Commentaires

Version du 13/05/05.

Partie Codage (presque) complete.
Manque encore les illustrations du CRC et meilleures explications.

Introduction

Vers la fin des années 30, Claude Shannon démontra qu'à l'aide de "contacteurs" (interrupteurs) fermés pour vrai et ouverts pour faux il était possible d'effectuer des opérations logiques en associant le nombre 1 pour vrai et 0 pour faux.
Ce codage de l'information à deux états est nommé base binaire. C'est grâce à ce codage que fonctionnent les ordinateurs. Il consiste à utiliser deux états électriques différents pour différencier les deux états, le "0" et le "1", et ainsi coder les informations.
Les nombres binaires étant de plus en plus longs, il a fallu introduire une nouvelle base : la base hexadécimale basée sur un système à 16 caractères par comparaison des deux de la base binaire ce qui implique une forte simplification(dans la longueur) des messages.

Chapitre 1

Le Codage dans l'informatique

1.1 Pourquoi coder ?

La communication nécessite la compréhension entre les deux entités communicantes. L'émetteur envoie de l'information au récepteur qui doit savoir l'interpréter pour la comprendre. Ainsi, le codage de l'information est la première étape de toute communication.

1.2 L'histoire du codage

Depuis toujours le codage des informations a eu une trés grande importance pour l'homme ; de l'apprentissage des signes, de la parole puis de l'écrit. Au début, l'écrit consistait essentiellement dans des dessins puis vint un alphabet plus simple à utiliser qui offrait de multiples combinaisons pour une plus grande richesse de l'expression. En réalité, les caractères de l'écrit ne sont que des symboles interprétables.
Le morse a été le premier codage à permettre une communication longue distance. C'est Samuel F.B.Morse qui l'a mis au point en 1844. Ce code est composé de points et de tirets (equivalent à un codage binaire). Il permit d'effectuer des communications beaucoup plus rapides que ne le permettait le système de courrier de l'époque aux Etats-Unis : le Pony Express. L'interpréteur était l'homme à l'époque, il fallait donc une bonne connaissance du code.
De nombreux codes furent inventés dont le code Baudot du nom de son inventeur Emile Baudot, et appelé "Murray Code" par les anglais.
Le 10 mars 1876, le Dr Graham Bell met au point le téléphone, une invention révolutionnaire qui permet de faire circuler de l'information vocale dans des lignes métalliques.
Ces lignes permirent l'essor des téléscripteurs, des machines permettant de coder et décoder des caractères grâce au code Baudot (les caractères étaient alors codés sur 5 bits, il y avait donc 32 caractères uniquement...).
Dans les années 60, le code ASCII (American Standard Code for Information Interchange) est adopté comme standard. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles.
Nous allons donc voir dans cette partie tout d'abord comment sont codés les caractères, puis comment se déroule le contrôle d'erreurs lors de transmissions grâce aux codes détecteurs et correcteurs.

1.3 Le codage des caractères

La mémoire de l'ordinateur conserve toutes les données sous forme numérique. Il n'existe pas de méthode pour stocker directement les caractères. Chaque caractère possède donc son équivalent en code numérique : c'est le code ASCII . Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127).

Les codes 0 à 31 ne sont pas des caractères. On les appelle caractères de contrôle car ils permettent de faire des actions telles que : retour à la ligne (CR), bip sonore (BEL). Les codes 65 à 90 représentent les majuscules Les codes 97 à 122 représentent les minuscules (Il suffit de modifier le 6ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale.).

1.3.1 Caractères ASCII Etendue

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Pour coder ce type de caractère il faut recourir à un autre code. Le code ASCII a donc été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...). Ce code attribue les valeurs 0 à 255 (donc codées sur 8 bits, soit 1 octet) aux lettres majuscules et minuscules, aux chiffres, aux marques de ponctuation et aux autres symboles (caractères accentués dans le cas du code iso-latin1).

Le code ASCII étendu n'est pas unique et dépend fortement de la plateforme utilisée, d'où tous les problèmes de caractères changés lors d'un passage d'une plateforme à une autre.

A	01000001	J	01001010	S	01010011
B	01000010	K	01001011	T	01010100
C	01000011	L	01001100	U	01010101
D	01000100	M	01001101	V	01010110
E	01000101	N	01001110	W	01010111
F	01000110	O	01001111	X	01011000
G	01000111	P	01010000	Y	01011001
H	01001000	Q	01010001	Z	01011010
I	01001001	R	01010010	espace	00100000

TAB. 2.1 – Un extrait de schéma de codage ASCII.

Par exemple, le codage suivant ASCII du message : LICENCE INFO est la chaîne :
01001100 **01001001** 01000011 **01000101** 01001110 **01000011** 01000101 **00100000** 01001001 **01001110** 01000110 **01001111**

1.3.2 Autres codes existants

Malgré le fait que le code ASCII soit le standard pour le codage de caractères, il en existe bien d'autres. Les plus connus sont le code EBCDIC (Extended Binary-Coded Decimal Interchange Code), développé par IBM, qui permet de coder des caractères sur 8 bits et le code Unicode sur 16 bits mis au point en 1991 qui permet de représenter n'importe quel caractère indépendamment de tout système d'exploitation ou langage de programmation.

1.4 Le contrôle d'erreurs

Le codage binaire est très pratique pour une utilisation dans des appareils électroniques tels qu'un ordinateur, dans lesquels l'information peut être codée grâce à la présence ou non d'un signal électrique. Cependant le signal électrique peut subir des perturbations (rayonnements électromagnétiques, distortion, présence de bruit), notamment lors du transport des données sur un long trajet et transmettre des bits erronés. Ainsi, le contrôle de la validité des données est nécessaire pour certaines applications (professionelles, bancaires, industrielles, confidentielles, relatives à la sécurité, ...).
C'est pourquoi il existe des mécanismes permettant de garantir un certain niveau d'intégrité des données, c'est-à-dire de fournir au destinataire une assurance que les données reçues sont bien similaires aux données émises. La protection contre les erreurs peut se faire de deux façons : soit en fiabilisant le support de transmission, c'est-à-dire en se basant sur une protection physique ;

soit en mettant en place des mécanismes logiques de détection et de correction des erreurs.
La plupart des systèmes de contrôle d'erreur au niveau logique sont basés sur un ajout d'information (on parle de "redondance") permettant de vérifier la validité des données. On appelle somme de contrôle cette information supplémentaire. Il existe deux sortes de code de contrôle ; les codes détecteurs qui détectent seulement l'erreur et les codes correcteurs qui eux détectent puis corrigent l'erreur.

1.5 Les Codes détecteurs

Il existe plusieurs méthodes de détection et/ou de correction d'erreurs dans le monde des communications, allant du plus simple au plus complexe. Le contrôle de parité est un exemple parmi les plus simples.

Le contrôle de parité ,appelé parfois VRC, pour Vertical Redundancy Check(vérification par redondance verticale), est un des systèmes de contrôle les plus simples. Il consiste à ajouter un bit supplémentaire (appelé bit de parité) à un certain nombre de bits de données appelé mot de code (généralement 7 bits, pour former un octet avec le bit de parité) dont la valeur (0 ou 1) est telle que le nombre total de bits à 1 soit pair. Pour être plus explicite il consiste à ajouter un 1 si le nombre de bits du mot de code est impair, 0 dans le cas contraire.

Le système de contrôle de parité ne détectant que les erreurs en nombre impair, il ne permet donc de détecter que 50% des erreurs. Ce système de détection d'erreurs possède également l'inconvénient majeur de ne pas permettre de corriger les erreurs détectées (le seul moyen est d'exiger la retransmission de l'octet erroné...).

Le contrôle de parité croisé (aussi appelé contrôle de redondance longitudinale ou Longitudinal Redundancy Check, noté LRC) consiste non pas à contrôler l'intégrité des données d'un caractère, mais à contrôler l'intégrité des bits de parité d'un bloc de caractères. Comme avant, à chaque octet est ajouté un bit de parité horizontale (LRC). Mais en plus, chaque groupe d'octets est aligné pour un contrôle de parité verticale (VRC). A tous les bits de rang 0, on ajoute un bit de parité, puis de même aux bits de rang 1, etc. A la fin, le contrôle de parité du contrôle vertical doit être le même que celui du contrôle horizontal.

FAIRE UN SCHEMA DE PARITE CROISE

Une autre méthode beaucoup utilisée est le CRC. Le contrôle de redondance cyclique (noté CRC, ou en anglais Cyclic Redundancy Check) est un moyen de contrôle d'intégrité des données puissant et facile à mettre en oeuvre. Il représente la principale méthode de détection d'erreurs utilisée dans les télécommunications.

1.5.1 Principe du contrôle CRC

L'idée du CRC est de transmettre, conjointement aux données que l'on désire envoyer, une valeur calculée par l'émetteur à partir des données à transmettre (on applique un algorithme basé sur la division de polynômes binaires, celui du CRC en question). Ensuite, on envoie les données et le CRC calculé sur la ligne. A la réception, les données sont lues et le récepteur effectue le même calcul de son côté avec le même algorithme. Si le résultat est identique à celui qui a été transmis, alors on est sûr à plus de 99.9% (dépend des algorithmes employés) que les données sont correctes et donc que le mesage n'a pas été altéré. Sinon, il y a eu une ou plusieurs erreurs lors de la transmission.

Le contrôle de redondance cyclique consiste à protéger des blocs de données, appelés trames (frames en anglais). A chaque trame est associé un bloc de données, appelé code de contrôle (parfois CRC par abus de langage ou FCS pour Frame Check Sequence dans le cas d'un code de 32 bits). Le code CRC contient des éléments redondants vis-à-vis de la trame, permettant de détecter les erreurs, mais aussi de les réparer.

Le principe du CRC consiste à traiter les séquences binaires comme des polynômes binaires, c'est-à-dire des polynômes dont les coefficients correspondent à la séquence binaire. Ainsi la séquence binaire 0110101001 peut être représentée sous la forme polynomiale suivante :

$0 * X^9 + 1 * X^8 + 1 * X^7 + 0 * X^6 + 1 * X^5 + 0 * X^4 + 1 * X^3 + 0 * X^2 + 0 * X^1 + 1 * X^0$
soit :
$X^8 + X^7 + X^5 + X^3 + X^0$
ou encore :
$X^8 + X^7 + X^5 + X^3 + 1$

De cette façon, le bit de poids faible de la séquence représente le degré 0 du polynôme, le 4ème bit en partant de la droite représente le degré 3 du polynôme (X^3)... Une séquence de n bits constitue donc un polynôme de degré maximal n-1. Toutes les expressions polynomiales sont manipulées par la suite avec une arithmétique modulo 2.

Dans ce mécanisme de détection d'erreur, un polynôme prédéfini (appelé polynôme générateur et noté G(X)) est connu de l'émetteur et du récepteur. La détection d'erreur consiste pour l'émetteur à effectuer un algorithme sur les bits de la trame afin de générer un CRC, et de transmettre ces deux éléments au récepteur. Il suffit alors au récepteur d'effectuer le même calcul afin de vérifier que le CRC est valide. Le contrôle d'erreur CRC est très précis, si un seul bit est incorrect, la valeur du CRC sera invalide. Le CRC et la somme de contrôle sont tous deux excellents pour découvrir les erreurs aléatoires de transmission, mais offrent peu de protection contre une attaque intentionnelle sur les données.

1.5.2 Algorithme pour le calcul du CRC

Le calcul va utiliser une opération de base : la division des polynômes.

Appelons M(x) le polynôme représentant les données à coder. Son degré est représenté par m.

Chacune des deux extrêmités connaissent un polynôme de degré k , appelé générateur, que l'on nommera G(x).

a) La première étape consiste à multiplier M(x) par Xk. Le résultat P(x) = Xk * M(x) est un polynôme de degré m+k . En fait, cela provoque un décalage de M(x) de rang k vers la droite . Ça permet de garantir que la soustraction entre P(x) et un polynôme de degré k est toujours positive.

b) On effectue la division euclidienne de P(x) par G(x) . Le calcul fournit :

un Quotient Q(x) , et

un Reste R(x) de degré k-1.

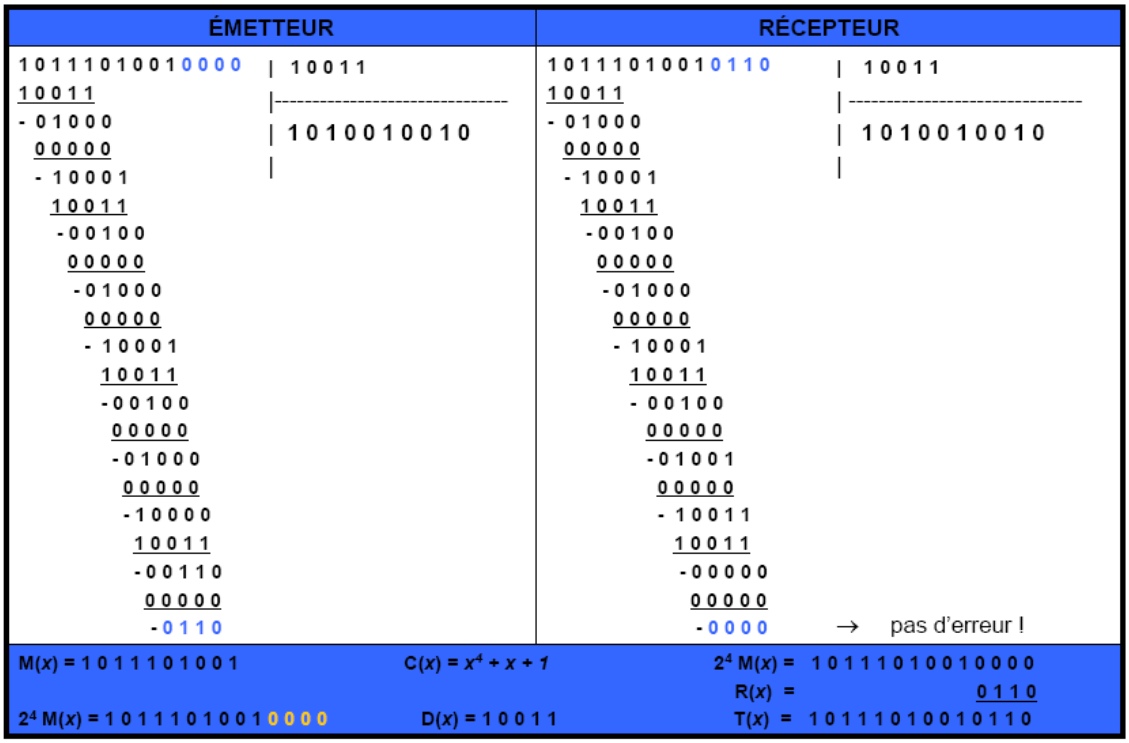
Soit : P(x) = Q(x) * G(x) + R(x)

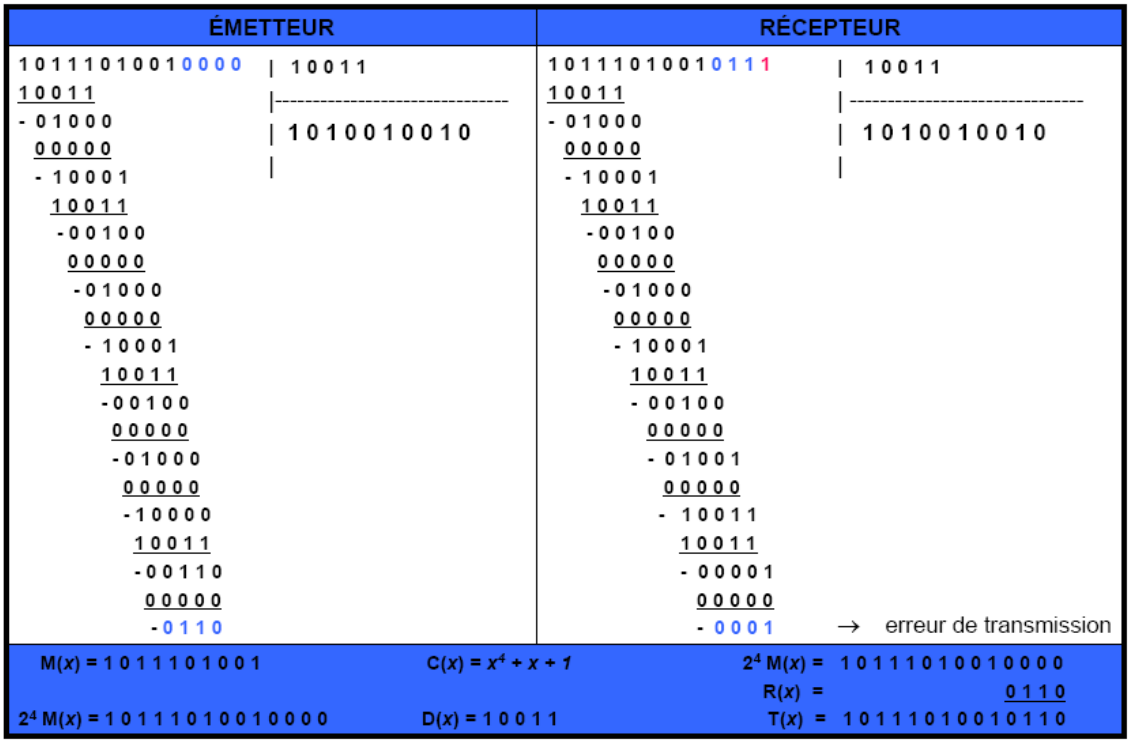
c) Le résultat final est calculé par S(x) = M(x) - R(x) = [Q(x) * G(x) + R(x)] - R(x) <=> S(x) = Q(x) * G(x) .

Donc, si l'on divise S(x) par G(x) , le reste doit être nul, des deux côtés de la connexion, ou alors les valeurs traitées ne sont pas les mêmes.

1.5.3 Un exemple

On veut émettre le message 1011101001, on utilise pour cela un contrôle CRC avec le polynôme $X^4 + X^1 + X^0$
On rajoute 4 Zéros au dividende car le polynôme est de puissance 4.





Taux de détection des erreurs avec un CRC sur 16 bits

Type d'erreur	Taux de détection
Sur 1 bit	100 %
Sur 2 bits	100 %
Nombre impair de bits	100 %
Suite de 16 bits ou moins	100 %
Suite de 17 bits	99,997 %
Suite de 18 bits ou plus	99,998 %

On remarque que le taux de détection avec un contrôle crc sur 16 bits est quasi-parfait, il y a trés peu de chance qu'il ne découvre pas l'(les)erreur(s) produites.

1.5.4 Les polynômes générateurs

Le degré du polynôme est d'autant plus important que la probabilité d'apparition d'une erreur l'est, ou que la longueur du bloc à protéger est importante. Les polynômes générateurs les plus couramment employés pour le calcul du CRC sont :

- CRC-12 : $X^{12} + X^{11} + X^3 + X^2 + X + 1$
  Pour les caractères codés sur 6 bits.

- CRC-16 : $X^{16} + X^{15} + X^2 + 1$

- CRC CCITT V41 : $X^{16} + X^{12} + X^5 + 1$
  (Ce code est notamment utilisé dans la procédure HDLC pour High Level Data Link Control.)

- CRC-32 (Ethernet) : $= X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

- CRC ARPA : $X^{24} + X^{23} + X^{17} + X^{16} + X^{15} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^5 + X^3 + 1$

1.6 Les Codes correcteurs

Les codes correcteurs d'erreurs sont utilisés dans les transmissions de données pour éliminer les effets du bruit. Mais ils nécessitent beaucoup de redondance pour être capable de reconstituer les données originales. Hamming, Reed-Solomon, Goppa, ... Principe : ajouter suffisamment de redondance pour pouvoir récupérer des erreurs n'importe où. Inconvénient : toutes les transmissions sont alourdies(longueur du message et calculs lors de la réception)

1.6.1 Code de Hamming

Erreurla plus simple à détecter : erreur sur un bit isolé. Distance de Hamming permet d'identifier le bit erronné.

Distance de Hamming : (XOR) nombre de bits différents entre 2 mots du code Ex : 10001001 | 10110001 = 3 Pour détecter (à coup sûr) x erreurs il suffit que la distance de Hamming h = x + 1 En effet ainsi s'il y x erreurs on ne pourra pas retomber sur un code existant (différent forcément de x+1 bits)

Un exemple

On veut transmettre le message 1011001.
On insère donc les bits de contrôle aux positions des puissances de 2 comme indiqué sur le schéma.

11	10	9	8	7	6	5	4	3	2	1
1	0	0	x	1	1	0	x	1	x	x

On considére les bits de données égaux à 1, et la représentation en base deux de leurs indices comme suit

$$
\begin{array}{rcl}
11 & = & 1\ 0\ 1\ 1 \\
7 & = & 0\ 1\ 1\ 1 \\
6 & = & 0\ 1\ 1\ 0 \\
3 & = & 0\ 0\ 1\ 1 \\
\hline
 & = & 1\ 0\ 0\ 1
\end{array}
$$

Le code de Hamming ajouté est donc ce résultat : 1001.
Le code transmis est le suivant

11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	1	1	0	0	1	0	1

Le receveur reçoit le message et passe à la vérification :
Il considère les bits de données et de contrôle qui sont égaux à 1 et calcule leur représentation en base 2.

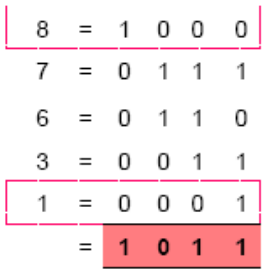
$$
\begin{array}{rcl}
11 & = & 1\ 0\ 1\ 1 \\
8 & = & 1\ 0\ 0\ 0 \\
7 & = & 0\ 1\ 1\ 1 \\
6 & = & 0\ 1\ 1\ 0 \\
3 & = & 0\ 0\ 1\ 1 \\
1 & = & 0\ 0\ 0\ 1 \\
\hline
 & = & 0\ 0\ 0\ 0
\end{array}
$$

La somme de contrôle est égale à 0 –> il n'y a pas eu d'erreur de transmission, le message est valide.

Maintenant voyons ce qu'il se passe si le message est corrompu. Changeons le bit N° 11 en 0 comme ceci :

11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	1	1	0	0	1	0	1

On refait le même calcul que procédemment avec les bits à 1.



Le contrôle d'erreur n'est plus égal à 0 ! –> il y a eu une erreur Hamming nous donne même l'endroit où se situe l'erreur 1011 = 11. L'erreur se situe au bit N° 11 Par contre si le message comporte 2 erreurs, Hamming détectera qu'il y a eu un problème mais ne pourra le corriger ; il ne pourra détecter les erreurs sur plus de 2 bits de transmission(valade seulement pour cet exemple).

Bibliographie

- **C.Servin**
  "Télécoms, de la transmission à l'architecture de réseaux"
  Collection systèmes Distribués, 2^e édition, 1997 (258 pages).

- **H. Hsu**
  "Signaux et communications"
  EdiScience, 2^e édition, 2004 (400 pages).