Guillaume
Bourgoin
M1 Info

4/12/2009

Protocoles Réseaux

Devoir 1

1) Construction de la matrice de contrôle

On a besoin là d'une matrice de 4 lignes (soit m, soit le nombre de bits qu'on ajoutera en codant un mot) et de 15 colonnes (soit $2^m - 1$, soit la longueur d'un mot codé), constituée de 15 vecteurs colonnes distincts, et différents de $\begin{pmatrix}0\\0\\0\\0\end{pmatrix}$ (soit toutes les autres combinaisons de 4 bits).
Classons les, par exemples, par ordre croissant, de $\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$ à $\begin{pmatrix}1\\1\\1\\1\end{pmatrix}$, puis isolons les 4 vecteurs qui forment la sous-matrice identité I_4 à droite pour plus de simplicité dans la suite des calculs.

Notons H la matrice de contrôle ainsi construite :

$$H = \left(\begin{array}{ccccccccccc|cccc} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

P | I_4

Notons P la sous-matrice à gauche de l'identité dans les questions suivantes.

2) La Matrice Génératrice associée

La matrice génératrice correspondante à H est une matrice de 11 lignes (soit $2^m - m - 1$, soit la taille des mots que l'on veut coder) et de 15 colonnes (soit $2^m - 1$, soit la taille des mots codés) constituée de la matrice identité I_{11} (qui générera une copie du mot à coder) et de la transposée de P notée P^T (qui générera les 4 bits spécifiques au mot codé).
Plaçons, par exemple, P^T à la droite de la matrice, de façon à avoir les nouveaux bits de code à la fin des mots codés.

Notons cette matrice G :

$$G = \left(\begin{array}{c|c} I_{11} & P^T \end{array}\right) = \left(\begin{array}{ccccccccccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

3) Le Code d'un mot

Pour obtenir le code d'un mot, on le multiplie par G :

(1) : copie du mot à coder
(2) : terminaison spécifique à ce mot dite "checksum"
(3) : mot codé

$$(0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0)\left(\begin{array}{ccccccccccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right) = \underbrace{0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0}_{(1)}\ \underbrace{0\,1\,0\,1}_{(2)}$$

(3)

4) Les Mots du Code

Le mot 0 1 0 1 0 1 0 1 0 1 0 1 0 0 serait, s'il appartenait à C, le résultat du codage de 0 1 0 1 0 1 0 1 0 1 0 et son checksum serait 1 1 0 0. Or, on sait que dans notre système ce mot a pour checksum 0 1 0 1 (cf partie 3).

Donc 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 n'appartient pas à C.

Autre méthode :

On pourrait également se placer non pas du coté de l'expéditeur (génération) mais du coté du destinataire (contrôle) pour mettre en évidence la présence d'une erreur.

Pour calculer le code d'erreur, dit "syndrome", du message codé, on le multiplie par H^T :

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

... sur le 5ème bit

détection d'une erreur ...

$(0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0)(1\ 0\ 0\ 1)$
$\quad 1\ 2\ 3\ 4\ 5$

Le syndrome est différent de (0000), on détecte donc une erreur, ce qui confirme que le mot n'appartient pas à C.

5) Le Poids du code

Rappelons que le poids de C désigne le nombre de "1" minimum que l'on peut trouver dans un mot de C, excepté dans le mot composé uniquement de 0.

- Cas des mots contenants un seul 1:

  Comme on peut le lire dans les ligne de la matrice G, tout mot de 11 bits contenant exactement un "1" a un code contenant entre 3 et 5 "1" (on peut voir les 11 lignes de la matrice génératrice comme les codes respectifs de ces 11 mots).

Comme on trouve des codes à 3 "1", le poids sera inférieur ou égal à 3.

- Cas des mots contenants 3 "1" ou plus:

  Les "1" de ces mots seront préservés dans les 11 premiers bits de leurs codes, ceux-ci ne pourront donc pas contenir moins de 3 "1", il est donc inutile d'étudier ce cas davantage.

- Cas des mots contenants 2 "1":

  Pour que C ait un poids de 2, il faudrait qu'il existe un mot de 11 bits dont le checksum serait (0000), mais il est facile de voir qu'un tel mot n'existe pas: le checksum d'un mot contenant 2 "1" est la somme de 2 ligne de P^T, or pour que le résultat d'une telle somme soit (0000) il faudrait que les 2 lignes additionnées soit identiques, et toutes les lignes de P^T sont différentes.

On a donc trouvé des mots contenants 3 "1" dans C, et on a démontré qu'il n'y existait aucun mot contenant moins de 3 "1", donc C est de poids 3.

6) Le Nombre de mots dans le code

A chaque mot de 11 bits est associé un unique code appartenant à C (obtenu en le multipliant par la matrice génératrice), et à chaque mot de C est associé un unique mot de 11 bits (les 11 premiers bits du mot du code).

On peut donc dire que les 2 ensemble des mots de 11 bits et de mots de C sont en bijection : il y a donc autant d'éléments dans les 2 ensembles, c'est-à-dire qu'il y a autant de mots dans C que de mots différents écrits sur 11 bits, donc C contient 2^{11} mots.

7) La Proportion du code

Si on prend un mot de 15 bits au hasard on aura 11 bits de "message" et 4 bits de checksum.

Sachant que chaque message n'a qu'un seul checksum associé, et qu'on peut constituer 16 checksum ($16 = 2^4$, car codé sur 4 bits) on a une chance sur 16 pour que le checksum corresponde bien au message, et que le mot de 15 bits appartiennent par conséquent à C.

Autre méthode (moins intuitive et plus mathématique) :

On sait qu'il existe 2^{15} mots de 15 bits.

On sait que parmi ces 2^{15} mots, 2^{11} appartiennent à C (§ partie 6).

Donc la proportion des mots de C parmi les mots de 15 bits est de :

$$\frac{2^{11}}{2^{15}} = \frac{1}{2^4} = \frac{1}{16}$$

8) Un code "parfait"

a) On a dans (m) :

- Les mots identiques à m à une lettre près, qui sont au nombre de 15, puisqu'on peut inverser chacune des 15 lettres de m.
- m lui même

$15 + 1 = 16$

On a donc 16 éléments dans (m).

b) NB: Nous supposerons ici qu'il faut lire "(m)" et non "m" à la fin de la question.

Si m est dans C, peut-on modifier un bit de m tout en le conservant dans C ?

- 1er cas: on modifie un bit du checksum (entre 12 et 15):

Comme expliqué dans la partie 7, un message aura un unique checksum qui lui est associé, donc si on change le checksum sans changer le message le mot n'est plus dans C.

- 2ème cas: on modifie un bit du message (entre 1 et 11):

Modifier un bit du message que l'on va crypter revient à additionner (ou soustraire, mais en modulo 2, c'est la même chose) une ligne de P^T au checksum, or, comme aucune ligne de P^T n'est nulle, cela modifie forcement le checksum.

Donc, si on modifie un bit du message sans modifier le checksum le mot n'est plus dans C.

Remarque : Le raisonnement du 1er cas ne marche pas ici, car plusieurs messages peuvent évidemment avoir le même checksum (puisqu'il y a 2^7 fois plus de messages possibles que de checksums possibles).

Donc non, si m est dans C, aucun autre mot de (m) n'est dans C.

Autre méthode (plus "scolaire"):

On aurait aussi pu s'appuyer sur le théorème qui dit que le nombre d'erreurs E qu'un code peut corriger est tel que $E = \left\lfloor \frac{P-1}{2} \right\rfloor$ où P est le poids du code, or le poids de C est 3 (cf partie 5), et $\left\lfloor \frac{3-1}{2} \right\rfloor = 1$.

Donc C peut toujours corriger une erreur.

Or, si la réponse à la question posée était "oui", on aurait 2 mots semblables à une lettre près reconnu sans erreur, ce qui voudrait dire que le second pourrait être le premier à une erreur près, et que cette erreur ne serait pas detectée, donc le code ne saurait pas toujours détecter une erreur isolée, et il y aurait contradiction avec le théorème.

La réponse à la question est donc "non".

c) On suppose implicite que m et m' sont 2 mots distincts.
La question peut alors se reformuler en "2 mots de C peuvent-ils n'avoir que 1 ou 2 bits de différence", car si ces 2 mots m et m' existaient, on aurait (m) ∩ (m') qui contiendrait au moins un mot (à la fois "m avec la première différence corrigé", et m', ou "m' avec la seconde différence corrigé"), et si (m) et (m') ont un mot en commun, ce mot a au plus une différence avec chacun, et m et m' ont donc au plus 2 différences l'un avec l'autre (soit 1 ou 2, puisqu'on les suppose distincts).

Donc: "Réponse à la question ⟺ Réponse à la question reformulée"

On a déjà montré, dans la partie 8b, que 2 mots de C ne peuvent pas n'avoir qu'un bit d'écart (lorsqu'un mot est dans C, tous les mots à un bit d'écart de celui-ci ne sont pas dans C).
Procédons de façon similaire pour les mots de 2 bits d'écart:

- 1er cas : On modifie 2 bits du checksum :

Pour les même raisons que dans le premier point de la partie 8b, si le mot de départ appartenait à C, le mot modifié n'appartiendra pas à C.

- 2ème cas : On modifie 2 bits du message :

On peut traiter ce cas en combinant ce qui est dit dans le second point de la partie 8b, et dans le troisième point de la partie 5 :

- Si on modifie 2 bits du message, on ajoute 2 lignes de P^T au checksum (cf partie 8b)
- Et si on ajoute 2 lignes de P^T au checksum, comme elles sont différentes l'une de l'autre, elles le modifient forcement (cf partie 5)

Donc modifier 2 bits du message devrait modifier aussi le checksum.
Donc le mot modifié n'appartient pas à C.

- 3ème cas : On modifie 1 bit du message, et 1 bit du checksum :

Comme toutes les lignes de P^T contiennent au moins 2 "1", et puisqu'inverser un bit du message revient à additionner une ligne de P^T avec le checksum, modifier 1 bit du message devrait modifier, non pas 1, mais au moins 2 bits du checksum.
Donc, là encore, le mot modifié n'appartient pas à C.

Par conséquent : non, (m) et (m') n'ont pas d'éléments en commun, c'est-à-dire que 2 mots de C ont au moins 3 bits de différence

Autre méthode :

Comme dans la partie 8b, on pourrait simplement dire que si (m) et (m') avaient un élément en commun, alors cet élément pourrait être m et m' à une erreur près, on ne pourrait donc pas corriger une telle erreur (ne sachant pas s'il faut transformer ce mot en m ou m') et on entrerait encore une fois en contradiction avec le théorème du nombre d'erreurs corrigeables par rapport au poids du code.

d) Cette fois ci, on se contentera de la façon mathématique (l'heure du rendu approche)

- Puisqu'on a 2^{11} éléments dans C (cf partie 6),
- qu'à chaque éléments de C on peut associer un groupe d'éléments de taille 2^4, dont les autres éléments ne sont pas dans C (cf partie 8 ab), et que ces 15 autres éléments sont spécifique à cette élément de C seulement (cf partie 8c)
- et qu'il existe 2^{15} mots binaire de longueur 15,

Alors on peut en déduire que l'union des (m) $\forall m \in C$ est égal à l'ensemble des mots binaires de longueur 15. cqfd

En d'autre termes : si on a 2^{11} "paquets" de 2^4 éléments chacun et que ces paquets sont disjoints, ils remplissent entièrement un sac d'une capacité de 2^{15} éléments.

9) Visualisation des mots hors du code

a) Ecrire m et toutes ses variantes en modifiant, ou non, les bits i, j, k, etc, revient à écrire tous le mots de longueur 4 possibles :

	i	j	k	l
m_1	0	0	0	0
m_2	0	0	0	1
m_3	0	0	1	0
m_4	0	0	1	1
m_5	0	1	0	0
⋮				
m_{16}	1	1	1	1

Remarque : l'un de ces m_x sera égal à m

On a donc 2^4 éléments dans (m, i, j, k, l)

b) NB: On supposera ici que la question sous-entend "$\forall m$"

Pour répondre à cette question on va s'inspirer des raisonnement utilisés dans les points 1 des parties 8b et 8c :
Un message a un et un seul checksum associé, donc pour tout mot de 15 bits, si on autorise les 4 bits de fin à prendre toutes les valeurs possibles, on tombera une et une seul fois sur le checksum correct, et seule cette combinaison donnera un mot de C.
Donc, on va choisir $(i,j,k,l) = (12,13,14,15)$

c) On peut remarquer, en observant P^T, que chaque bit du checksum est la somme de 7 bits particuliers du message (les positions des "1" dans la colonne de P^T qui est au dessus de ce bit) et que les 4 autres bits du message n'ont aucun impact sur cette somme.
Fort de cette remarque, il devient facile de répondre à la question :

Prenons un mot simple de C, par exemple :

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

modifions à présent un bit de somme, le 4ème par exemple :

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

Le mot n'appartient évidemment plus à C.

P^T :

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

On note que ce bit est normalement la somme de bits 5, 6, 7, 8, 9, 10 et 11

Donc cette égalité $(0+0+0+0+0+0+0=1)$ restera fausse quoi qu'on fasse avec les bits 1 à 4 :

$m =$ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
(i, j, k, l)

Donc le mot m ci-dessus et le quadruplet $(i,j,k,l) = (1,2,3,4)$ satisfont la condition : aucun des mots de (m,i,j,k,l) n'est dans C

Remarque : On peut aller encore plus loin et modifier les autres bits de sommes tout en conservant l'égalité fausse

On pourrait donc modifier non pas 4 mais 7 valeurs tout en restant hors de l'ensemble C.

Note : - Tout au long du sujet j'emploie le terme "bit" là où le terme "lettre" serait peut-être plus approprié.