UMONS
Université de Mons

Mémoire

Code détecteurs et correcteurs d'erreurs : découverte et expérimentation

Silon Christophe
Etudiant en Master
christophe.silon@student.umons.ac.be

Faculté des Sciences

Master en Sciences Informatiques

2022-2023

Sommaire

1 Introduction

A

2 État de l'art

Les réseaux de communications comportent du bruit, c'est-à-dire que lors de la transmission d'information, il est possible que des bits soient altérés.
Nous allons définir le taux d'erreurs comme étant la probabilité qu'un bit reçu soit différent du bit émis.
Ce taux d'erreurs va dépendre du moyen utilisé pour effectuer une communication. Les moyens de stockages sont également soumis aux bruits. La figure suivante reprend les différents taux d'erreurs en fonction du moyen de communication/stockage utilisé.

Ligne	Taux d'erreurs
Disquette	10^{-9} : à 5 Mo/s, 3 bits erronés par minute
CD-ROM optique	10^{-5} : 7ko erronés sur un CD de 700 Mo
DAT audio	10^{-5} : à 48 kHz, deux erreurs par seconde
Disques Blu-ray	$< 2 \cdot 10^{-4}$: environ 1.6Mo erronés pour 32Go
Hard Disk Drives (HDD)	$> 10^{-4}$: environ 10^6 erreurs pour 10Go
Technologie Flash	10^{-5}
Solid State Drives (SSD)	$< 10^{-5}$
Mémoires à semi-conducteurs	$< 10^{-9}$
Liaison téléphonique	entre 10^{-4} et 10^{-7}
Télécommande infrarouge	10^{-12}
Communication par fibre optique	10^{-9}
Satellite	10^{-6} (Voyager), 10^{-11} (TDMA)
ADSL	10^{-3} à 10^{-9}
Réseau informatique	10^{-12}

Figure 1: Ordre de grandeur du taux d'erreurs [2]

Shannon introduit en 1948 le théorème du codage de canal, permettant de transmettre un message à un destinaire en passant par un canal non fiable. Le message transmis est codé, c'est-à-dire que ce message va contenir de la redondance de telle sorte à permettre la détection/correction des erreurs.

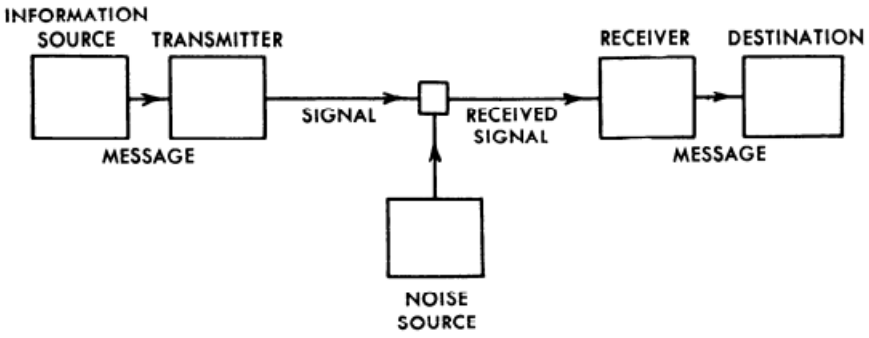


Figure 2: Diagramme de l'émission d'un message dans un système de communications [4]

Nous allons supposer dans la suite de l'article que nous allons passer par un canal binaire, où chaque symbole transmis dans le canal est un bit. L'émetteur du message transmettra dans le canal des symboles d'un alphabet $V = \{0, 1\}$.

Lors du codage d'un message m, le message est découpé en blocs de taille k. Chaque bloc M_i est ensuite traité individuellement l'un après l'autre.

Exemple de découpage d'un message de taille 24 par blocs de taille $k = 8$.
$m = 011101001101011010011101$

Bloc	Valeur
M_1	01110100
M_2	11010110
M_3	10011101

Chaque bloc obtenu après le découpage est codé grâce à une fonction de codage ϕ en rajoutant r symboles de redondance d'informations afin de pouvoir détecter voire corriger les erreurs.

Donc à chaque bloc (appelé mot) de taille k à transmettre $s = [s_1, ..., s_k]$, nous allons transmettre le bloc codé (appelé mot codé) $\phi(s) = [c_1, ..., c_k, ..., c_n]$. Le bloc codé $\phi(s)$ comportera alors $n = k + r$ symboles. L'ensemble $C_\phi = \{\phi(s) : s \in V^k\}$, image par ϕ de V^k, est appelé $code(n, k)$ [2].

Le rendement R d'un $code(n, k)$ est défini comme le taux de bits d'informations par rapport aux bits codés.

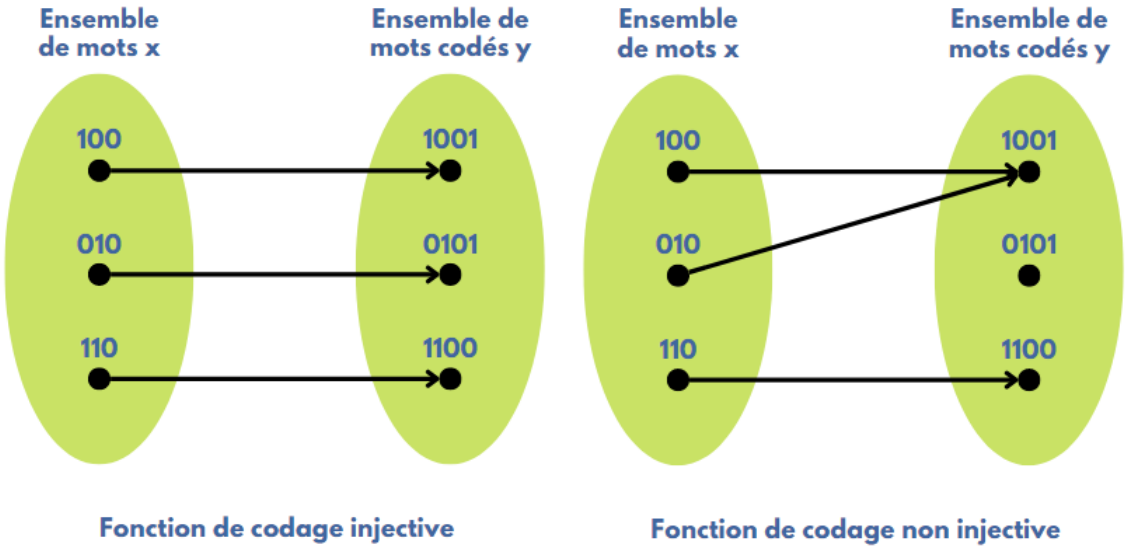
Par exemple, pour un bloc de taille $k = 8$ et un nombre de bits de redondances $r = 2$ (ce qui nous donne la taille totale du bloc codé $n = 8 + 2 = 10$), nous avons : $R = \frac{8}{10} = 0.8$

Un code(n, k) est *systématique* si une partie du mot codé coïncide avec l'entièreté du message. Le code est alors également *séparable* car on peut extraire directement tous les symboles d'informations du mot codé. Souvent, les k symboles d'informations sont placés dans l'ordre au début des mots codés.

L'intérêt des codes systématiques est de permettre un codage et un décodage rapide afin de transmettre les informations avec un coût le plus faible possible.

2.0.1 Décoder les messages

Pour pouvoir décoder un message, il est nécessaire que la fonction de codage ϕ soit injective, c'est-à-dire que tout mot code y est obtenu par le codage de au plus un mot x, tel que $\phi(x) = y$.



La détection d'erreurs se base sur le fait que le mot codé reçu a été altéré par le canal ($\overline{C_\phi} = V^n \setminus C_\phi$). Il existe deux types de corrections :

1. La correction directe : n'est possible uniquement dans le cas où la fonction de codage possède les caractéristiques nécessaires pour corriger un message, et le nombre de bits altérés ne doit également pas être trop élevé (ce nombre dépend de la fonction de codage utilisée).

2. La correction par retransmission : s'effectue dans le cas où le mot altéré n'a pas pu être corrigé directement, une requête ARQ (Automatic Repeat Request) est envoyé à l'émetteur du message afin de demander une retransmissions du message.

Le choix de la fonction de codage va être influencé par plusieurs facteurs :

1. Le rendement : doit être maximisé afin d'optimiser l'utilisation du réseau et réduire le taux d'erreur.

2. La probabilité de détecter l'erreur : doit être le plus élevé possible afin d'éviter

En fonction de la nature du moyen de transmission ou du stockage des données, les erreurs peuvent être aléatoires (isolées) ou en rafales (les unes à la suite des autres), ce qui va influencer le choix de fonction de codage utilisée. D'autres facteurs influencent ce choix, comme le rendement, une probabilité de détecter l'erreur élevée et une fonction permettant de rapidement coder et décoder les messages.

2.0.2 Probabilité d'erreurs lors de la transmission

Nous allons noter p la probabilité qu'un bit soit soit altéré. Nous faisons l'hypothèse que chaque bit a une même probabilité p d'être altéré. La probabilité qu'il y ait au moins une erreur dans un mot codé va dépendre de sa longueur N, donc le fait de rajouter de la redondance dans le message, augmente le risque d'erreur lors de la transmission. Ce qui implique que plus le rendement est faible, plus la probabilité pour qu'il y ait au moins une erreur dans un mot codé est élevée.

La probabilité pour qu'il y ait au moins un bit soit altéré dans un mot codé est obtenu avec la formule :

$$p_{error}(N, p) = 1 - (1 - p)^N$$

2.1 Code de bit de parité

Un code de bit de parité est un code systématique ajoutant un bit de parité à la fin d'un message. Lorsque l'on code un mot $m = (s_1, ..., s_k)$, on obtient un mode codé $\phi(m) = (s_1, ..., s_k, c_{k+1})$ où $c_{k+1} = (\sum_{i=1}^{k} s_i) \bmod 2$.
Ce bit de parité prendra donc la valeur 0 dans le cas où le nombre de bits valant 1 du message est pair, et la valeur 1 si ce nombre est impair.

Exemple pour le message $m = 011101001101011010011101$, avec $k = 8$

Mots M_i du message	Mots codés $\phi(M_i)$
$M_1 = 01110100$	$\phi(M_1) = 011101000$
$M_2 = 11010110$	$\phi(M_2) = 110101101$
$M_3 = 10011101$	$\phi(M_3) = 100111011$

Avec cette fonction de codage, nous obtenons toujours un nombre paire de bits valant 1 pour un mot codé. L'erreur est alors détecté si le nombre de bit valant 1 du mot codé est impaire. La fonction ne permet alors que de détecter l'erreur lorsque le nombre de bit altéré d'un mot codé est impaire.

Exemple de détection d'erreurs :

Mots codés $\phi(M_i)$	Mots codés altérés $\overline{\phi(M_i)}$	Nb erreurs	Erreur détectée?
$\phi(M_1) = 011101000$	$\overline{\phi(M_1)} = 011\mathbf{0}01000$	1	Oui
$\phi(M_2) = 110101101$	$\overline{\phi(M_2)} = 110\mathbf{0}01\mathbf{0}01$	2	Non
$\phi(M_3) = 100111011$	$\overline{\phi(M_3)} = 1\mathbf{1}011\mathbf{0}01\mathbf{0}$	3	Oui

Cette fonction permet uniquement de détecter (dans certains cas) les erreurs mais ne permet pas de les détecter.

2.1.1 Probabilité de détecter l'erreur

Pour détecter l'erreur lors de l'utilisation du code de bit de parité, nous devons dans un premier temps calculer la somme des probabilités d'avoir un nombre impaire de bits altérés.
Il faut donc calculer la probabilité qu'il y ait exactement k bits altérés. Pour calculer cette probabilité, il est nécessaire d'utiliser la loi binomiale.

$$P(X = k) = \binom{N}{k} p^k (1-p)^{n-k}$$

Exemple avec N = 16 et la probabilité d'erreur p = 0.01 :

Valeur de k	Probabilité d'erreur
0	0,851457
1	0,137609
2	0,010425
3	0,000491
4	0,000016
...	...

Maintenant que nous connaissons la probabilité d'avoir exactement k erreurs, nous pouvons calculer la probabilité de détecter l'erreur :

$$1 - \sum_{i=1}^{N/2} P(X = 2i)$$

Pour l'exemple précédent, nous obtenons une probabilité de détecter l'erreur de :
1 - (0,010425 + 0,000016 + ...) = 0,989559
Soit 98.96%.

2.2 Codes cycliques (CRC)

Les codes CRC (Cyclic Redundancy Code) sont des codes systématiques représentant les mots de taille n sous formes d'un polynôme de degré $n-1$. La position k d'un bit dans un mot donné représente un monôme de degré k. Un mot binaire $u = [u_{m-1}...u_0] \in \{0,1\}^m$ est représenté par un polynôme :

$$P_u(X) = \sum_{i=0}^{m-1} u_i X^i$$

Par exemple, pour un mot binaire donné $u = [110101]$, nous obtenons le polynôme $P_u = X^5 + X^4 + 0X^3 + X^2 + 0X + 1 = X^5 + X^4 + X^2 + 1$

Chaque monôme utilisé dans ce code fait partie du champ fini $\mathbb{F}_2$, c'est à dire qu'il ne peut avoir que la valeur 0 ou 1. Les opérations tels que la multiplication, l'addition, la soustraction et la division sur ce champ sont définies et satisfont des règles de base.

Les opérations sont définies comme suit :

Opération				
Multiplication	0 * 0 = 0	0 * 1 = 0	1 * 0 = 0	1 * 1 = 1
Addition	0 + 0 = 0	0 + 1 = 1	1 + 0 = 1	1 + 1 = 0
Soustraction	0 - 0 = 0	0 - 1 = 1	1 - 0 = 1	1 - 1 = 0
Division	/	0 / 1 = 1	/	1 / 1 = 1

La division d'un élément par un autre valant 0 est non définie dans cette structure.

Un polynôme $P_1(X)$ est divisible par un second polynôme $P_2(X)$ si le degré du premier est supérieur ou égal au degré du second.

Exemple de division de deux polynômes :

$$\begin{array}{llllll|l}
+x^7 & +x^5 & +x^4 & +x^3 & & +1 & x^3+x+1 \\
+x^7 & +x^5 & +x^4 & & & & x^4+1 \\
\hline
 & & & +x^3 & & +1 & \\
 & & & +x^3 & +x & +1 & \\
\hline
 & & & & +x & \textbf{Reste} &
\end{array}$$

Le code CRC se base sur le reste de la division euclidienne de deux polynômes, le premier polynôme représente le mot source binaire $s = [s_{k-1}...s_0]$, le second représente un *polynôme générateur* P_g de degré r :

$$P_g(X) = X^r + \sum_{i=0}^{r-1} g_i X^i$$

Le mot s est codé par le mot de code binaire $c = [c_{n-1}...c_0] = [s_{k-1}...s_0 c_{r-1}...c_0]$ où $[c_{r-1}...c_0]$ est la représentation du reste de la division euclidienne de $X^r.P_s$ par P_g [2], nous obtenons un polynôme vérifiant :

$$P_c = P_s.X^r + (P_s.X^r mod P_g)$$

Le mot codé est donc un multiple du polynôme générateur P_g, il est alors nécessaire pour l'émetteur et le récepteur du message de définir le polynôme générateur utilisé. Le récepteur détectera l'erreur si le polynôme reçu P_c est divisible par P_g sans qu'il n'y ait de reste de la division.

Exemple de codage pour le message 10011101, et un polynôme générateur donné $P_g = x^3 + 1$ de degré $r = 3$:

Le message source $s = 10011101$ est équivalent au polynôme $P_s = x^7+x^4+x^3+x^2+1$, $P_s.X^r = (x^7 + x^4 + x^3 + x^2 + 1) * x^3 = x^{10} + x^7 + x^6 + x^5 + x^3$, nous pouvons à présent calculer le reste $(P_s.X^r mod P_g)$:

$$\begin{array}{llllll|l}
+x^{10} & +x^7 & +x^6 & +x^5 & +x^3 & & x^3+1 \\
+x^{10} & +x^7 & & & & & x^7+x^3+x^2 \\
\hline
 & & +x^6 & +x^5 & +x^3 & & \\
 & & +x^6 & & +x^3 & & \\
\hline
 & & & +x^5 & & & \\
 & & & +x^5 & & +x^2 & \\
\hline
 & & & & & +x^2 & \textbf{Reste}
\end{array}$$

Nous obtenons $[c_{r-1}...c_0] = [100]$, le message codé vaut alors $c = [10011101100]$.

Exemple de détection d'erreur pour le message codé (repris de l'exemple précédent) 10011101100 et un polynôme générateur donné $P_g = x^3 + 1$ de degré $r = 3$. Le message après transmission a été altéré comme suit 100**1**0101**1**0, $P_c = x^{10} + x^7 + x^5 + x^3 + x^2 + x$. Calculons le reste $(P_s.X^r mod P_g)$:

$$
\begin{array}{cccccccc}
+x^{10} & +x^7 & +x^5 & +x^3 & +x^2 & +x & | & x^3+1 \\
+x^{10} & +x^7 & & & & & | & x^7+x^2+1 \\
\hline
 & & +x^5 & +x^3 & +x^2 & +x & & \\
 & & +x^5 & & +x^2 & & & \\
\hline
 & & & +x^3 & & +x & & \\
 & & & +x^3 & & & +1 & \\
\hline
 & & & & & +x & +1 & \textbf{Reste}
\end{array}
$$

Le reste différent de zéro indique une erreur.

En plus des propriétés intéressantes des CRC que nous verrons par la suite, le codage et le décodage d'un message est très rapide, ce qui rend ces codes particulièrement intéressant à haut débit.

2.2.1 La détection d'erreurs dans les CRC

Si le polynôme générateur possède plus de un coefficient non négatif, alors le code peut détecter toutes les erreurs portant sur un seul bit, car le reste de la division euclidienne entre ce polynôme et un bit d'erreur unique ne peut pas être nul.

Tous les polynômes générateurs peuvent détecter une erreur portant sur $r - i$ bits d'affilés, où i est égal à la position du coefficient non nul du monôme de degré le plus faible. Par exemple, le polynôme générateur $x^8 + x^5$ peut détecter $8 - 5 = 3$ bits altérés d'affilés.

Cette propriété se justifie par le fait qu'un polynôme de degré r rajoute r bits de redondances à un message, et que l'erreur ne peut pas être détectée que dans le cas où le polynôme de l'erreur P_e est un multiple du polynôme générateur P_g.

Cette propriété est particulièrement intéressant, elle nous permet de comprendre pourquoi tous les CRC existant possèdent le monôme $+1$ dans leur polynôme générateur. Elle justifie également l'utilisation des CRC dans le cas où les erreurs peuvent survenir en rafales.

Si le polynôme générateur possède $(x + 1)$ comme l'un de ses facteurs, alors le polynôme du mot codé (qui est un multiple de ce polynôme générateur P_g) a également $(x + 1)$ comme un de ses facteurs. Tous les polynômes ayant $(x + 1)$ comme un de ses facteurs possèdent un nombre pair de coefficients. C'est démontré en considérant le polynôme $v(x) = (1 + x)w(x)$. En substituant $x = 1$, on obtient $v(1) = (1+1)w(1) = 0$, impliquant le fait que $v(x)$ a un nombre pair de coefficients. Donc si le polynôme générateur possède $(x + 1)$ comme l'un de ses facteurs, alors tous les mots codés ont un nombre de coefficients non nul pair et toutes les erreurs portant sur un nombre impair de bits sont détectées [3].

Par exemple, pour le polynôme générateur $P_g = x^6+x^4+x+1 = (x+1)(x^5+x^4+1)$, et le polynôme d'un message source $P_s = x^9 + x^5 + x^4 + x^2$, on obtient le polynôme codé $P_c = x^{15} + x^{11} + x^{10} + x^8 + x^4 + x^3$. Ce polynôme contient 6 coefficients non nuls.

Considérons maintenant un modèle à double erreurs $e(x) = x^i + x^j = x^i(1 + x^{j-i}$ pour un i donné, $0 \leq i \leq n-2$ et pour un j donné, $i+1 \leq j \leq n-1$. Si $g(x)$ ne possède pas x comme un de ses facteurs et si ce polynôme ne divise pas de manière égale $[1 + x^{j-i}]$ avec $1 \leq j-1 \leq n-1$, alors le polynôme générateur détecte tous les modèles à double erreurs [3].

2.3 Somme de contrôle Internet (RFC 1071)

La somme de contrôle Internet est un protocole utilisé dans les protocoles TCP/UDP/IP pour détecter les erreurs, le principe est de rajouter un certains nombre de bits qui représentera le complément à un de la somme des différents mots d'un paquet. Il a été conçu pour avoir un bon équilibre entre le coût de calcul et la chance de détecter une erreur avec succès.

La somme de contrôle internet se base donc sur le complément à un de la somme des mots du segment. Le complément à un est obtenu en inversant tous les bits d'un nombre.

Par exemple pour un mot binaire donné $u = [11010100]$, on obtient le complément à un [00101011].

Lors de l'addition des mots du segment, le but est de garder la somme totale sur 16 bits, donc à chaque fois que l'addition de 2 mots donne un résultat sur 17 bits, le bit de poids le plus fort est reporté et additionné au bit de poids le plus faible.

Par exemple, pour l'addition de deux mots binaires sur 4 bits, avec $u_1 = [1001]$, $u_2 = [1010]$, on obtient

$$\begin{array}{cccccc} & & 1 & 0 & 0 & 1 \\ + & & 1 & 0 & 1 & 0 \\ \hline & \mathbf{1} & 0 & 0 & 1 & 1 \\ & & & & & \mathbf{1} \\ \hline & & 0 & 1 & 0 & 0 \end{array}$$

Lorsque le destinataire reçoit le message avec la somme de contrôle, il va à son tour calculer la somme de tous les mots du segment, une fois cette somme obtenue, il va l'additionner à la somme de contrôle, si un des bits de cette addition est égale à 0, alors le message a été corrompu lors du transfert.

Exemple de calcul de somme internet cs pour un paquet donné de 48 bits :

$m = 110011101010110000001000110111001011010101001111$
$m_1 = 1100111010101100$
$m_2 = 0000100011011100$
$m_3 = 1011010101001111$

m_1			1	1	0	0	1	1	1	0	1	0	1	0	1	1	0	0
m_2		+	0	0	0	0	1	0	0	0	1	1	0	1	1	1	0	0
			1	1	0	1	0	1	1	1	1	0	0	0	1	0	0	0
m_3		+	1	0	1	1	0	1	0	1	0	1	0	0	1	1	1	1
	1		1	0	0	0	1	1	0	0	1	1	0	1	0	1	1	1
																		1
$m_1 + m_2 + m_3$			1	0	0	0	1	1	0	0	1	1	0	1	**1**	**0**	**0**	**0**
`Complément à un`			0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	1

La somme de contrôle internet vaut donc 0111001100100111, l'expéditeur du message m va donc rajouter cette somme à la fin du message. Le récepteur reçoit le message et calcule si celui-ci est corrompu :

m_1			1	1	0	0	1	1	1	0	1	0	1	0	1	1	0	0
m_2		+	0	0	0	0	1	0	0	0	1	1	0	1	1	1	0	0
			1	1	0	1	0	1	1	1	1	0	0	0	1	0	0	0
m_3		+	1	0	1	1	0	1	0	1	0	1	0	0	1	1	1	1
	1		1	0	0	0	1	1	0	0	1	1	0	1	0	1	1	1
																		1
$m_1 + m_2 + m_3$			1	0	0	0	1	1	0	0	1	1	0	1	1	0	0	0
cs		+	0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	1
			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Tous les bits de la somme entre $(m_1 + m_2 + m_3) + cs$ valent 1, le message n'est donc pas corrompu.

Reprenons l'exemple précédent, et altérons quelques bits lors de l'envoi du message afin d'illustrer l'erreur, le destinataire reçoit le message et découpe le message en mots de 16 bits :

$m_1 = 1100101010101100$
$m_2 = 0000100011011000$
$m_3 = 1011010101001111$
$cs = 0111001100100111$

m_1			1	1	0	0	1	0	1	0	1	0	1	0	1	1	0	0
m_2	+		0	0	0	0	1	0	0	0	1	1	0	1	1	0	0	0
			1	1	0	1	0	0	1	1	1	0	0	0	0	1	0	0
m_3	+		1	0	1	1	0	1	0	1	0	1	0	0	1	1	1	1
		1	1	0	0	0	1	0	0	0	1	1	0	1	0	0	1	1
																		1
$m_1 + m_2 + m_3$			1	0	0	0	1	0	0	0	1	1	0	1	0	1	0	0
cs	+		0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	1
			1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1

Au moins un des bits de la somme entre $(m_1 + m_2 + m_3) + cs$ vaut 0, le message a donc été corrompu.

Le calcul de la somme de contrôle possède plusieurs propriétés mathématiques intéressantes, qui peuvent être exploitées afin d'accélérer le codage et la vérification d'erreur [1].

- **L'associativité et la commutativité** : tant que le message est bien découpé en mots de 16 bits, l'addition de chaque mot peut se faire dans n'importe quel ordre et il peut être divisé en groupe. Par exemple pour l'addition de $m_1 + m_2 + m_3 + m_4$, on pourrait l'effectuer dans l'ordre suivant : $m_1 + (m_3 + m_2) + m_4$.

- **L'addition en parallèle** : si la machine possède des mots dont la taille est un multiple de 16, il est possible d'accélérer le codage en codant en parallèle les mots. Dû au fait que l'addition est associative, au lieu d'additionner les mots un par un, nous allons utiliser l'entièreté de la taille du mot de la machine, en faisant les additions en parallèle. Il faut faire attention que l'addition de chaque mot peut provoquer un bit de débordement, et que celui-ci ne doit pas être rajouté à un autre mot, mais doit être rajouté au bit de poids le plus faible de ce mot.

  Exemple d'addition en parallèle sur une machine avec une architecture 32 bits, le message à envoyer est découpé en 4 mots $m_1 + m_2 + m_3 + m_4$, les bits des mots m_1 et m_2 sont mis dans une variable commune de 32 bits et additionné avec la variable contenant m_3 et m_4. Une fois que tous les mots ont été additionnés, il suffit d'additionner les 16 premiers bits de la variable contenant le résultat avec les 16 derniers bits.

- **Actualisation incrémentielle** : si la somme de contrôle a déjà été calculée, qu'un des mots a changé et qu'il est nécessaire de recalculer cette somme. Pour calculer cette nouvelle somme de contrôle C', il suffit simplement de rajouter à l'ancienne somme de contrôle C la différence entre le nouveau mot m' et le mot qui a changé m. Ce phénomène s'explique par le fait que l'addition est associative.

$$C' = C + (-m) + m' = C + (m' - m)$$

Une implémentation efficace est primordiale pour avoir de bonnes performances.

2.3.1 La détection d'erreurs dans la somme de contrôle internet

La somme de contrôle internet se base donc sur la somme des différents mots du message, pour ne pas détecter l'erreur, il suffit que la somme de chaque colonne reste inchangée.

Par exemple, pour un message donné, il suffit que 2 bits à la même position d'un mot différent (dont les valeurs étaient respectivement 0 et 1 ou 1 et 0) soient inversés :
$m_1 = 0100111010101100$
$m_2 = 1000100011011100$
$m_3 = 1011010101001111$
$cs = 0111001100100111$
$m_1 + m_2 + m_3 + cs = 1111111111111111$

La somme de contrôle va permettre de détecter toutes les erreurs en rafale portant sur au plus 15 bits [1]. Sur des données uniformément distribuées, il détecte d'autres types d'erreurs à un taux proportionnel à 1 sur 2^{16}. Sur des données non uniformément distribuées, ses performances peuvent être nettement inférieures. Une étude montre que l'utilisation de cette somme de contrôle sur des données réelles est comparable à des données uniformément distribuées avec une somme de contrôle sur 10 bits [6]. Cela signifie qu'au lieu d'avoir une chance de détecter les autres types d'erreurs avec un taux proportionnel à 1 sur 2^{16}, on aurait plutôt un taux de 1 sur 2^{10}.

Avant de tester la somme de contrôle d'un paquet, à la couche liaison, les CRC vont s'occuper de détecter les erreurs. Si le CRC utilisé ne détecte aucune erreur, cette somme va tester à son tour si une erreur est présente. Une étude de Stone et al. [5] montre que entre un paquet sur quelques millions et un paquet sur 10 milliards comportera une erreur non détectée par cette somme.

Une grande limitation de cette somme de contrôle est le fait que la somme d'un ensemble de valeurs de 16 bits est la même, quel que soit l'ordre dans lequel les additions sont effectuées (commutativité). Donc si pour une certaine raison, deux mots sont inversés dans le message, l'erreur ne sera pas détectée.

2.4 Codes de Hamming

A

2.5 Codes de Reed-Solomon

A

3 Conclusion

4 Bibliographie

References

[1] Robert Braden, David Borman, and Craig Partridge. "Computing the internet checksum". In: *ACM SIGCOMM Computer Communication Review* 19.2 (1989), pp. 86–94.

[2] Jean-Guillaume Dumas et al. *Théorie des codes-3e éd.: Compression, cryptage, correction*. Dunod, 2018.

[3] Tenkasi V Ramabadran and Sunil S Gaitonde. "A tutorial on CRC computations". In: *IEEE micro* 8.4 (1988), pp. 62–75.

[4] Claude Elwood Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[5] Jonathan Stone and Craig Partridge. "When the CRC and TCP checksum disagree". In: *ACM SIGCOMM computer communication review* 30.4 (2000), pp. 309–319.

[6] Jonathan Stone et al. "Performance of checksums and CRCs over real data". In: *IEEE/ACM Transactions on Networking* 6.5 (1998), pp. 529–543.