université de BORDEAUX

TD4 - Contrôle d'Erreurs

1 Bit de parité

Pour commencer, nous considérons un code très simple. À la fin du message qu'on veut envoyer nous rajoutons un bit qui correspond à la somme modulo 2 de tous les bits du message. Si, par exemple, on veut envoyer le message 1101001, alors on envoie le message 11010010. Ainsi le receveur du message pourra tester si le message est correct en réalisant le calcul : 1+1+0+1+0+0+1=0 mod 2. Si le premier bit change durant le transfert et que le message est devenu 01010010 alors la somme (0+1+0+1+0+0+1=1 mod 2) ne correspond plus au bit de parité, on sait donc qu'il y a une erreur.

1. Les messages suivant respectent-ils le code :
   1001011101110101111010 et 1111001010001010111011.
2. Quelles critiques pouvez-vous émettre sur ce code de vérification ?

2 Double-bit de parité

On se propose d'utiliser l'idée de l'exercice précédent de façon un peu plus élaborée. On commence par découper notre message en mots de taille fixe. Par exemple, disons qu'on veut envoyer le message 011101010. On décide de découper le message en mots de taille 3, on obtient 011.101.010. On place ce message dans un tableau et on calcul le bit de parité pour chaque ligne et chaque colonne comme dans le tableau suivant :

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

On envoie alors le message : 011101010001100.

1. Déterminer la valeur d'un tel code pour le message 1010111010111100 en utilisant un découpage en mots de taille 4.
2. Montrer que ce code permet non seulement de détecter mais aussi de corriger une erreur unique.
3. Lorsqu'il y a exactement deux erreurs ce code permet-il de les détecter ?
4. Lorsqu'il y a exactement deux erreurs ce code permet-il de les corriger ?
5. Lorsqu'il y a exactement trois erreurs ce code permet-il de les détecter ?
6. Lorsqu'il y a trois erreurs ce code permet-il de les corriger ?

Au final pour pouvoir corriger des erreurs on est obligé d'ajouter beaucoup de bits supplémentaires ! Par exemple sur les CD audio, l'encodage utilisé dépense 588 bits sur la galette pour n'encoder que 192 bits de musique. Les 588 bits sont d'ailleurs éparpillés sur la galette, pour éviter qu'une rayure en endommage une grande partie d'un seul coup.

3 Checksum TCP/UDP/IP

Le principe de somme de contrôle utilisé par les protocoles TCP, UDP et IP est très simple : c'est le complément à un de la somme par compléments à un des données, sur 16 bits. Autrement dit : on fait la somme des données du message par morceaux de 16 bits ; à chaque fois qu'il y a une retenue on ajoute 1 à la somme ; et à la fin on inverse les bits du résultat. S'il y a un nombre impair d'octets, on ajoute un octet nul pour faire un morceau de 16 bits.

Ainsi, pour le message `45 34 33 45 91 11 01 01` :

- on effectue d'abord le calcul `0x4534 + 0x3345 = 0x7879` qui ne produit donc pas de retenue.
- On continue la somme avec `0x7879 + 0x9111 = 0x1098a` qui produit une retenue, donc on ajoute 1 pour obtenir `0x098b` sur 16 bits.
- On termine avec `0x098b + 0x0101 = 0x0a8c`, pas de retenue.
- Et l'on inverse les bits : `~0x0a8c = 0xf573`

Le checksum est ainsi `0xf573`, le message envoyé est alors : `45 34 33 45 91 11 01 01 f5 73`.

Si l'on refait le calcul du checksum sur ce message envoyé, le début du calcul est le même, mais l'on termine avec `0x0a8c + 0xf573 = 0xffff` et après inversion on obtient bien `0x0000`, la somme est bien tombée juste.

1. Calculer le checksum du message `84 56 82 12 74 45`.
2. Vérifier que le message `f1 34 10 23 fe a7` a bien une somme de contrôle nulle.
3. Donner des exemples d'erreurs de transmission détectées par cette somme de contrôle. Est-ce qu'on peut les corriger ?

Pour aller plus loin

1. Dans le cas de l'en-tête IP, la somme porte sur l'en-tête seulement, et elle est écrite au milieu de l'en-tête. On commence ainsi par calculer la somme avec le champ checksum initialisé à zéro, et l'on note ensuite la somme dans l'en-tête. Pourquoi est-ce équivalent à ajouter le checksum à la fin de l'en-tête ?
2. Donner des exemples d'erreurs de transmissions non détectées par cette somme de contrôle.

Pour information, dans le cas de TCP et UDP, la somme de contrôle ne porte pas seulement sur l'en-tête TCP/UDP et les données, mais aussi sur un pseudo-en-tête contenant les adresses IP source et destination, le numéro de protocole (6 pour TCP, 17 pour UDP), et la longueur. La présence des adresses IP dans ce calcul brise un peu la séparation stricte entre IP et TCP/UDP...

4 CRC

Le code CRC va utiliser une clef et réaliser un calcul pour obtenir un code similaire à celui du bit de parité mais plus robuste, c'est en fait une division de polynôme, qu'on effectue comme un genre de division posée en binaire avec xor. Regardons un exemple sur N=4 bits, la clef choisie est K=0x3, soit un diviseur G=10011 [1] et le message est 1001011, on ajoute au message des zéros pour les N=4 bits qui composeront le CRC. Comme pour une division en décimal, le principe est d'éliminer à chaque étape le 1 le plus à gauche en "soustrayant" la clef avec xor. On s'arrête lorsque le reste a moins de bits que la clef.

	1	0	0	1	0	1	1	**0**	**0**	**0**	**0**
xor	1	0	0	1	1	⋮	⋮	⋮	⋮	⋮	⋮
		0	0	0	1	1	⋮	⋮	⋮	⋮	⋮
			0	0	1	1	1	⋮	⋮	⋮	⋮
				0	1	1	1	⋮	⋮	⋮	⋮
					1	1	1	0	⋮	⋮	⋮
xor					1	0	0	1	1	⋮	⋮
						1	1	1	1	⋮	⋮
xor						1	0	0	1	1	⋮
							1	1	0	1	⋮
xor							1	0	0	1	1
								1	**0**	**0**	**1**

On envoie alors le message 10010111001. À nouveau, à la réception, il nous suffira de faire le même calcul pour vérifier si le message n'a pas été modifié. En fait, on peut aussi mettre le message entier en haut (donc avec les 4 chiffres du CRC en haut à droite à la place des zéros), et constater que l'on retombe à zéro.

1. Trouver le code CRC-4 utilisant la clef K=0x9 correspondant au message 0010111011 (sur 10 bits).
2. Le message 0100101101 est-il correct sachant qu'on utilise un code CRC-3 de clef 0x2 ?
3. Le protocole Ethernet utilise un code CRC-32 pour son checksum [2]. Si l'on reçoit une trame complètement aléatoire, quelle est la probabilité pour que l'on constate que le CRC est tout de même correct ?

1. Il faut concaténer à la clef K sur N bits un bit supplémentaire de poids forts à 1 pour former le diviseur G=(1|K) sur N+1 bits.
2. avec G=0x104C11DB7=100000100110000010001110110110111

Pour aller plus loin

4. Donner un exemple d'erreur de transmission qui n'est pas détectée par le CRC.
5. Dans le cas où on utilise le diviseur G=10011, si une erreur de transmission inverse deux bits, est-on sûr de le détecter ?