connu que les bits de moindre poids sont moins aléatoires, et les solutions à base modulaire devraient être évitées [PTVF92]). L'algorithme suivant DOIT être utilisé :

Entrée :
raw_valeur : entier aléatoire généré par l'algorithme interne de PRNG entre 1 et 0x7FFFFFFE (2^^31-2) inclus.
maxv : limite supérieure utilisée durant l'opération d'adaptation.

Résultat :
scaled_valeur : entier aléatoire entre 0 et maxv-1 inclus.

Algorithme :
scaled_valeur = (non signé long) ((double)maxv * (double)raw_valeur / (double)0x7FFFFFFF);

(Note : l'invocation de type C ci-dessus à "non signé long" est équivalente à utiliser inf() avec des valeurs positives à virgule flottante.)

Dans le présent document, pmms_rand(maxv) note la fonction de PRNG qui met en œuvre l'algorithme "minimal standard" de Park-Miller, défini ci-dessus, et qui adapte la valeur brute entre 0 et maxv-1 inclus, en utilisant l'algorithme d'adaptation ci-dessus. De plus, une fonction devrait être fournie pour permettre l'initialisation du PRNG avec un germe (c'est-à-dire, un entier de 31 bits entre 1 et 0x7FFFFFFE inclus) avant la première invocation de pmms_rand(maxv).

6. Spécification complète du schéma LDPC-Staircase

6.1 Généralités

Le schéma LDPC-Staircase est identifié par l'identifiant de codage de FEC pleinement spécifié 3.

Le PRNG utilisé par le schéma LDPC-Staircase doit être initialisé par un germe. Ce germe de PRNG est un attribut de FEC OTI spécific de l'instance (paragraphe 4.2.3).

6.2 Création de matrice de vérification de parité

La matrice LDPC-Staircase peut être divisée en deux parties : le côté gauche de la matrice définit dans quelles équations les symboles de source sont impliqués ; le côté droit de la matrice définit dans quelles équations les symboles de réparation sont impliqués.

Le côté gauche DOIT être généré en utilisant la fonction suivante :

```
/*
* Initialiser le côté gauche de la matrice de vérification de parité. Cette fonction suppose qu'une matrice vide de taille n-k *
    k a été précédemment allouée/réinitialisée et que les fonctions matrix_has_entry(), matrix_insert_entry() et
    degree_of_row() peuvent y accéder.
* (Entrées) : les paramètres k, n et N1. */

void left_matrix_init (int k, int n, int N1)
{
   int i;                                   /* indice de rangée ou variable temporaire */
   int j;                                   /* indice de colonne */
   int h;                                   /* variable temporaire  */
   int t;                                   /* limite de gauche dans la liste des choix possibles u[] */
   int u[N1*MAX_K];                         /* tableau utilisé pour avoir une distribution homogène de 1. */

/* Initialise une liste de tous les choix possibles afin de garantir une distribution homogène de "1". */

   pour (h = N1*k-1; h ≥ 0; h--) {
      u[h] = h % (n-k);
 }

/* Initialise la matrice avec N1 "1" par colonne, de façon homogène. */
 t = 0;
 pour (j = 0; j < k; j++) {                 /* pour chaque colonne de symboles de source. */
    pour (h = 0; h < N1; h++) {             /* ajoute N1 "1". */
/* vérifier qu'il reste des choix valides disponibles. */
       pour (i = t; i < N1*k && matrix_has_entry(u[i], j); i++);
       si (i < N1*k) {                      /* choisir un indice dans la liste des choix possibles. */
          faire {
             i = t + pmms_rand(N1*k-t);
          } quand (matrix_has_entry(u[i], j));
          matrix_insert_entry(u[i], j);     /* remplacer par u[t] qui n'a jamais été choisi. */
          u[i] = u[t];
          t++;
       } autrement {                        /* aucun choix restant, en choisir un au hasard. */
          faire {
             i = pmms_rand(n-k);
          } quand (matrix_has_entry(i, j));
          matrix_insert_entry(i, j);
       }
    }
 }

/* Ajoute des bits supplémentaires pour éviter des rangées avec moins de deux "1". C'est nécessaire quand le taux de code
    est inférieur à 2/(2+N1) */
   pour (i = 0; i < n-k; i++) {             /* pour chaque rangée */
     si (degree_of_row(i) == 0) {
        j = pmms_rand(k);
        matrix_insert_entry(i, j);
      }
     si (degree_of_row(i) == 1) {
```

```
        faire{
           j = pmms_rand(k);
        } tandis que si (matrix_has_entry(i, j));
        matrix_insert_entry(i, j);
     }
  }
}
```

Le côté droit (l'escalier) DOIT être généré en utilisant la fonction suivante :

```
/* * Initialise le côté droit de la matrice de vérification de parité avec une structure d'escalier.
   * (Entrée) : les paramètres k et n.  */
void right_matrix_staircase_init (int k, int n)
{
   int i;                                   /* indice de rangée */
   matrix_insert_entry(0, k);               /* première rangée */
   pour (i = 1; i < n-k; i++) {             /* pour les rangées suivantes */
      matrix_insert_entry(i, k+i);          /* identité */
      matrix_insert_entry(i, k+i-1);        /* escalier */
   }
}
```

Noter que juste après la création de cette matrice de vérification de parité, quand les groupes de symboles de codage sont utilisés (c'est-à-dire, G > 1) la fonction qui initialise les deux tableaux de permutation aléatoires (paragraphe 5.6) DOIT être invoquée. Ceci est vrai chez un envoyeur et chez un receveur.

6.3 Codage

Grâce à la matrice d'escalier, la création de symboles de réparation est directe : chaque symbole de réparation est égal à la somme de tous les symboles de source dans l'équation associée, plus les précédents symboles de réparation (excepté le premier symbole de réparation). Donc, le codage DOIT suivre l'ordre naturel des symboles de réparation : commencer avec le premier symbole de réparation et générer un symbole de réparation avec l'ESI i avant un symbole avec ESI i+1.

6.4 Décodage

Le décodage consiste principalement à résoudre un système de n-k équations linéaires dont les variables sont les n symboles de source et de réparation. Bien sûr, le but final est de récupérer la valeur de seulement k symboles de source.

À cette fin, de nombreuses techniques sont possibles. Une d'elles est l'algorithme trivial suivant [ZP74] : étant donné un ensemble d'équations linéaires, si une d'elles a seulement une variable restante inconnue, alors la valeur de cette variable est celle du terme constant. Donc, on remplace cette variable par sa valeur dans toutes les équations linéaires restantes et on recommence. La valeur de plusieurs variables peut donc être trouvée de façon récurrente. Appliquée aux codes de FEC LDPC travaillant sur un canal d'écrasement, la matrice de vérification de parité définit un ensemble d'équations linéaires dont les variables sont les symboles de source et les symboles de réparation. Recevoir ou décoder un symbole est équivalent à avoir la valeur d'une variable. L'Appendice A représente une mise en œuvre possible de cet algorithme.

Une élimination gaussienne (ou toute variante optimisée) est une autre technique de décodage possible. Des solutions hybrides qui commencent par utiliser l'algorithme trivial ci-dessus et finissent par une élimination gaussienne sont aussi possibles [CR08].

Parce que l'interopérabilité ne dépend pas de l'algorithme de décodage utilisé, le présent document ne recommande aucune technique particulière. Ce choix est laissé au développeur de codec.

Cependant, choisir une technique de décodage va avoir de grands impacts pratiques. Cela va impacter les capacités d'écrasement : une élimination gaussienne permet de résoudre le système avec un plus petit nombre de symboles connus par rapport à la technique triviale. Cela va aussi impacter la charge de CPU : une élimination gaussienne exige plus de traitement que l'algorithme trivial ci-dessus. Selon le cas d'utilisation ciblé, le développeur de codec va préférer une caractéristique à d'autres.

7. Spécification complète du schéma LDPC-Triangle

7.1 Généraités

LDPC-Triangle est identifié par l'identifiant de codage de FEC pleinement spécifié 4.

Le PRNG utilisé par le schéma de LDPC-Triangle doit être initialisé par un germe. Ce germe de PRNG est un attribut de FEC OTI spécifique de l'instance (paragaphe 4.2.3).

7.2 Création de matrice de vérification de parité

La matrice LDPC-Triangle peut être divisée en deux parties : le côté gauche de la matrice définit dans quelles équations les symboles de source sont impliqués ; le côté droit de la matrice définit dans quelles équations les symboles de réparation sont impliqués.

Le côté gauche DOIT être généré en utilisant la même fonction left_matrix_init() que dans LDPC-Staircase (paragraphe 6.2).

Le côté droit (le triangle) DOIT être généré en utilisant la fonction suivante :

```
/* * Initialiser le côté droit de la matrice de vérification de parité avec une structure de triangle.
   * (Entrée) : paramètres k et n. */
void right_matrix_staircase_init (int k, int n)
{
   int i;                                   /* indice de rangée */
   int j;                                   /* indices de colonnes choisis au hasard dans 0..n-k-2 */
   int l;                                   /* limitation du nombre de "1" ajoutés par rangée */
  matrix_insert_entry(0, k);                /* première rangée */
   pour (i = 1; i < n-k; i++) {             /* pour les rangées suivantes */
      matrix_insert_entry(i, k+i);          /* identité */
      matrix_insert_entry(i, k+i-1);        /* escalier */
/* on remplit maintenant le triangle */
      j = i-1;
      pour (l = 0; l < j; l++) {            /* limite le nombre de "1" ajoutés */
         j = pmms_rand(j);
         matrix_insert_entry(i, k+j);
      }
   }
}
```

Noter que juste après la création de cette matrice de vérification de parité, quand les groupes de symboles de codage sont utilisés (c'est-à-dire, G > 1) la fonction qui initialise les deux tableaux de permutation aléatoires (paragraphe 5.6) DOIT être invoquée. Ceci est vrai chez un envoyeur et chez un receveur.

7.3 Codage

Ici aussi, la création de symboles de réparation est directe : chaque symbole de réparation de ESI i est égal à la somme de tous les symboles de source et de réparation (avec ESI inférieur à i) dans l'équation associée. Donc, le codage DOIT suivre l'ordre naturel des symboles de réparation : commencer par le premier symbole de réparation, et générer le symbole de réparation avec ESI i avant le symbole avec ESI i+1.

7.4 Décodage

Le décodage consiste principalement à résoudre un système de n-k équations linéaires, dont les variables sont les n symboles de source et de réparation. Bien sûr, le but final est de récupérer seulement la valeur des k symboles de source. À cette fin, de nombreuses techniques sont possibles, comme expliqué au paragraphe 6.4.

Parce que l'interopérabilité ne dépend pas de l'algorithme de décodage utilisé, le présent document ne recommande aucune technique particulière. Ce choix est laissé aux mises en œuvre de codec.

8. Considérations sur la sécurité

8.1 Position du problème

Un système de livraison de contenu est potentiellement soumis à de nombreuses attaques : certaines visant le réseau (par exemple, pour compromettre l'infrastructure d'acheminement, en compromettant le composant de contrôle d'encombrement) d'autres visant le protocole de livraison de contenu (CDP, *Content Delivery Protocol*) (par exemple, pour compromettre son comportement normal) et finalement certaines attaques visent le contenu lui-même. Comme le présent document se concentre sur le bloc de construction de FEC indépendamment de tout CDP particulier (même si ALC et NORM sont deux candidats naturels) cette Section discute seulement les menaces supplémentaires auxquelles un CDP arbitraire peut être exposé quand il utilise ces blocs de construction.

Plus précisément, il existe plusieurs sortes d'attaques :
- o celles qui sont destinées à donner l'accès à un contenu confidentiel (par exemple, dans le cas d'un contenu non libre) ;
- o celles qui essayent de corrompre l'objet à transmettre (par exemple, pour injecter du code malveillant dans un objet, ou pour empêcher un receveur d'utiliser un objet), et
- o celles qui essayent de compromettre le comportement du receveur (par exemple, en rendant le décodage d'un objet coûteux en calcul).

Ces attaques peuvent être lancées soit contre le flux de données lui-même (par exemple, en envoyant des symboles falsifiés) ou contre les paramètres de FEC qui sont envoyés dans la bande (par exemple, dans une instance EXT_FTI ou FDT) ou hors bande (par exemple, dans une description de session).

8.2 Attaques contre le flux de données

On examine d'abord les attaques contre le flux de données.

8.2.1 Accès à des objets confidentiels

Le contrôle d'accès à un objet confidentiel transmis est normalement fourni au moyen du chiffrement. Ce chiffrement peut être fait sur l'objet entier (par exemple, par le fournisseur de contenu, avant le processus de codage de FEC) ou paquet par paquet (par exemple, quand IPsec/ESP est utilisé [RFC4303]). Si la confidentialité est un problème, il est RECOMMANDÉ qu'une de ces solutions soit utilisée. Même si on mentionne ici ces attaques, elles ne sont pas en relation ni facilitées par l'utilisation de la FEC.

8.2.2 Corruption du contenu

La protection contre les corruptions (par exemple, après l'envoi de paquet falsifiés) est réalisée au moyen d'un schéma de vérification de l'intégrité du contenu/authentification de l'envoyeur. Ce service peut être fourni au niveau de l'objet, mais dans ce cas, un receveur n'a pas de moyen d'identifier quels symboles sont corrompus si l'objet est détecté comme corrompu. Ce service peut aussi être fourni au niveau du paquet. Dans ce cas, après avoir retiré tous les paquets falsifiés, l'objet peut être récupéré, dans certains cas. Plusieurs techniques peuvent fournir ce service d'authentification de la source/intégrité du contenu :

- o au niveau de l'objet, l'objet PEUT être signé numériquement (avec la cryptographie à clé publique) par exemple, en utilisant RSASSA-PKCS1-v1_5 [RFC3447]. Cette signature permet à un receveur de vérifier l'intégrité de l'objet, une fois qu'il a été complètement décodé. Même si les signatures numériques sont coûteuses en calcul, ce calcul intervient seulement une fois par objet, ce qui est généralement acceptable;

- o au niveau du paquet, chaque paquet peut être signé numériquement. Une limitation majeure est les frais généraux de calcul et de transmission élevés que cette solution exige (sauf peut-être si la cryptographie à courbe elliptique (ECC, *Elliptic Curve Cryptography*) est utilisée). Pour éviter ce problème, la signature peut s'étendre sur un ensemble de symboles (au lieu d'un seul) afin d'amortir le calcul de signature. Mais si un seul symbole manque, l'intégrité de l'ensemble ne peut pas être vérifiée;

- o au niveau du paquet, un schéma de code d'authentification de message (MAC, *Message Authentication Code*) de groupe [RFC2104] peut être utilisé, par exemple, en utilisant HMAC-SHA-1 avec une clé secrète partagée par tous les membres du groupe, envoyeurs, et receveurs. Cette technique crée un résumé cryptographiquement sécurisé (grâce à la clé secrète) d'un paquet qui est envoyé avec le paquet. Le schéma de MAC de groupe ne crée pas une charge de traitement ou frais généraux de transmission prohibitifs, mais a une limitation majeure : il fournit seulement un service d'authentification/intégrité de groupe car tous les membres du groupe partagent la même clé secrète de groupe, ce qui

signifie que chaque membre peut envoyer un paquet falsifié. Il est donc restreint aux situations où les membres du groupe sont pleinement de confiance (ou en association avec une autre technique comme une pré vérification) ;

- o au niveau du paquet, l'authentification tolérante aux pertes de flux à synchronisation efficace (TESLA, *Timed Efficient Stream Loss-Tolerant Authentication*) [RFC4082] est une solution intéressante qui est robuste aux pertes, fournit un vrai service d'authentification/intégrité, et ne crée pas de charge de traitement ou de frais généraux de transmission prohibitifs. Cependant, vérifier un paquet exige un petit délai (une seconde ou plus) après sa réception.

Les techniques qui s'appuient sur la cryptographie à clé publique (signatures numériques et TESLA durant le processus d'amorçage, quand il est utilisé) exigent que les clés publiques soient associées de façon sûre aux entités. Cela peut être réalisé par une infrastructure de clé publique (PKI, *Public Key Infrastructure*) ou par une toile de confiance PGP, ou en pré distribuant les clés publiques de chaque membre du groupe.

Les techniques qui s'appuient sur la cryptographie à clés symétriques (MAC de groupe) exigent qu'une clé secrète soit partagée par tous les membres du groupe. Cela peut être réalisé au moyen d'un protocole de gestion de clé de groupe, ou simplement en pré distribuant la clé secrète (mais cette solution manuelle a de nombreuses limitations).

Il appartient au développeur de CDP, qui connaît les exigences et caractéristiques de sécurité de la zone d''application cible, de définir quelle solution est la plus appropriée. Néanmoins, au cas où on se soucie de la menace de corruption de l'objet, il est RECOMMANDÉ qu'au moins une de ces techniques soit utilisée.

8.3 Attaques contre les paramètres de FEC

On examine maintenant les attaques contre les paramètres de FEC (ou les FEC OTI). Les FEC OTI peuvent être envoyées dans la bande (c'est-à-dire, dans une EXT_FTI ou dans une instance de FDT contenant les FEC OTI pour l'objet) ou hors bande (par exemple, dans une description de session). Les attaques contre ces paramètres de FEC peuvent empêcher le décodage de l'objet associé : par exemple, modifier le paramètre B va conduire à un partage de bloc différent.

Il est donc RECOMMANDÉ que des mesures de sécurité soient prises pour garantir l'intégrité des FEC OTI. À cette fin, les paquets portant les paramètres de FEC envoyés dans la bande dans une extension d'en-tête EXT_FTI DEVRAIENT être protégés par une des techniques par paquet décrites ci-dessus : signature numérique, MAC de groupe, ou TESLA. Quand les FEC OTI sont contenues dans une instance de FDT, cet objet DEVRAIT être protégé, par exemple, en le signant numériquement avec des signatures numériques XML [RFC3275]. Finalement, quand les FEC OTI sont envoyées hors bande (par exemple, dans une description de session) elles DEVRAIENT être protégées, par exemple, en les signant numériquement avec la [RFC3852].

Les mêmes considérations concernant les aspects de gestion de clé s'appliquent aussi ici.

9. Considérations relatives à l'IANA

Les valeurs d'identifiants de codage de FEC et d'identifiants d'instance de FEC sont l'objet d'un enregistrement par l'IANA. Pour les lignes directrices générales sur les considérations relatives à l'IANA telles qu'elles s'appliquent au présent document, voir la [RFC5052].

Le présent document alloue l'identifiant 3 de codage de FEC pleinement spécifié sous l'espace de noms "ietf:rmt:fec:encoding" à "LDPC Staircase Codes".

Le présent document alloue l'identifiant 4 de codage de FEC pleinement spécifié sous l'espace de noms "ietf:rmt:fec:encoding" à "LDPC Triangle Codes".

10. Remerciements

Le paragraphe 5.5 est dérivé d'un document antérieur et nous tenons à remercier S. Peltotalo et J. Peltotalo de leur contribution. Nous tenons aussi à remercier Pascal Moniot, Laurent Fazio, Mathieu Cunche, Aurelien Francillon, Shao Wenjian, Magnus Westerlund, Brian Carpenter, Tim Polk, Jari Arkko, Chris Newman, Robin Whittle, et Alfred Hoenes de leurs commentaires.

Enfin et non le moindre, les auteurs expriment leur reconnaissance à Radford M. Neal (Université de Toronto) dont le logiciel LDPC (http://www.cs.toronto.edu/~radford/ldpc.software.html) a inspiré le présent travail.

11. Références

11.1 Références normatives

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

[RFC5052] M. Watson et autres, "Bloc de construction de la correction d'erreur directe (FEC)", août 2007. (*Remplace* RFC3452) *(P.S.)*

11.2 Références pour information

[CA90] Carta, D., "Two Fast Implementations of the Minimal Standard Random Number Generator", Communications of the ACM, Vol. 33, No. 1, pp.87-88, janvier 1990.

[CR08] Cunche, M. and V. Roca, "Improving the Decoding of LDPC Codes for the Packet Erasure Channel with a Hybrid Zyablov Iterative Decoding/Gaussian Elimination Scheme", rapport de recherche INRIA RR-6473, mars 2008.

[LDPC-codec] Roca, V., Neumann, C., Cunche, M., et J. Laboure, "LDPC-Staircase/LDPC-Triangle Codec Reference Implementation", INRIA Rhone-Alpes et STMicroelectronics, <http://planete-bcast.inrialpes.fr/>.

[MK03] MacKay, D., "Information Theory, Inference and Learning Algorithms", Cambridge University Press, ISBN: 0-521-64298-1, 2003.

[NRFF05] Neumann, C., Roca, V., Francillon, A., et D. Furodet, "Impacts of Packet Scheduling and Packet Loss Distribution on FEC Performances: Observations and Recommendations", ACM CoNEXT'05 Conference, Toulouse, France (une version étendue est disponible comme rapport de recherche INRIA RR-5578), octobre 2005.

[PM88] Park, S. et K. Miller, "Random Number Generators: Good Ones are Hard to Find", Communications of the ACM, Vol. 31, No. 10, pp.1192-1201, 1988.

[PTVF92] Press, W., Teukolsky, S., Vetterling, W., and B. Flannery, "Numerical Recipes in C; Second Edition", Cambridge University Press, ISBN: 0-521-43108-5, 1992.

[rand31pmc] Whittle, R., "31 bit pseudo-random number generator", septembre 2005, <http://www.firstpr.com.au/dsp/rand31/rand31-park-miller-carta.cc.txt>.

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997.

[RFC3275] D. Eastlake 3rd , J. Reagle, D. Solo, "Syntaxe et traitement de signature en langage de balisage extensible (XML)", mars 2002. *(D.S.)*

[RFC3447] J. Jonsson et B. Kaliski, "Normes de cryptographie à clés publiques (PKCS) n° 1 : Spécifications de la cryptographie RSA version 2.1", février 2003. *(Obsolète, remplacée par* RFC8017) *(Information)*

[RFC3453] M. Luby et autres, "Utilisation de la correction d'erreur directe (FEC) en diffusion groupée fiable", décembre 2002. *(Info.)*

[RFC3852] R. Housley, "Syntaxe de message cryptographique (CMS)", juillet 2004. *(Obsolète, voir la* RFC5652)

[RFC4082] A. Perrig et autres, "Authentification de flux tolérante aux pertes en temps efficace (TESLA) : Introduction à la transformation d'authentification de source de diffusion groupée", juin 2005. *(Information)*

[RFC4303] S. Kent, "Encapsulation de charge utile de sécurité dans IP (ESP)", décembre 2005. (*Remplace* RFC2406) *(P.S.)*

[RFC4648] S. Josefsson, "Codages de données Base16, Base32 et Base64", octobre 2006. (*Remplace* RFC3548) *(P.S.)*

[RFC5740] B. Adamson, C. Bormann, M. Handley, J. Macker, "Protocole de transport de diffusion groupée fiable orientée NACK (NORM)", novembre 2009. *(Remplace RFC3940.* (P.S.)

[RFC5775] M. Luby, M. Watson, L. Vicisano, "Instance de protocole de codage en couches asynchrone (ALC)", avril 2010. (*Remplace* RFC3450). *(P.S.)*

[RFC6726] T. Paila, et autres, "FLUTE – Livraison de fichier sur transport unidirectionnel", novembre 2012. (*Remplace la* RFC3926) *(P.S.)*

[RN04] Roca, V. and C. Neumann, "Design, Evaluation and Comparison of Four Large Block FEC Codecs: LDPC, LDGM, LDGM-Staircase and LDGM-Triangle, plus a Reed-Solomon Small Block FEC Codec", rapport de recherche INRIA RR-5225, juin 2004.

[WI08] Whittle, R., "Park-Miller-Carta Pseudo-Random Number Generator", janvier 2008, <http://www.firstpr.com.au/dsp/rand31/>.

[ZP74] Zyablov, V. and M. Pinsker, "Decoding Complexity of Low-Density Codes for Transmission in a Channel with Erasures", Traduit de "Problemy Peredachi Informatsii", Vol.10, No. 1, pp.15-28, janvier-mars 1974.

Appendice A. Algorithme trivial de décodage (pour information)

Un algorithme de décodage trivial est présenté ci-dessous (voir dans [LDPC-codec] les détails omis ici) :

Initialisation : allouer un tableau partial_sum[n-k] de mémoires tampon, chacune de la taille de symbole. Il y a une entrée par équation car les mémoires tampon sont destinées à mémoriser la somme partielle de chaque équation ; remettre toutes les mémoires tampon à zéro;

```
/* * Pour chaque nouveau symbole reçu ou décodé, essayer de faire progresser le décodage du bloc de source associé.
   * Note : dans le cas d'un groupe de symboles (G>1), cette fonction est invoquée pour chaque symbole du paquet reçu.
   * Note : une fonction de rappel indique à l'appelant qu'un ou des nouveaus symboles ont été décodés.
   * new_esi  (Entrée) : ESI du nouveau symbole reçu ou décodé.
   * new_symb (Entrée) : Mise en mémoire tampon du nouveau symbole reçu ou décodé.  */
void
 décodage_step(ESI_t   new_esi, symbol_t  *new_symb)
   {
   Si (new_symb est un symbole déjà décodé ou reçu ) {
        Retourner ;                                  /* ne pas perdre de temps avec ce symbole */
   }
   Si (new_symb est le dernier symbole de tsource manquant) {
        Se rappeler que le décodage est fini ;
        Retourner ;                                  /* le travail est maintenant terminé... */
     }
```

Créer une liste vide d'équations avec les symboles décodés durant cette étape de décodage ;

```
/* * D'abord, ajouter de nouveau symbole à la somme partielle de toutes les équations où le symbole apparaît. */
   Pour (chaque équation eq dans laquelle new_symb est une variable et a plus d'une variable inconnue) {
      Ajouter new_symb à partial_sum[eq];
      Supprimer entry(eq, new_esi) de la matrice H ;
      Si (le nouveau degré de l'équation eq == 1) { /* un nouveau symbole peut être décodé, se rappeler de l'équation */
         Ajouter eq à la liste des équations ayant les symboles décodés durant cette étape de décodage ;
      }
   }
```

/* * Puis finir avec les invocations récurrentes pour étape_de_décodage() pour chaque nouveau symboe décodé. */

```
   Pour (chaque équation eq dans la liste des équations ayant des symboles décodés durant cette étape de décodage) {

/* * À cause de la récurrence ci-dessous, on a besoin de vérifier que le décodage n'est pas fini, et que l'équation est
    __toujours__ de degré 1 */
      Si (décodage est fini) {
            couper;                                    /* sortie de la boucle */
        }
      Si ((degré de l'équation eq == 1) {
            Soit dec_esi l'ESI du nouveau symbole décodé dans l'éequation eq ;
          Supprimer entry(eq, dec_esi);
          Allouer une mémoire tampon, dec_symb, pour ce symbole et copier partial_sum[eq] à dec_symb;
          Informer l'appelant qu'un nouveau symbole a été décodé via une fonction de rappel ;
/* finalement, invoquer cette fonction de façon récurrente */
          étape_de_décodage(dec_esi, dec_symb);
      }
   }
Libérer la liste des équations ayant des symboles décodés ;
   Retourner ;
}
```

Adresse des auteurs

Vincent Roca
INRIA
655, av. de l'Europe
Inovallee; Montbonnot
ST ISMIER cedex 38334
France
mél : vincent.roca@inria.fr
http://planete.inrialpes.fr/people/roca/

Christoph Neumann
Thomson
12, bd de Metz
Rennes 35700
France
mél : christoph.neumann@thomson.net
URI : http://planete.inrialpes.fr/people/chneuman/

David Furodet
STMicroelectronics
12, Rue Jules Horowitz
BP217
Grenoble Cedex 38019
France
mél : david.furodet@st.com
URI : http://www.st.com/

Déclaration complète de droits de reproduction