Cours Réseaux - Transmission des données

Thierry Vaira

BTS SN-IR

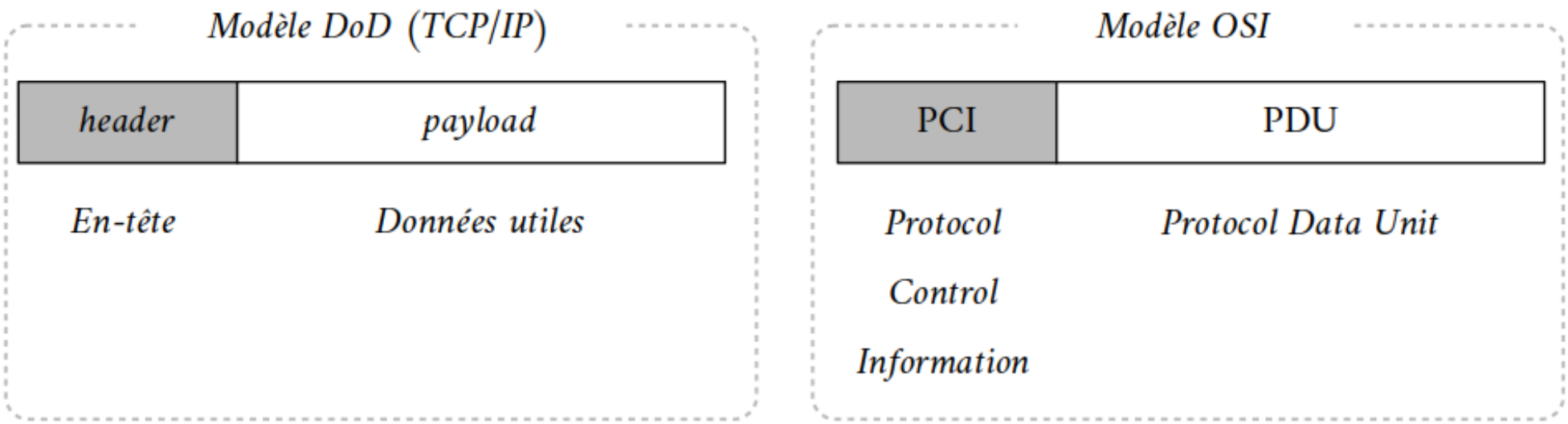
tvaira@free.fr ☽ v0.1

Sommaire

1. Introduction
2. Les protocoles orientés ASCII
3. Les protocoles orientés binaire

Définition

- La charge (***payload***) symbolise les **données utiles** transportées par un **protocole**.



Différents types de réseaux

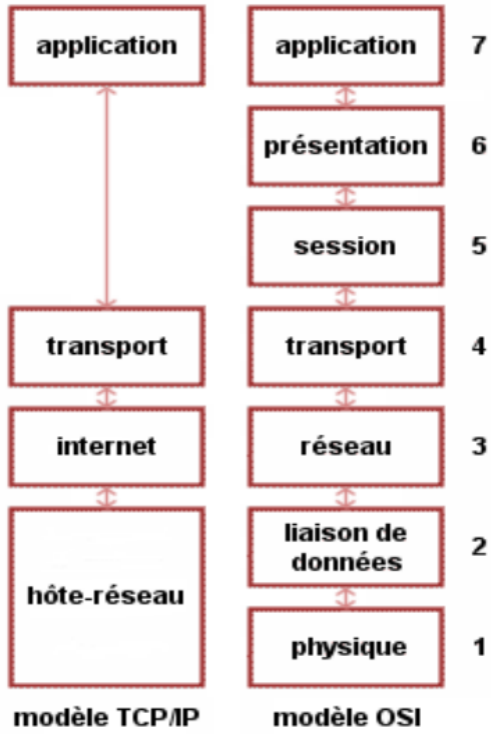
On distingue deux catégories de réseaux :

- **« réseaux informatiques »** : quantité relativement importante de données (essentiellement basés sur les protocoles TCP/IP)
- **« réseaux de terrain »** ou **« bus de terrain »** (ou réseaux locaux industriels) : quantité relativement faible de données (capteurs/actionneurs)

Il existerait plus de 2000 bus de terrain différents ! Les technologies les plus répandues sont : Modbus, Profibus, Interbus-S, ASI, Lonworks et bus CAN.

Réseaux informatiques

Un **réseau informatique** est basé sur un modèle de référence (DoD TCP/IP ou OSI) :



La plupart des protocoles de la couche Application sont orientés ASCII.
Les protocoles des autres couches sont « binaires ».

payload des réseaux TCP/IP

- Couche Application : dépend du protocole de couche Application
- Couche Transport : illimité pour TCP (segmentation) et 65527 octets pour UDP (généralement limité à 8184 octets)
- Couche Réseau : 68 octets minimum à 65527 octets maximum pour IPv4 (fragmentation)
- Couche Interface : 46 octets minimum à 1500 octets maximum pour Ethernet_II

MSS et MTU

- MSS (*Maximum Segment Size*) : taille maximale des données d'un segment TCP pour être transporté dans un paquet seul et non fragmenté
- MTU (*Maximum Transmission Unit*) : taille maximale d'un paquet pouvant être transmis en une seule fois (sans fragmentation) par une interface. MTU pour Ethernet_II : 46 octets minimum à 1500 octets maximum
- Calcul : MSS = MTU - *header* IP - *header* TCP
- Exemple : MSS = 1500 - 20 - 20 = 1460 octets

Réseaux de terrain

Un **réseau de terrain** est basé le plus souvent sur la **restriction du modèle OSI à 2 ou 3 couches** :

- la couche **Application** (qui peut être vide dans de nombreux réseaux)
- la couche **Liaison** qui doit assurer un transport fiable de quantité assez faible de données mais en respectant des contraintes "temps réel" (déterminisme) = niveau trame
- la couche **Physique** qui doit respecter des contraintes fortes liées à l'environnement (température, vibrations, ...) = niveau bit

payload des réseaux de terrain

- AS-i : 4 bits
- Bus CAN : 0 à 8 octets
- Sigfox : 12 octets (140 messages montants maximum par jour)
- Profibus : 1 à 246 octets
- ModBus : 253 octets

Les protocoles orientés ASCII

- Les protocoles orientés ASCII transmettent les octets sous la forme de caractères (encodés dans le jeu standard le plus souvent).
- Cette technique peut être utilisée sur des réseaux informatiques (les protocoles HTTP, FTP, ... de la couche Application) ou des réseaux de terrain (protocole NMEA183 ou commandes AT par exemple).

Le délimiteur de fin est souvent `\r\n` (pour les protocoles de la couche Application) sinon `\r` ou `\n` ou aucun.

Cas des réseaux informatiques

- Avantage : les données sont transmises « octet par octet » ce qui permet de s'affranchir du **problème de l'*endianness*** sinon il faudrait utiliser un protocole spécifique pour transmettre des données numériques (cf. protocole XDR (*eXternal Data Representation*))
- Inconvénient : cela génère **plus de d'octets** mais la taille des données utiles (*payload*) et les débits sont relativement importants dans les réseaux informatiques

Exemple : la valeur 1789

- en binaire codée sur 16 bits (2 octets) : `0x06FD` en *big endian* ou `0xFD06` en *little endian*
- en ASCII codée sur 4 octets : '1' (`0x31`) '7' (0x37) '8' (`0x38`) '9' (`0x39`)

Pour les protocoles des autres couches, TCP/IP définit le standard *network byte order* qui est le *big endian*.

Cas des réseaux de terrain

- Avantage : la simplicité de mise en œuvre car les données sont transmises « octet par octet » sur une liaison série RS232, RS422 ou RS485
- Inconvénient : cela génère plus de d'octets mais la quantité de données reste relativement faible dans les réseaux de terrain

Exemple : la trame `"AT\r\n"` = 4x8 = 32 bits pour 40 bits transmis

- START + 'A' (0x41) [+ PARITE] + STOP = 10 bits (minimum)
- START + 'T' (0x54) [+ PARITE] + STOP = 10 bits (minimum)
- START + '\r' (0x13) [+ PARITE] + STOP = 10 bits (minimum)
- START + '\n' (0x10) [+ PARITE] + STOP = 10 bits (minimum)

Exemple : le protocole HTTP I

- Le protocole HTTP (*HyperText Transfert Protocol*) sert notamment au dialogue entre un client web (navigateur par exemple) et un serveur web (apache ou IIS par exemple). C'est un protocole de la couche Application.
- HTTP est un protocole orienté texte (ASCII), basé sur TCP. Il existe deux spécifications la 1.0 (RFC 1945) et la 1.1 (RFC 2616). Il est utilisé pour la transmission de documents distribués et multimédia. Les messages HTTP sont basés sur un système de requête/réponse.
- Exemple de requête :

```
GET /index.html HTTP/1.1\r\n
Host: tvaira.free.fr\r\n
\r\n
```

Exemple : le protocole HTTP II

- Exemple de réponse :

```
HTTP/1.1 200 OK\r\n
Content-Length: 215\r\n
Content-Type: text/html\r\n
\r\n
<html><body><h1>It works!</h1></body></html>
```

L'encodage base64 I

- Lorsque l'on veut représenter des données binaires (une image, un exécutable) dans un document texte (page web, courriel, ...) la transcription hexadécimale en ASCII des octets multiplierait la taille par deux. L'utilisation d'un encodage permet de limiter cette augmentation.
- L'encodage base64 est un codage de données utilisant 64 caractères (en fait 65 avec le caractère de complément) pour permettre la représentation de 6 bits par un caractère.
- Ce processus de codage consiste à coder chaque groupe de 24 bits successifs de données par une chaîne de 4 caractères. Si moins de 24 bits, on ajoute des « = » complémentaires pour former quand même 4 caractères.

L'encodage base64 II

- Chaque groupe de 6 bits est utilisée comme index dans la table de codage.

Valeur	Codage	Valeur	Codage	Valeur	Codage	Valeur	Codage
0	000000 A	17	010001 R	34	100010 i	51	110011 z
1	000001 B	18	010010 S	35	100011 j	52	110100 0
2	000010 C	19	010011 T	36	100100 k	53	110101 1
3	000011 D	20	010100 U	37	100101 l	54	110110 2
4	000100 E	21	010101 V	38	100110 m	55	110111 3
5	000101 F	22	010110 W	39	100111 n	56	111000 4
6	000110 G	23	010111 X	40	101000 o	57	111001 5
7	000111 H	24	011000 Y	41	101001 p	58	111010 6
8	001000 I	25	011001 Z	42	101010 q	59	111011 7
9	001001 J	26	011010 a	43	101011 r	60	111100 8
10	001010 K	27	011011 b	44	101100 s	61	111101 9
11	001011 L	28	011100 c	45	101101 t	62	111110 +
12	001100 M	29	011101 d	46	101110 u	63	111111 /
13	001101 N	30	011110 e	47	101111 v		
14	001110 O	31	011111 f	48	110000 w	(complément)	=
15	001111 P	32	100000 g	49	110001 x		
16	010000 Q	33	100001 h	50	110010 y		

- Exemple : Soit la donnée 38 (0x26)
  - 0x26 codé en binaire : 00100110
  - ajoute des bits 0 pour obtenir un multiple de 6 bits : 00100110 0000
  - on forme des groupes de 6 bits : 001001 et 100000
  - en les prenant comme index dans la table de codage, on obtient : J et g
  - on ajoute le caractère de complément pour former 4 caractères : Jg==

Exemple : la norme NMEA 0183 I

- La norme NMEA 0183 est une spécification pour la communication entre équipements marins, dont les équipements **GPS**. Elle est définie et contrôlée par la *National Marine Electronics Association* (NMEA), association américaine de fabricants d'appareils électroniques maritimes
- La norme NMEA 0183 utilise une simple communication série pour transmettre une "phrase" à un ou plusieurs écoutants. Une trame NMEA utilise tous les caractères ASCII.
- Exemple de trame :
  `$GPGGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000*0E\r\n`
  - `$` : délimiteur de début de trame
  - `GP` : identifiant du « parleur » (*talker id* ici `GP` pour GPS)
  - `GGA` : type de trame (ici *Global Positioning System Fixed Data*)

Exemple : la norme NMEA 0183 II

- `064036.289,4836.5375,N,00740.9373,E ...` : données séparées par des ',' de la trame GGA (contient notamment la longitude et la latitude)
- `*` : délimiteur de *checksum*
- `0E` : valeur du *checksum* (pour la détection d'erreurs de transmission)
- `\r\n` : délimiteur de fin de trame

Exemple : commandes Hayes ou commandes AT

- À l'origine, la firme Hayes, fabricant de modems, a développé un protocole pour la commande d'un modem externe à partir d'un ordinateur. Les commandes AT (ou Commandes Hayes) sont des commandes que l'on peut directement envoyer au modem, lorsque celui-ci est en mode Command.
- Chaque commande est envoyée sous la forme d'une ligne de texte encodée en ASCII commençant par `"AT"`.
- Les commandes AT sont maintenant utilisées pour communiquer avec de nombreux périphériques (modules Xbee, Bluetooth, etc ...).
- Exemple de trame pour un module ESP8266 WiFi :

```
AT+CIPSTAMAC?
+CIPSTAMAC:"ec:fa:bc:14:40:8d"
```

Programmation

- Les types `string` des différents langages de programmation fournissent généralement un ensemble de fonctions de traitement de chaînes de caractères.
  - Par exemple : `substr()`, `split()`, `section()`, `mid()`, etc ...
- Des fonctions de conversion de valeurs numériques (`int`, `float`, `double`) en chaînes de caractères (et inversement) sont aussi nécessaires.
  - Par exemple : `sprintf()`, `sscanf()`, `atoi()`, `atol()`, `atof()`, `stringstream`, etc ...

La taille relativement faible (quelques octets) des données utiles (*payload*) oblige à utiliser des protocoles « binaires » dans lesquels on met souvent en œuvre des techniques d'optimisation de taille.
Par exemple :

- données brutes issues des convertisseurs analogiques-numériques (CAN)
- tronquage des valeurs (valeur entière, *offset* (décalage), coefficient, ...)
- conversion des `float` (4 octets) et des `double` (8 octets)

Le protocole devra préciser l'ordre des octets (*endianness*) en indiquant la position du MSB (*Most Significant Bit* ou *Most Significant Byte*) et du LSB (*Least Significant Bit* ou *Least Significant Byte*).

Exemple n°1 : pression atmosphérique

On désire transmettre la pression atmosphérique issue d'un capteur en un minimum de bits. Les valeurs records mesurées sont : 870 hPa (min) et 1 086,8 hPa (max).

- Il faudrait **32 bits** pour transmettre des valeurs de pression atmosphérique sous la forme d'un `float`.
- Il faudrait **11 bits** au maximum pour transmettre des valeurs entières de pression atmosphérique comprises entre 0 et 2047 hPa.
- Les valeurs varient entre 870 hPa et 1 087 hPa soit 217 hPa donc il suffit de **8 bits** pour coder une valeur entière de pression atmosphérique (on utilisera un **décalage de 850** par exemple) : de 850 à 1105 hPa.

Exemple n°2 : température

On désire transmettre des mesures de température issues d'un capteur en un minimum de bits.

- Il faudrait **32 bits** pour transmettre des valeurs de température sous la forme d'un `float`.
- Il faudrait **16 bits** pour transmettre des valeurs entières signées de température comprises entre -32768 et +32767 °Celsius.
- Il faudrait **16 bits** pour transmettre des valeurs entières signées de température **en centièmes de dégrés** comprises entre -327,68 et +327,67 °Celsius : on multiplie par 100 et on conserve la valeur entière puis on divise par 100 à la réception.
- Il faudrait **10 bits** pour transmettre des valeurs brutes d'un CAN 10 bits etc ...
- *Remarque :* en utilisant le degré Kelvin, on pourra transmettre des valeurs de températures non signées strictement positives.

Programmation

Pour manipuler des bits, on utilisera en C/C++ :

- opérateurs de masque : ET bit à bit &, OU bit à bit |
- opérateurs de décalage binaire :
  - décalage de *n* bits à gauche : «n
  - décalage de *n* bits à droite : »n
- opérateur d'inversion binaire : ~

Exemple n°1 : Bus CAN I

Les mesures effectuées (U_{in}, I_{in}, U_{out} et $T_{ambiante}$) sont numérisées puis encapsulées dans une trame de données CAN.

Bits	7	6	5	4	3	2	1	0
Byte1	BVLR	OVT	NOC	UNDV	X	X	MSB U_{IN}	
Byte2	LSB U_{IN}							
Byte3	X	X	X	X	X	X	MSB I_{IN}	
Byte4	LSB I_{IN}							
Byte5	X	X	X	X	X	X	MSB U_{OUT}	
Byte6	LSB U_{OUT}							
Byte7	$T_{AMBIANTE}$							

Exemple n°1 : Bus CAN II

Extraction :

```
unsigned char pMesures[8];

unsigned short uInMSB = pMesures[0] & 0x03;
unsigned short uInLSB = pMesures[1];
unsigned short uIn = (uInMSB << 8) | uInLSB;

uInMSB = pMesures[2] & 0x03;
uInLSB = pMesures[3];
uIn = (uInMSB << 8) | uInLSB;

uInMSB = pMesures[4] & 0x03;
uInLSB = pMesures[5];
uIn = (uInMSB << 8) | uInLSB;

double temperature = (double)pMesures[6];
```

Exemple n°1 : Bus CAN III

ÉCHELLES DE MESURES ET DE CONVERSIONS ANALOGIQUES / NUMÉRIQUES

	U_{IN}		I_{IN}		U_{OUT}		$T_{AMBIANTE}$	
	Valeur codée sur 10 bits	Tension correspondante	Valeur codée sur 10 bits	Intensité correspondante	Valeur codée sur 10 bits	Tension correspondante	Valeur codée sur 8 bits	Température correspondante
min	0	0 V	0	0 A	0	0 V	- 128	-128°C
max	1023	28 V	1023	150 A	1023	28 V	127	127°C

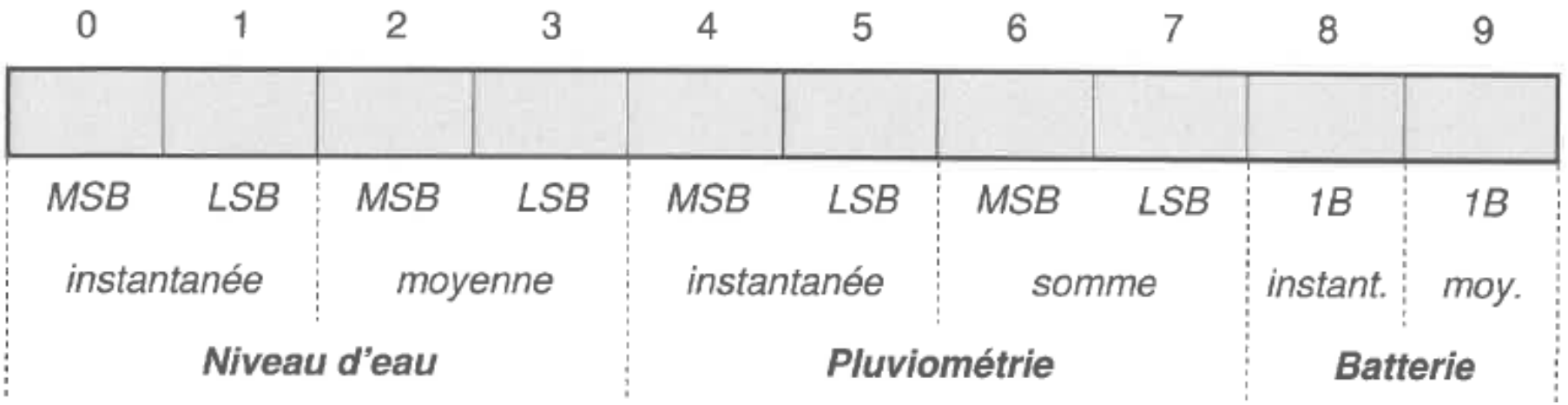
Décodage :

```
// 1023 est la valeur max possible sur 10 bits
// 28V est la tension max
double uPan = ((double)uIn / 1023)*28 ;
// 150A est l'intensité max
double iPan = ((double)uIn / 1023) * 150;
// 28V est la tension max
double uBat = ((double)uIn / 1023) * 28;

double temperature = (double)pMesures[6];
```

Exemple n°2 : Trame Sigfox I

La trame Sigfox contient 10 octets de données utiles. Ces données (mesures fournies par les capteurs) sont contenues dans un tableau `unsigned char message[10]`. Elles sont organisées de la façon suivante :



Exemple n°2 : Trame Sigfox II

Les valeurs de niveau d'eau sont exprimées en cm (valeurs entières de 0 à 999). Les valeurs de pluviométrie sont exprimées en mm (valeurs entières de 0 à 999). Les valeurs de tension de la batterie sont exprimées en dixièmes de volt.

Remarque : pour la conversion en valeurs entières, les valeurs sont tronquées et ne sont pas arrondies.

Exemple de fabrication pour le niveau d'eau :

```
unsigned char message[10];

// conversion des niveaux d'eau en cm
unsigned short niveauEauCm = (unsigned short)(niveauEau * 100);

// ajout des valeurs de niveau d'eau
// dans le tableau message[]
message[0] = (niveauEauCm & 0xFF00) >> 8; // MSB niveau eau instantané
// ou : message[0] = niveauEauCm >> 8;
message[1] = niveauEauCm & 0x00FF; // LSB niveau eau instantané
// ou : message[1] = niveauEauCm; // LSB niveau eau instantané
```