Transmission d'information codée

Comme nous l'avons vu dans un TD précédent, les transmissions d'information codée numériquement sont soumises à des aléas de transmission, dûs aux technologies utilisées ou au contexte. Il n'est pas rare qu'un message envoyé sous forme binaire subissent une, voire même plusieurs modifications sur son trajet entre l'émetteur et le récepteur. Il est donc nécessaire d'imaginer des dispositifs permettant de détecter, ou mieux de corriger, les erreurs de transmissions.

Détection d'erreur : bit de parité

L'utilisation d'un bit de parité est un dispositif permettant de détecter des erreurs. Son principe est décrit ci-dessous.

Un problème de transmission portera sur un ou plusieurs bits. Chacun des bits affectés se verra transformé (il y a deux états possibles) et, si on connait l'état initial du bit, il y a une seule transformation possible, soit passage de 0 à 1, soit passage de 1 à 0.

Pour chaque information envoyée on ajoute un bit calculé de telle sorte que le **nombre total de bit à 1** de l'ensemble (bits de l'information + bit calculé) **soit pair** [1]. On appelle ce bit le *bit de parité*. L'ensemble (information + bit de parité) est expédié. La position du bit de parité est évidement fixée (nous prendrons ici le bit de poids fort du message).

Lors de la réception de l'ensemble, on calcule la parité du message reçu, puis

- si la parité est incorrecte, ceci signifie qu'il y a eu, à coup sûr, une erreur de transmission sur au moins 1 bit (en fait 3, 5, ou 7 bits modifiés auraient le même impact sur la parité). On n'aura pas d'autre recours que de demander un nouvel envoi.
- si la parité est correcte, on considérera que le message est **probablement** correct. Il suffit alors de supprimer le bit de parité pour retrouver l'information supposée correct. Ce n'est qu'une *probabilité*, pas une certitude, parce que deux bits erronés ne modifient pas la parité et il est donc possible, mais improbable, que le message reçu soit incorrect.

Exercice 1 : Émission/Réception avec bit de parité.

Dans cet exercice on considère que l'information est un message composé d'une suite de lettres. Ces lettres sont représentées en binaire, en respectant la convention US-ASCII. Cette convention utilise 7 bits. Le bit de poids fort de chaque octet peut donc faire office de bit de parité. Dans la suite du sujet, on utilise ce codage, restreint aux capitales. La table de correspondance figure ci-dessous :

A	B	C	D	E	F	G	H	I	J	K	L	M
0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48	0x49	0x4a	0x4b	0x4c	0x4d

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0x4e	0x4f	0x50	0x51	0x52	0x53	0x54	0x55	0x56	0x57	0x58	0x59	0x5a

Soit le message *HELLO* à expédier.

Q 1. Donner le codage hexadécimal, puis binaire de chaque caractère à émettre.

Q 2. Déduisez-en la suite d'octets à émettre (chaque octet est obtenu en joignant le bit de parité à son caractère).

Vous recevez la suite d'octet `42 CF 4E CB 4F 56 D2`.

Q 3. Pour chaque caractère, indiquez si il y a eu erreur de transmission et si possible la valeur du caractère émis.

Q 4. Votre correspondant vous assure qu'il a envoyé un mot du dictionnaire. Est-ce possible ?

Code auto-correcteur

Si l'on consent à dépenser un peu plus (en coût de communication), il est possible de retrouver l'original (toujours sur une démarche probabiliste). Il suffit d'utiliser plusieurs bits de parité, chacun opérant sur un sous-ensemble du message binaire initial. En voici une application sur un code de 4 bits utiles (avec 3 bits de parité). Le codage décrit est un exemple de code de Hamming.

1. Autre expression de l'algorithme : si le nombre de bit à 1 du caractère est pair on joint un 0 au message, sinon on joint un 1.

Émission

Les bits de **données** sont notés d_i et les bits de **parités** $\mathbf{p_i}$. Ils sont placés comme suit dans l'octet transmis (le bit de poids le plus fort n'est pas utilisé) :

–	d_3	d_2	d_1	$\mathbf{p_2}$	d_0	$\mathbf{p_1}$	$\mathbf{p_0}$

Les bits de parités sont calculés pour trois sous ensembles décrit par chaque ligne du tableau :

	d_3	d_2	d_1	$\mathbf{p_2}$	d_0	$\mathbf{p_1}$	$\mathbf{p_0}$
contrôlés par $\mathbf{p_0}$	X		X		X		X
contrôlés par $\mathbf{p_1}$	X	X			X	X	
contrôlés par $\mathbf{p_2}$	X	X	X	X			

Les 4 bits utiles sont distribués dans l'octet. Les bits de parité sont calculés. Les sept bits résultant sont émis.

Par exemple :

1. On doit envoyer `1110` = $(d_3 d_2 d_1 d_0)_2$.
2. On complète le tableau suivant qui calcule les bits de parité :

1	1	1	$\mathbf{p_2}$	0	$\mathbf{p_1}$	$\mathbf{p_0}$	
1		1		0		?	$\mathbf{p_0} \leftarrow 0$
1	1			0	?		$\mathbf{p_1} \leftarrow 0$
1	1	1	?				$\mathbf{p_2} \leftarrow 1$

3. L'octet envoyé sera donc :

–	d_3	d_2	d_1	$\mathbf{p_2}$	d_0	$\mathbf{p_1}$	$\mathbf{p_0}$
–	1	1	1	1	0	0	0

4. On envoie donc `0x78`.

Réception

On calcule un **diagnostic** pour chacun des bits de parité (0 si correct, 1 si incorrect) :
- s_0 est un bit de parité pour le sous-ensemble de bit contrôlés normalement par $\mathbf{p_0}$,
- s_1 pour le sous-ensemble contrôlés par $\mathbf{p_1}$,
- s_2 pour le sous-ensemble contrôlés par $\mathbf{p_2}$.

On considère ensuite le *syndrôme* $(s_2 s_1 s_0)_2$:

- si le syndrôme est nul `000`, aucune erreur n'est détectée, on récupère les 4 bits de données,
- si le syndrôme est non nul, donc de `001` à `111`, sa valeur indique le numéro du bit sur lequel porte l'erreur probable. Il suffit d'inverser ce bit pour obtenir l'émission probable.

Par exemple :

1. On reçoit `0x72=01110010`.
2. Le message reçu est `1110`
3. Le calcul du syndrôme donne :

rang	–	7	6	5	4	3	2	1	
	–	d_3	d_2	d_1	$\mathbf{p_2}$	d_0	$\mathbf{p_1}$	$\mathbf{p_0}$	
message reçu	–	1	1	1	0	0	1	0	
$\mathbf{p_0}$	–	1		1		0		0	$s_0 \rightarrow 0$
$\mathbf{p_1}$	–	1	1			0	1		$s_1 \rightarrow 1$
$\mathbf{p_2}$	–	1	1	1	0				$s_2 \rightarrow 1$

4. Le syndrôme est `110` c'est donc le bit 6, c'est-à-dire d_2, qui est erroné.
5. Le message initial était donc probablement `1010`.

<u>Exercice 2</u> :

Q 1. En utilisant ce code, combien d'octets faut-il envoyer pour transmettre le message *HELLO*?

Q 2. Quels sont les octets à envoyer pour transmettre le caractère `H`?

Q 3. En considérant l'exemple précédent, que pouvez-vous déduire de la réception des octets : `0x71`, `0x72`, `0x74`, `0x78` et `0x7c`