Compte Rendu du TP2 de Communication numérique

PARA Yaël GONZALEZ Jules TEYSSIER Théo

6 Avril 2020

1 Code de Hamming

1) Un code de Hamming peut détecter jusqu'à une erreur.

2) Un code de Hamming est bien linéaire.

3) Le surcoût r introduit par un code de Hamming est donné par la formule :
$r = \lceil log_2(k) \rceil$

4) L'efficacité maximum des codes de Hamming en fonction de r :
r = 3, efficacité maximum de 0.57
r = 4, efficacité maximum de 0.67
r = 5, efficacité maximum de 0.76

5) Cette efficacité maximum est obtenue avec la formule : $R = k/n$ en prenant le k maximum pour chaque r.

6) L'efficacité maximum tend vers 1 lorsque k tend vers l'infini.

7) Dans le cas d'un code de Hamming [7, 4] les bits de controle seront :
$x_{001} = m_1 + m_2 + m_4$
$x_{010} = m_1 + m_3 + m_4$
$x_{100} = m_2 + m_3 + m_4$

8) Pour corriger une erreur à la reception, on fait un XOR entre les posistion des bits à 1 dans le code reçu et on change le bit à la position obtenue (si on obtient 0 il n'y a pas eu d'erreur donc on ne fait rien).
Par exemple pour le mot 0101100 on à :
$i = 010 + 100 + 101 = 011$
On a donc une erreur sur le 3ème bit, le message codé est donc : 0111100
Il faut ensuite enlever les bits de parité, le message est donc : 1100

9) Matrice génératrice pour ce code [7, 4] :

1	1	1	0	0	0	0
1	0	0	1	1	0	0
0	1	0	1	0	1	0
1	1	0	1	0	0	1

10) Matrice de contrôle pour ce code [7, 4] :

0	0	0	1	1	1	1
0	1	1	0	0	1	1
1	0	1	0	1	0	1

11 et 13) Le code source du codeur / décodeur-correcteur est disponnible dans le fichier hamming/hamming.c

12) Voici les tests avec notre codeur :

```
teyssier@Theo-debian:~/Documents/Polytech/Semestre6/CN/TP2/Hamming$ ./hamming 0000
mot code : 0000000
k : 4
r : 3
```

FIGURE 1 – mot 0000

```
teyssier@Theo-debian:~/Documents/Polytech/Semestre6/CN/TP2/Hamming$ ./hamming 1111
mot code : 1111111
k : 4
r : 3
```

FIGURE 2 – mot 1111

```
teyssier@Theo-debian:~/Documents/Polytech/Semestre6/CN/TP2/Hamming$ ./hamming 1011
mot code : 0110011
k : 4
r : 3
```

FIGURE 3 – mot 1011

```
teyssier@Theo-debian:~/Documents/Polytech/Semestre6/CN/TP2/Hamming$ ./hamming 10101010101
mot code : 1011010010101011
k : 11
r : 5
```

FIGURE 4 – mot 101 0101 0101

```
teyssier@Theo-debian:~/Documents/Polytech/Semestre6/CN/TP2/Hamming$ ./hamming 11110000101
mot code : 1011111000001011
k : 11
r : 5
```

FIGURE 5 – mot 111 1000 0101

12) Voici les tests avec notre décodeur :

FIGURE 6 – mot 000 0001

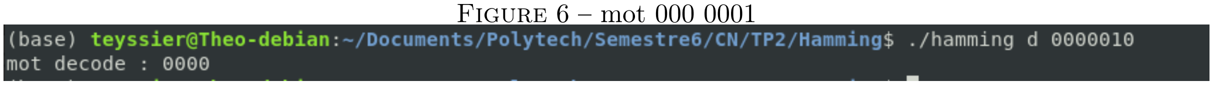

FIGURE 7 – mot 000 0010

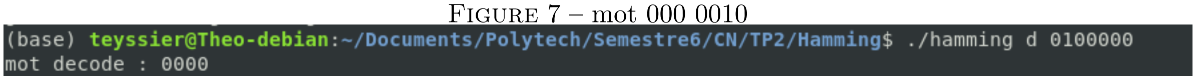

FIGURE 8 – mot 010 0000

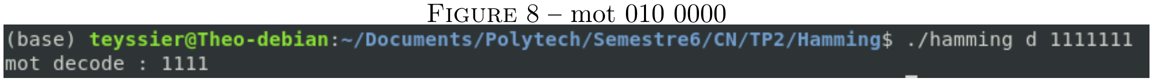

FIGURE 9 – mot 111 1111

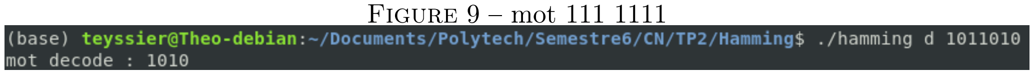

FIGURE 10 – mot 101 1010

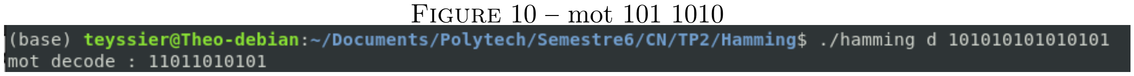

FIGURE 11 – mot 101 0101 0101 0101

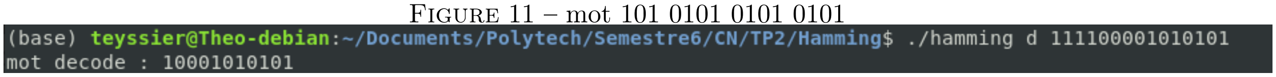

FIGURE 12 – mot 111 1000 0101 0101

2 Detection d'erreur CRC

1) Le code obtenu avec l'ajout d'un CRC est systématique, car le code obtenu est composé du message envoyé sur les bits de poids fort suivi des bits de parité sur les bits de poids faible.

2) Un code C est cyclique s'il est invariant par permutation circulaire, c'est à dire que l'ensemble des mots du code C est stable par décalage circulaire :

3) La représentation polynomiale du message 10011101 est $D(X) = X^7 + X^4 + X^3 + X^2 + 1$

4) Le nombre de bits de parité ajoutés au message dans le code est égal au degré du polynôme générateur. Ici, nous allons ajouter 4 bits de parité au message afin de former le code.

5) Afin de former le code $C(x)$ à transmettre, on doit ajouter 1000 après le message à envoyer, le code est alors 100111011000. Ceci est démontré dans le calcul ci dessous :

```
100111010000|10111
10111       |--------
-----       |10101000
0010010     |
  10111     |
-------     |
  0010110   |
    10111   |
    -----   |
    00001000|
```

6)Nous vérifions que $C(X)$ est un multiple de $G(X)$ en effectuant la division euclidienne suivante :

$$\begin{array}{l|l} 100111011000 & 10111 \\ \underline{10111} & \overline{10101000} \\ 0010010 & \\ \quad\underline{10111} & \\ \quad 0010111 & \\ \quad\quad\underline{10111} & \\ \quad\quad 00000000 & \end{array}$$

Le reste est nul, donc $C(X)$ divise bien $G(X)$.
7) Afin de détecter les erreurs de transmission, nous divisons le mot de code $W(X)$ reçu par le polynôme générateur $G(X)$ en utilisant l'opération XOR (l'addition modulo 2). Si le reste de la division est non nul, il y a une erreur.
$W(X)$ est de la forme $R(X) = C(X) + E(X)$, où $C(X)$ représente le code qui est formé par le polynôme générateur $G(X)$, et $E(X)$ le polynôme représentatif des erreurs en transmission (les bits sont à 1 aux positions des erreurs dans le polynôme $W(X)$).
On sait que $C(X)$ est un multiple de $G(X)$, donc la detection d'erreur dans le mot $W(X)$ est déterminée par le reste de la division de $E(X)$ par $G(X)$.
Le polynôme $G(X)$ peut détecter les erreurs de 1 bit, car dans ce cas là, le polynôme $E(X)$ est de la forme $E(X) = X^i$, avec $0 \leq i \leq 11$.
Le polynôme $G(X)$ ne peut diviser un tel polynôme, donc la division euclidienne de $E(X)$ par $G(X)$ donnera forcément un reste non nul.

8) Dans le cas d'une erreur double, on a $E(X) = X^i + X^j = X^i \times (1 + X^j - i)$, avec $0 \leq i, j \leq 11$. Ainsi, pour être assurés d'une détection d'erreur sur 2 bits, il faut que $G(X)$ ne divise pas de polynôme de la forme $X^k + 1$, avec $0 \leq k \leq 11$.
On déduit facilement que $G(X)$ de degré 4 ne divise pas les polynômes de la forme $X^k + 1$ pour $0 \leq k \leq 4$.
De plus, $G(X) = X^4 + X^2 + X + 1$ ne divise pas :
$X^5 + 1 = X \times G(X) - (X^3 + X^2 + X - 1)$
$X^6 + 1 = (X^4 + X^2 + X + 1) \times (X^2 + 1) - X^3 + X + 2$
$X^7 + 1 = G(X) \times (X^3 - X - 1) + (2X^2 + 2X + 2)$
$X^8 + 1 = (X^4 + X^2 + X + 1) \times (X^4 - X^2 - X) + 2X^3 + 2X^2 + X + 1$
$X^9 + 1 = (X^4 + X^2 + X + 1) \times (X^5 - X^3 - X^2 + 2) - 2X^3 - X^2 - 2X - 1$
$X^{10} + 1 = (X^4 + X^2 + X + 1) \times (X^6 - X^4 - X^3 + 2X + 2) - X^3 - 4X^2 - 4X - 1$
$X^{11} + 1 = (X^4 + X^2 + X + 1) \times (X^7 - X^5 - X^4 + 2X^2 - 1) - 4X^3 - 3X^2 - X + 2$
Donc $G(X)$ détecte toutes les erreurs sur 2 bits.

9) $G(X)$ et $C(X)$ ont un nombre pair de coefficients non nuls, donc si le nombre d'erreur sur $W(X)$ est impair, E(X) possède un nombre impair de coefficients non nuls, est n'est donc pas factorisable par $X+1$. Or, d'après le cours, $G(X)$ est un polynôme factorisable par $X + 1$, car il possède un nombre pair de coefficients. On en déduit que $G(X)$ ne divise pas $E(X)$, et que $G(X)$ peut détecter les erreurs sur un nombre impair de bits.

10) Une erreur en rafale de longueur 4 aura une représentation polynomiale de la forme : $E(X) = X^(i + 3) + X^(i + 2) + X^(i + 1) + X^i = X^i \times (X^(i + 2) + X^(i + 1) + X^i + 1) = X^i \times P(X)$, avec $0 \leq i \leq 8$.
$G(X)$ ne divise pas X^i, de plus $G(X)$ ne divise pas le polynôme $P(X) = (X^(i + 2) + X^(i + 1) + X^i + 1)$,

car $G(X)$ possède un coefficient de plus que $P(X)$. De plus, $G(X)$ ne divisera pas les polynômes possédant moins de 4 coefficients, on peut donc en déduire que $G(X)$ détecte les rafales d'erreur de longueur inférieure ou égale à 4.

11 et 12) Nous obtenons les résultats de codage suivants grâçe à notre fonction de codage avec $G(X)$ en polynôme générateur :

Mot Source	Codage
11001101	110011010010
10101011	101010111011
00110101	001101011010

13 et 14) Les mots 10001101000 et 11101011101 génèrent une erreur, et les codes précédemment générés dans la question précédente sont bien décodés sans erreur.

15) L'opération coûteuse est le Xor réalisé entre 2 mots de même longueur, nous ne considérons pas l'opération Xor bit à bit.
Dans notre algorithme de codage, après chaque itération, nous effectuons un Xor bit à bit entre G (polynôme générateur de degré d) et une partie de T (le dividende) de même longeur que G.
Après chaque itération, nous effectuons un décalage du bit de poids fort de T, qui change après chaque Xor.
L'algorithme s'arrête lorsque le reste est calculé, c'est à dire lorsque le bit de poids fort du résultat du Xor est situé sur les d derniers bits de T.
Ainsi, dans le pire des cas, nous effectuons un décalage de 1 du bit de poids fort après chaque itération, et dans ce cas là nous effectuons autant de Xor qu'il y a de bits dans le message à coder.

16) Pour effectuer le décodage, nous effectuons une division euclidienne entre C (code reçu) et G afin de détecter la présence ou non d'erreur dans le message. Donc comme pour l'algorithme de codage, la complexité au pire en nombre d'opérations Xor entre 2 mots de même longueur est égale au nombre de bits constituant la partie "message" du code, c'est à dire : nombre de bits de C - nombre de bits de G + 1