

Réseaux

Architectures et Protocoles

Gilles Grimaud – USTL
www.lifl.fr/~grimaud

Format d'enseignement

13 semaines pour :

1h30 de cours par semaine
1h30 de TD & TP par semaine

Objectifs du cours

- Objectifs

Acquérir des notions sur :

- les supports matériels des réseaux
- le fonctionnement des matériels
- le rôle et la réalisation des logiciels de base

Maîtriser la programmation :

- des logiciels de base d'un réseau
- de clients et de serveurs TCP/UDP

Tour d'horizon

- Définir et classer les réseaux numériques
- Les supports matériels de la communication
- La notion de réseau
- La notion d'assemblage, de connexion
- Les bases de l'exploitation applicative
- Conclusion : le modèle OSI

Introduction aux Réseaux

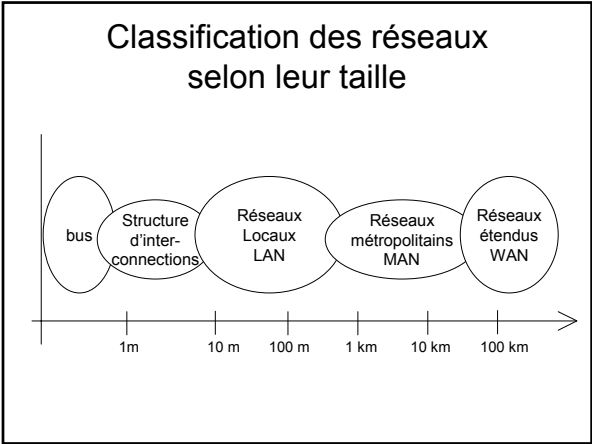
Un réseau numérique est constitué d'un ensemble d'ordinateurs connectés entre eux par des liaisons physiques.

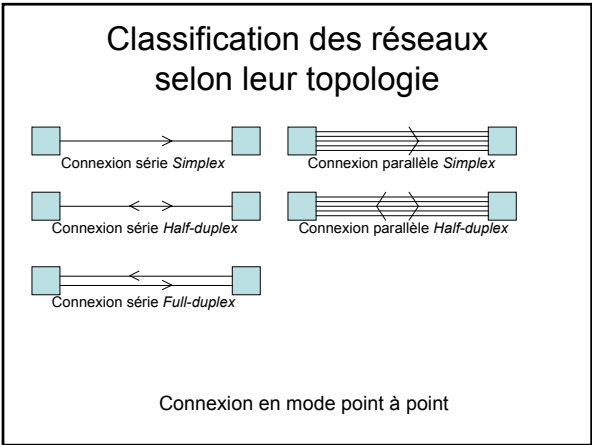
Un réseau numérique permet l'échange entre machines distantes de données qui sont si nécessaire relayées de liaison en liaison par les machines intermédiaires.

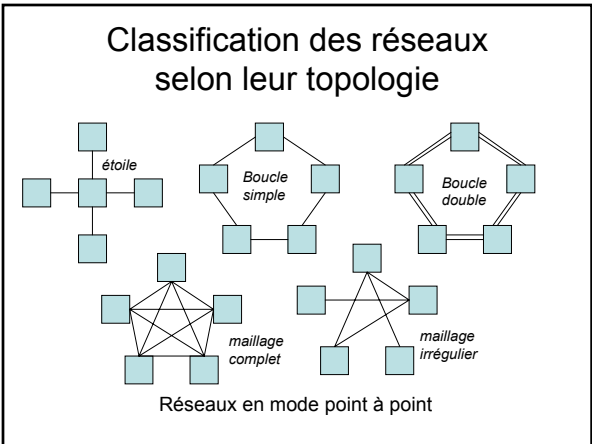
Echanger des informations numériques

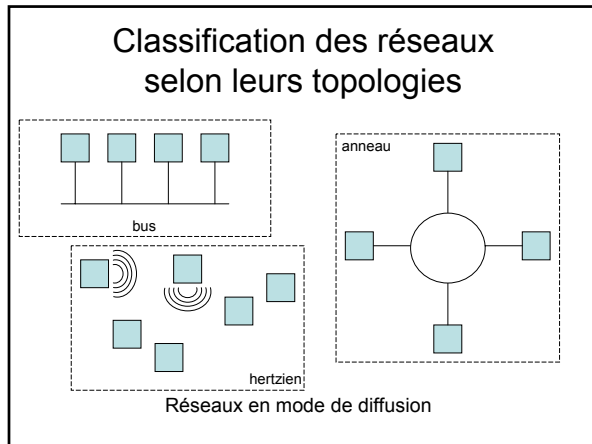
Deux modes de fonctionnement d'un réseau :

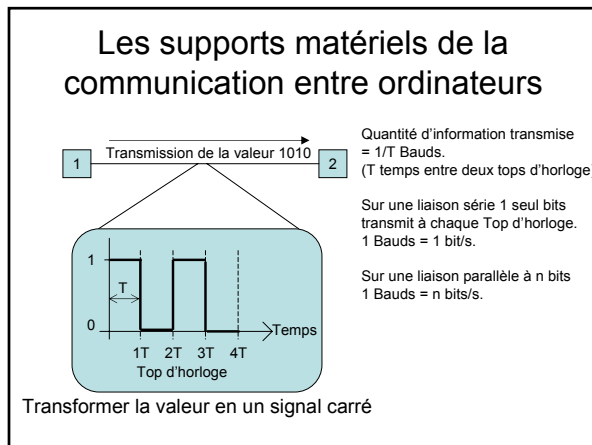
- **avec connexion**
une machine établit une connexion avec une autre ;
ensuite elles échangent des données ;
finalement elles terminent la connexion.
⇒ communication sur le modèle du téléphone.
- **sans connexion**
une machine envoie un message (appelé *datagramme*) ;
le réseau achemine le *datagramme* jusqu'au destinataire ;
Le *datagramme* est stocké dans une « boîte au lettre » ;
Le destinataire récupère le message lorsqu'il le souhaite.
⇒ communication sur le modèle du courrier postal.

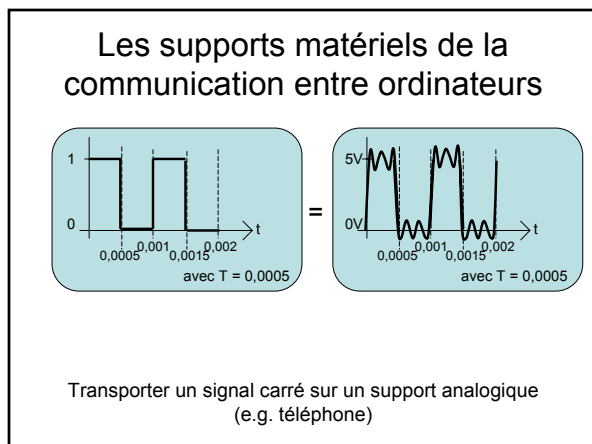




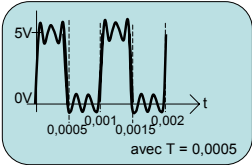
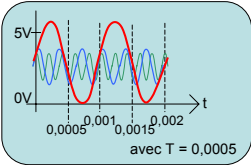






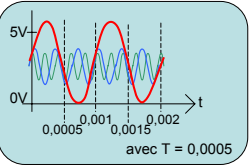


Les supports matériels de la communication entre ordinateurs

 $=$ 

un signal analogique est une somme (limitée) d'harmoniques.

Les supports matériels de la communication entre ordinateurs

 $= g(t) =$

$\frac{1}{2} +$ $\frac{2}{\pi} \sin (2000 \pi t) +$ $\frac{2}{3 \pi} \sin (6000 \pi t) +$ $\frac{2}{5 \pi} \sin (10000 \pi t) +$ \dots

Chaque harmonique correspond à un signal sinusoïdal donné.

Les supports matériels de la communication entre ordinateurs

$\frac{1}{2} +$ $\frac{2}{\pi} \sin (2000 \pi t) +$ $\frac{2}{3 \pi} \sin (6000 \pi t) +$ $\frac{2}{5 \pi} \sin (10000 \pi t) +$ \dots

 $= g(t) =$

$\frac{c}{2} +$ $\sum_{n=1}^{\infty} a_n \sin (2 \pi n f t) +$ $\sum_{n=1}^{\infty} b_n \cos (2 \pi n f t)$

Avec $f = 1 / T$ (fréquence fondamentale)
 $c = 2 / T \int_0^T g(t) d t$
 $a_n = 2 / T \int_0^T g(t) \sin (2 \pi n f t) d t$
 $b_n = 2 / T \int_0^T g(t) \cos (2 \pi n f t) d t$

Dont la somme infinie constitue une **série de Fourier**.

Notion de liaison

La liaison entre deux ordinateurs nécessite des procédures d'établissement, de maintien et de libération des transmissions de données sur le support physique. Ces procédures détectent et corrigent, si possible, les erreurs dues au support physique de communication.

Notion de liaison

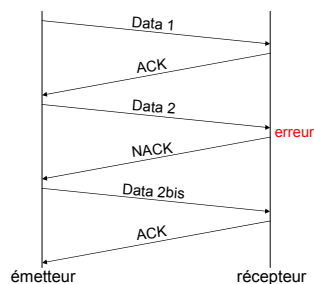
Convenir d'un codage détecteur d'erreur pour les données transmises.

L'exemple du bit de parité :

0	01000001	0..0..1..0..0..0..0..1	0	01000001
1	01110000	1..0..1..1..1..0..0..0..0	1	01110000
1	01100001	1..0..1..1..0..0..0..0..1	1	01100001
1	01000110	1..0..1..0..0..0..0..1..0	1	01000010

Notion de liaison

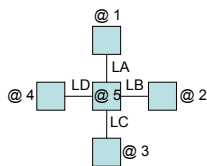
Dialogue de liaison de type *send and wait*



Notion de réseau

Un réseau est constitué de différentes liaisons entre ordinateurs. La gestion d'un réseau nécessite l'existence de mécanismes d'**adressage** des différentes machines, de **routage** et de **contrôle de flux** des paquets de données transportés sur chaque liaison.

Adressage



Routage

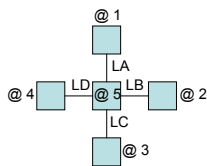


Table @3	
@ des	Liaison
@ 1-5	LC

Table @5	
@ des	Liaison
@ 1	LA
@ 2	LB
@ 3	LC
@ 4	LD

Routage

@ des	Liaison
@ 1-2;4-5	LC
@ 6-10	LE

@ des	Liaison
@ 1	LA
@ 2	LB
@ 3;6-10	LC
@ 4	LD

@ des	Liaison
@ 1-4	LE
@ 6-7;9-10	LBus

Routage

Gestion via un routage centralisé :

- **Fixe** : pas de mise à jour. Tables fixées une fois pour toute en *fcn* de la topologie du réseau.
- **Synchrone** : Tables mises à jour au même moment par un centre de contrôle. (à partir d'informations reçues dynamiquement).
- **Asynchrone** : tables mises à jour indépendamment les unes des autres dans certaines parties du réseau (avec émission d'un compte-rendu de son état au centre de contrôle).

Routage décentralisé
inondation, *hot potatoes*, routage adaptatif.

@ des	Liaison
@ 1-2;4-5	LC
@ 6-10	LE

@ des	Liaison
@ 1	LA
@ 2	LB
@ 3;6-10	LC
@ 4	LD

@ des	Liaison
@ 1-4	LE
@ 6-7;9-10	LBus

Contrôle de flux

Le contrôle de flux a pour **objectif** :

- minimiser le temps de transfert des paquets ;
- éviter la congestion du réseau ;

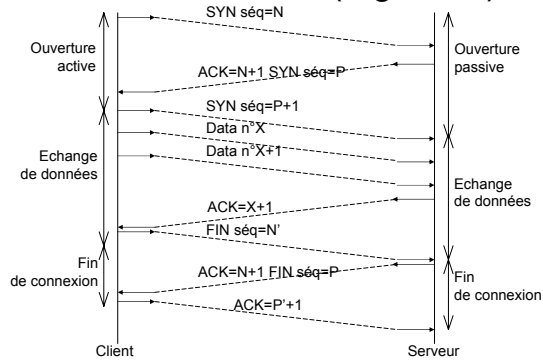
Techniques :

Contrôle par crédits & crédits dédiés. Contrôle par fenêtre.

Notion de connexion

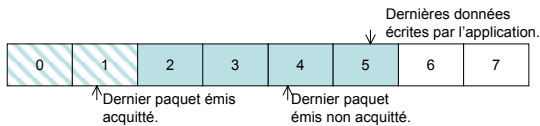
Les applications s'échangent en règle générale des données de taille et de contenu variés. Une connexion assure aux applications la capacité de transférer des séquences de données. Pour cela elle fragmente (défragmente) ces données en paquets autonomes qui sont émis sur (reçus depuis) le réseau. Elle assure la fiabilité des données en gérant la perte de paquets ou leur corruption.

Mode connecté (e.g. TCP)

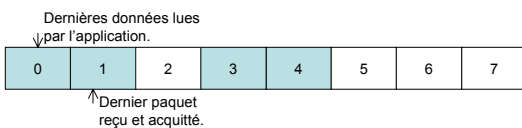


Connexion TCP

Gestion d'une fenêtre d'émission :



Gestion d'une fenêtre de réception :



Bases d'exploitation logicielle

Pour une application, le réseau apparaît comme un support sur lequel il est possible d'initier l'émission ou d'attendre la réception de données à destination, ou en provenance de n'importe quelle autre application.

Base d'exploitation logicielle

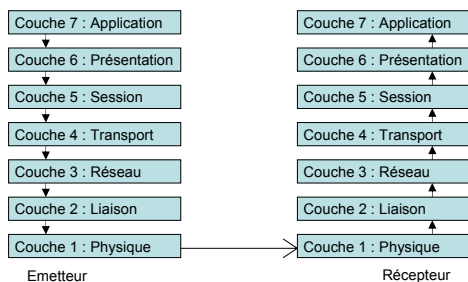
Coté Client :

```
Socket s;  
InputStream iS;  
int t;  
...  
s =  
  new socket("134.206.11.6",765);  
iS = s.getInputStream();  
...  
t=iS.read();  
... // t == 43  
s.close();
```

Coté Serveur :

```
ServerSocket serv;  
Socket s;  
OutputStream oS;  
...  
serv = new Serversocket(765);  
...  
s = serv.accept();  
...  
oS = s.getOutputStream();  
oS.write(43);  
...  
s.close();
```

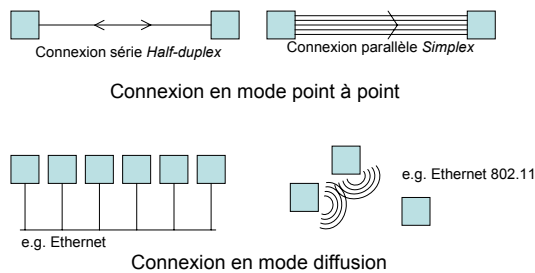
Conclusion : Le modèle OSI



Couche 1 : Physique

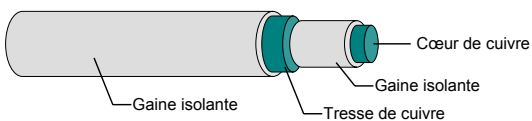
Transport physique de
l'information

Support physique de communication



Support de transmission

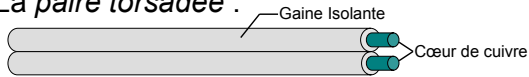
- Le *câble coaxial* :



Jusqu'à 150MHz en large bande (fiabilité 1/10⁷).
Support encombrant. Télévision et téléphone.
Version 10 Base 2 (10MHz sur 200m)
Version 100 Base 5 (100MHz sur 500m)
Connecté au poste avec un BNC (Ethernet fin)

Support de transmission

- La *paire torsadée* :



Origine : téléphone (prise RJ45).

56kbit/s avec les modems récents (fiabilité $1/10^5$).

10 (voir 100) Mbits/s (sur quelques mètres).

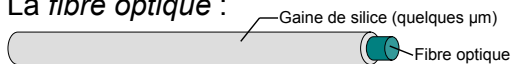
Utilisée dans les réseaux 10 Base T

(étoile en mode diffusion ou point à point).

Evolution vers 100 Base T voir « Gigabit ».

Support de transmission

- La *fibre optique* :



Support de transmission récent.

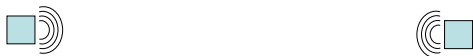
Supporte le transport de plusieurs GBits/s sur de très longues distances (fiabilité $1/10^{12}$).

Faible sensibilité électromagnétique & difficultés d'écoute.

Emetteur diode Electroluminescente (LED) ou diode Laser. Récepteur photosensible.

Support de transmission

- *Sans fil* :

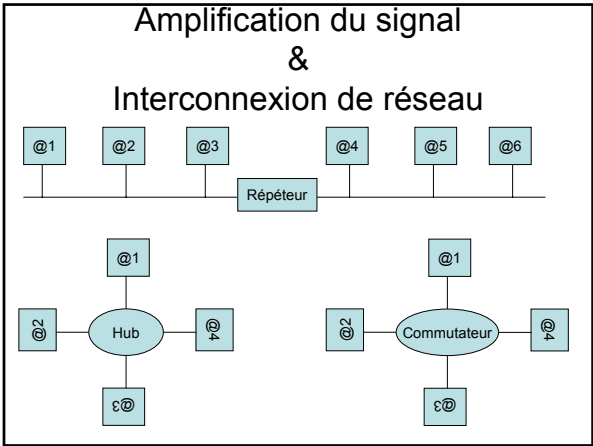


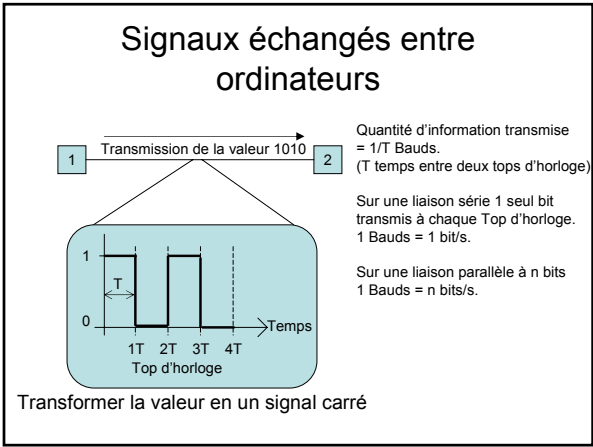
Différents types : infrarouge, hertzien (2.4GHz)

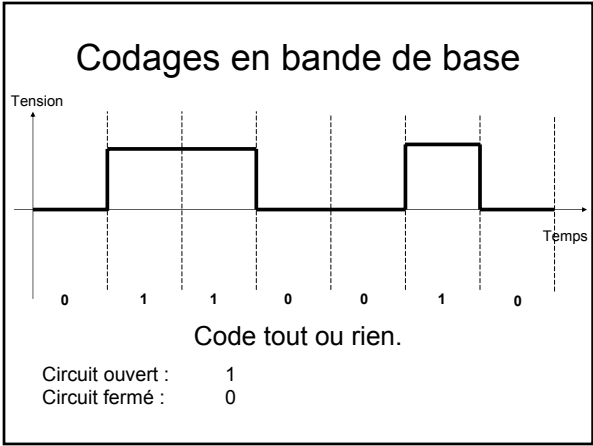
Débit : 11 MBits/s

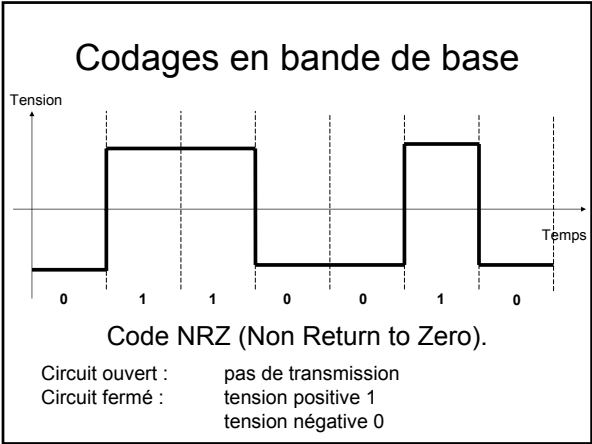
Portée moyenne : 10m à 150m

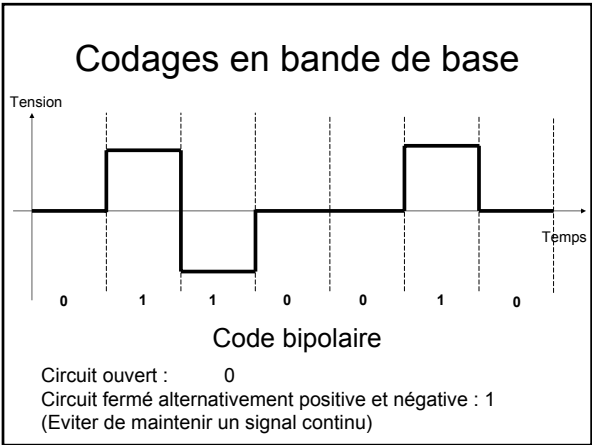
Forte sensibilité aux perturbations électromagnétiques. Pas de sécurité physique.

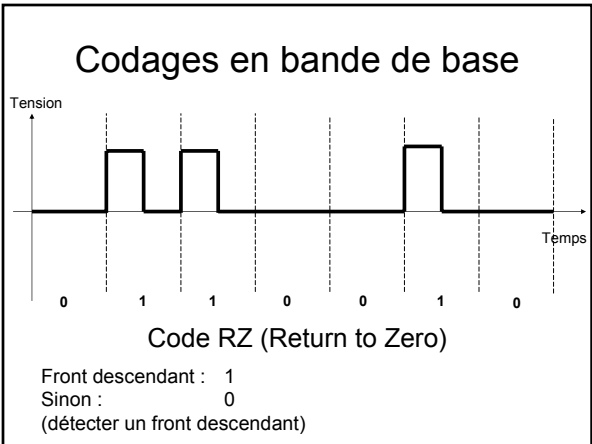


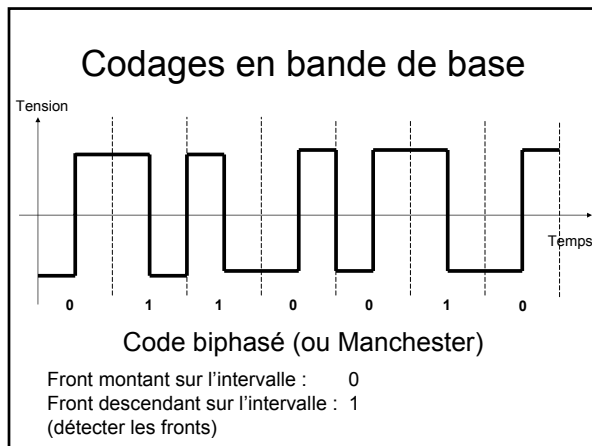


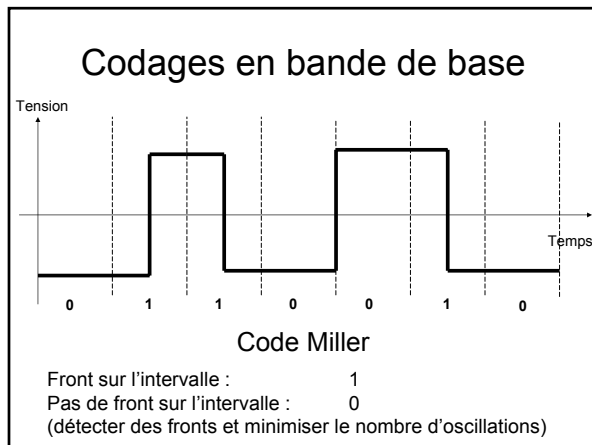










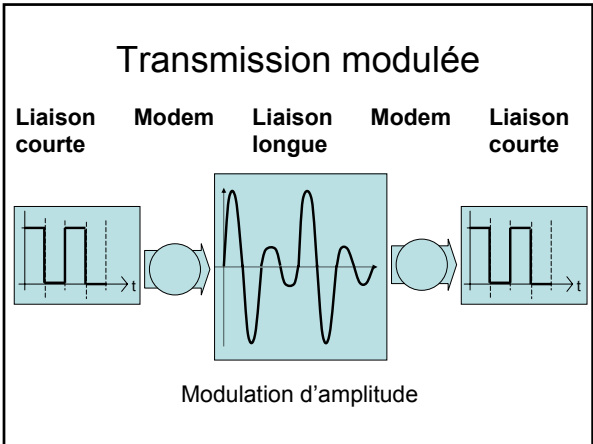


Transmission modulée

Problème de la transmission en bande de base :
dégradation du signal.

Usage limité au **réseau local**.

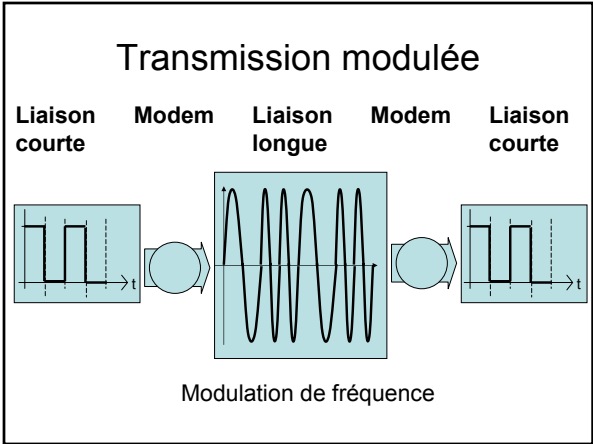
⇒ **Utilisation d'un modem** (modulateur - démodulateur)
Convertisseur bande de base en :
⇒ **Modulation d'amplitude**
⇒ **Modulation de fréquence**
⇒ **Modulation de phase**
et réciproquement ...



Modulation d'amplitude

Modulation de l'amplitude d'un signal sinusoïdal.

Pour	Contre
<ul style="list-style-type: none">• Transporter un signal alternatif est moins coûteux (moins de perte).• La modulation d'amplitude est un circuit électrique simple (premier utilisé).	<ul style="list-style-type: none">• Sensible à la perturbation du signal (orage, lignes électriques...).

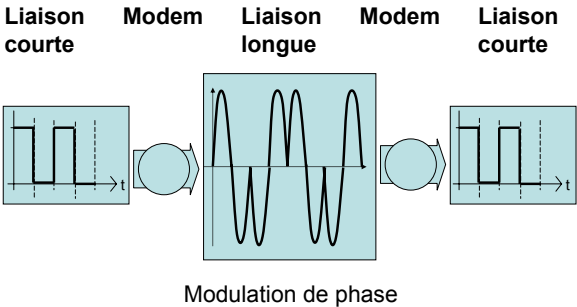


Modulation de fréquence

Modulation de la fréquence d'un signal sinusoïdal.

Pour	Contre
<ul style="list-style-type: none">• Transporter un signal alternatif est moins coûteux (moins de perte).• La modulation de fréquence est résistante aux perturbations (d'amplitude).	<ul style="list-style-type: none">• Système de démodulation moins trivial à concevoir. (la FM a vue le jour après la AM).

Transmission modulée



Modulation de phase

Modulation de la phase d'un signal sinusoïdal.

Pour	Contre
<ul style="list-style-type: none">• Les dispositifs de (dé)modulation de phase permettent de coder facilement plus de deux états.• La modulation de phase est résistante aux perturbations (d'amplitude).	<ul style="list-style-type: none">• Système de démodulation non trivial.

Transmission modulée

Les transmissions modulées peuvent **combiner plusieurs** formes de **modulations** simultanées.

Exemple :

1 niveau de modulation d'amplitude +

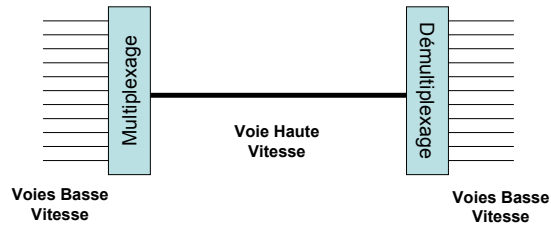
1 niveau de modulation de fréquence

Permet de coder [0|1] en AM et [0|1] en FM.

Donc un temps d'horloge permet de coder 4 valeurs (00, 01, 10, 11) sur 2 bits :

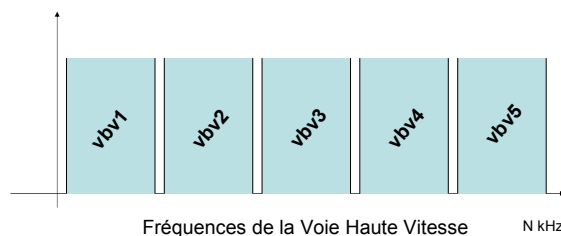
Dans ce cas 1 Baud = 2 bits/s .

Multiplexage

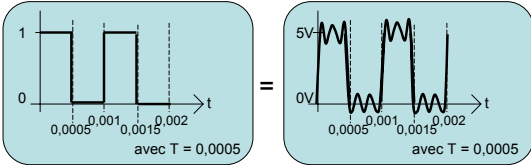


- Multiplexage *fréquentiel* ;
- Multiplexage *temporel* ;
- Multiplexage *statistique*.

Multiplexage fréquentiel

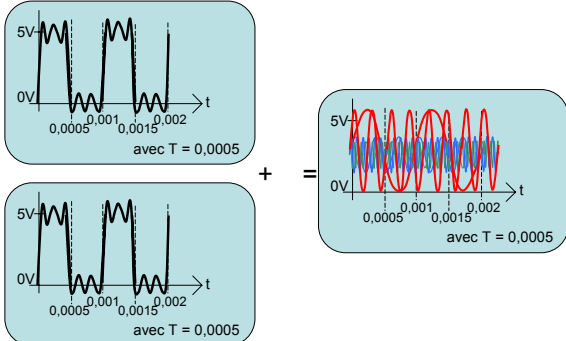


Les supports matériels de la communication entre ordinateurs

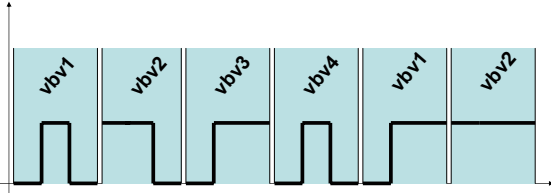


Transporter un signal carré sur un support analogique
(e.g. téléphone)

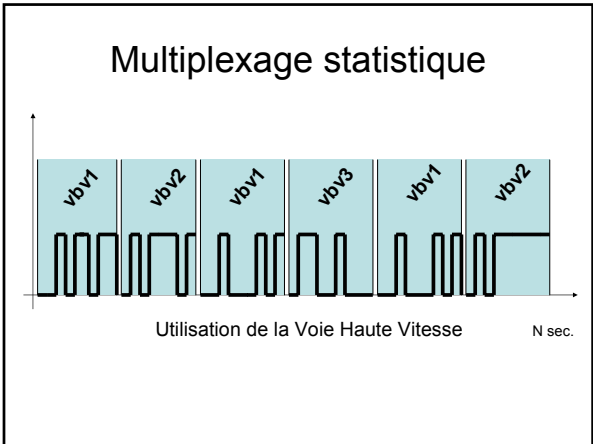
Les supports matériels de la communication entre ordinateurs

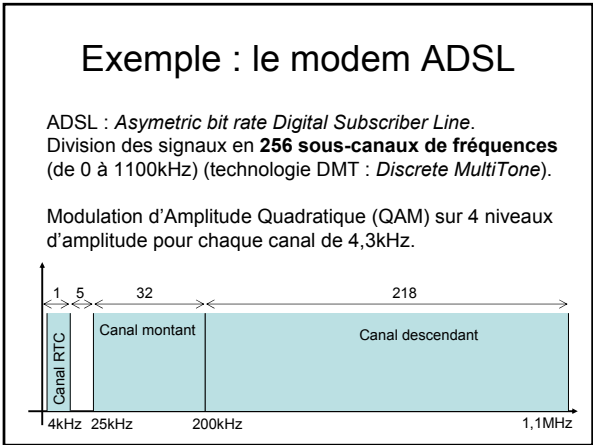


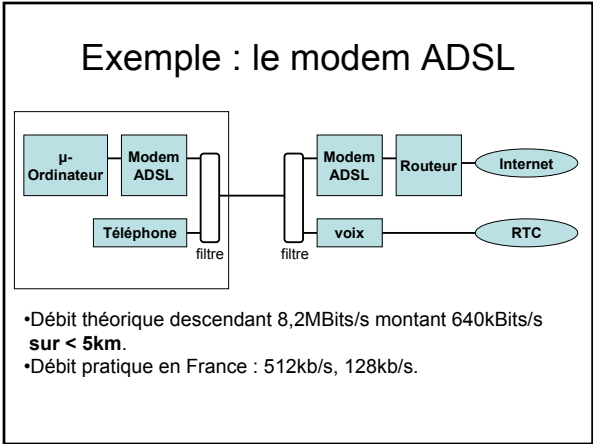
Multiplexage temporel



Utilisation de la Voie Haute Vitesse N sec.







Gestion de la liaison

Rôle et fonctionnement de la couche 2 du modèle OSI.

Détection et correction d'erreur

Aucun support physique de communication n'est absolument fiable. Une liaison conventionnelle a une probabilité d'erreur entre :

10^{-5} et 10^{-7}

Cette probabilité croît avec la distance parcourue par l'information. Elle peut exploser dans des conditions exotiques.

Le logiciel doit prévoir ces risques de défaillance, donc savoir les reconnaître (**détecter**) et décider d'une mesure appropriée pour y remédier (**corriger**) ou pas (erreur de moindre importance).

Bit de parité

Convenir du nombre de bit avant le bit de parité et de la parité souhaitée (paire/impair). Par exemple 8 + pair.

0	01000001	0..0..1..0..0..0..0..1	0	01000001
1	01110000	1..0..1..1..1..0..0..0..0	1	01110000
1	01100001	1..0..1..1..0..0..0..0..1	1	01100001
1	01000110	1..0..1..0..0..0..0..0..1	1	01000110

Autre exemple de code ASCII 7 bits + 1 parité.

Code à Redondance Cyclique

Les valeurs transmises sont vues comme des polynômes manipulés avec une arithmétique modulo 2 :

$$\begin{array}{rcl} 101101 & = & x^5 + x^3 + x^2 + x^0 \\ + 1011 & = & x^3 + x + x^0 \\ \hline 100110 & = & x^5 + x^2 + x \end{array}$$

$$\begin{array}{rcl} 101101 & = & x^5 + x^3 + x^2 + x^0 \\ - 1011 & = & x^3 - x - x^0 \\ \hline 100110 & = & x^5 + x^2 + x \end{array}$$

Code à Redondance Cyclique

Les deux interlocuteurs (l'émetteur et le récepteur du signal) conviennent d'un :

« *polynôme générateur* » noté $G(x)$

Exemple :

$$\begin{aligned} \text{CRC-12} &= x^{12} + x^{11} + x^3 + x^2 + x^1 + 1 \\ \text{CRC-16} &= x^{16} + x^{15} + x^2 + 1 \\ \text{CRC-CCITT} &= x^{16} + x^{12} + x^5 + 1 \end{aligned}$$

On transmet une séquence binaire M' construite à partir du message binaire M concaténé avec un CRC de d bits construit de tel sorte que $M'(x)/G(x) = 0$.

Code à Redondance Cyclique

Comment construire le CRC ?

Avec un message
 $M = 1101011011$
 Et une fonction génératrice
 $G(x) = x^4 + x + 1$
 On construit un message
 $M' = 1101011011\ 0000$
 (4 zéro car G de degré 4).

Le reste de

$$\frac{M'}{G(x)} = \boxed{1110}$$

$$M' = 1101011011\ 1110$$

$$\begin{array}{r} 11010110110000 \\ \underline{10011} \\ 00011 \\ \underline{00011} \\ 00000 \\ \underline{00000} \\ 00101 \\ \underline{00000} \\ 01011 \\ \underline{00000} \\ 10110 \\ \underline{10011} \\ 01010 \\ \underline{00000} \\ 10100 \\ \underline{10011} \\ 01110 \\ \underline{00000} \\ 1110 \end{array}$$

Code de Hamming

Distance de hamming :

$d_H(000,010) = d_H(000,001) = \dots = 1$
 $d_H(000,110) = d_H(110,101) = \dots = 2$
 $d_H(000,111) = d_H(100,011) = \dots = 3$

Le code (0 = 000, 1 = 111) est :

- **2_detecteur** car il détecte jusqu'à 2 erreurs dans le symbole transmis ;
- **1_correcteur** car il permet de corriger une erreur par symbole

Un code de hamming est **t_detecteur** si :
 $d_{Hmin}(\text{Symboles}) > t$

Un code de hamming est **t_correcteur** si :
 $d_{Hmin}(\text{Symboles}) \geq 2t+1$

Exemple : Code de Hamming 1_correcteur pour 4 symboles

Soit S un alphabet de 4 symboles avec $S = \{ 00000, 00111, 11100, 11011 \}$

$\text{Min } d_H(x,y) = 3$
 $(x,y) \in S^2, x \neq y$

\Rightarrow Code 2_detecteur et 1_correcteur.

Corriger 00101 : $d_H(00101,00000) = 2$ donc 00101 n'est pas le premier symbole.
 $d_H(00101,00111) = 1$ donc 00101 est le second symbole
(avec 1 erreur)

S nous permet de coder 4 valeurs binaires différentes soit 2 bits :

Valeur	Code	Erreurs possibles					
00	00000	00001	00010	00100	01000	10000	
01	00111	00110	00101	00011	01111	10111	
10	11100	11101	11110	11000	10100	01100	
11	11011	11010	11001	11111	10011	01011	

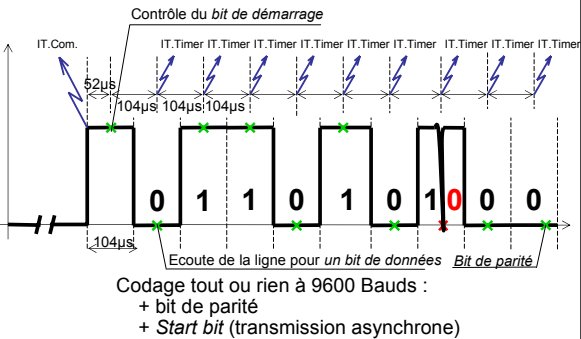
distance de Hamming

Du matériel aux couches basses
du logiciel.

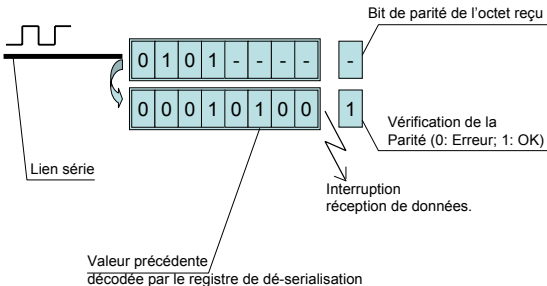
Selon les types de niveau de
perfectionnement du processus de
connexions physiques, le matériel pourra :

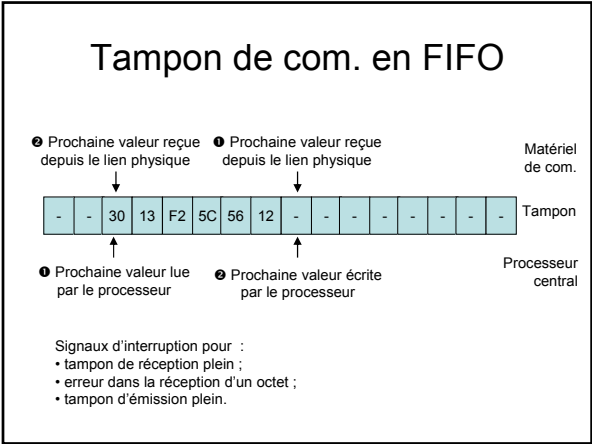
- donner la valeur instantanée de la ligne ;
- donner le dernier octet décodé ;
- stocker les octets reçus et non encore lus dans
un tampon ;
- ou directement écrire les valeurs reçues en
mémoire centrale (DMA).

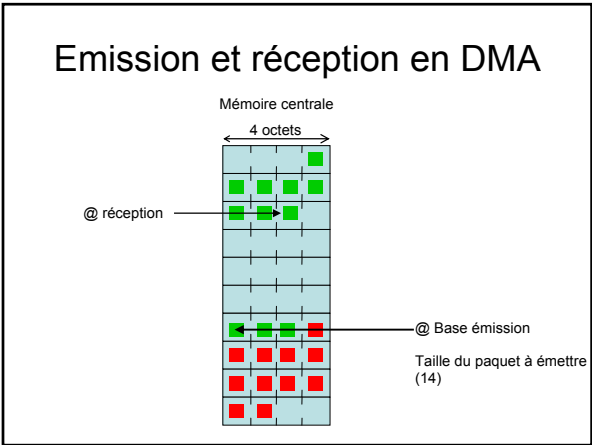
Échantillonnage d'un code de base



Registre de (dé)sérialisation
(UART)





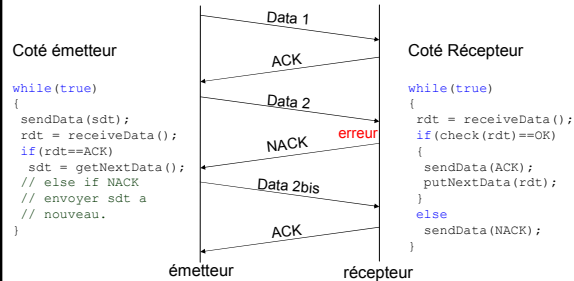


Protocoles de liaison

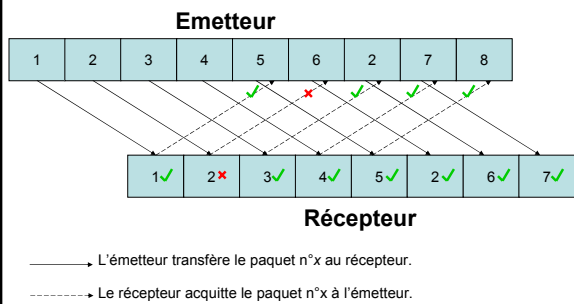
Deux approches différentes selon :

- Communication point à point :
 - Envoyer et attendre ,
 - Fenêtre de réception ;
- Communication en mode diffusion :
 - ALHOA ,
 - Ethernet (ISO 802.3).

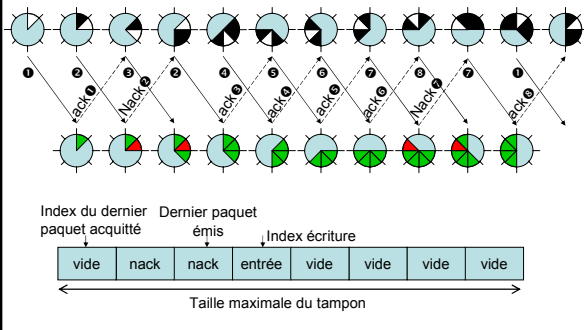
Protocoles du modèle point à point - Type « envoyer et attendre » -



Protocoles du modèle point à point - Protocoles à fenêtre glissante -



Protocoles du modèle point à point - Protocoles de fenêtre glissante -



Protocole BSC

SYN	SYNchronous idle	Octets de synchronisation des horloges.
ENQ	Enquiry	Invite une station à émettre ou à recevoir.
SOH	Start Of Heading	Début d'en-tête.
STX	Start of TeXt	Fin d'en-tête et début de texte.
ETB	End of Transmission Block	Fin de block de donné.
ETX	End of TeXt	Fin du texte et début des caractères de ctrl.
ACK	ACKnowledgement	Accusé de réception positif.
NACK	Negative ACKnowledgement	Accusé de réception négatif.
DLE	Data Link Escape	Caractère d'échappement de transmission.
EOT	End Of Transmission	Fin d'un transfert de données.

Émetteur Séquence de transmission de la forme : Récepteur

« SYN SYN SYN SYN SOH ... en-tête ... STX ... texte ... ETX BCC EOT »

BCC obtenu avec le polynôme CRC-CCITT : $x^{16} + x^{12} + x^5 + 1$

Protocole SIIP

Exemple de Message SIIP.

Protocole PPP

Automate de base pour une connexion PPP

Protocole PPP - établissement - RFC 1661 -

Nom	direction	Description
Requête de configuration	l → R	Propose une configuration spécifique des paquets transportés.
AZR (Ack) positif de configuration	l ← R	Accusé de réception positif pour une demande de configuration des paquets
AZR (Ack) négatif de configuration	l ← R	Accusé de réception négatif pour une demande de configuration des paquets.
Configuration rejetée	l ← R	Une demande de négociation sur une option non négociable a été formulée.
Requête de terminaison	l → R	Demande de fermeture de la ligne.
AZR (Ack) de terminaison	l ← R	Demande de fermeture acceptée.
Code rejeté	l ← R	Identifiant de requête inconnue.
Protocole rejeté	l ← R	Requête de protocole inconnu.
Requête d'écho	l → R	Demande d'écho (pour test)
Réponse d'écho	l ← R	Retour d'écho...
Requête à jeter	l → R	Requête à ne pas traiter. (pour test).

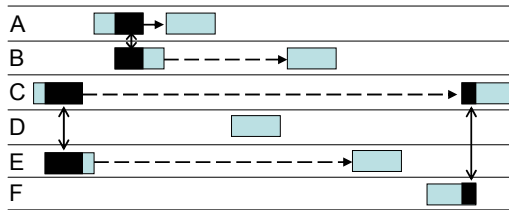
RFC 1661 : « <http://www.ietf.org/rfc.html> »

[illegible]

Protocoles du modèle en diffusion

- ALOHA -

Les stations A, B, C, D, E et F sont connectées via un support physique commun.

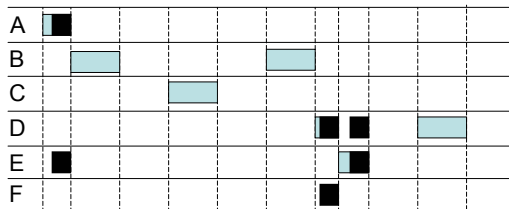


Émettre et, en cas de collision, attendre une durée aléatoire avant de ré-émettre.

Protocoles du modèle en diffusion

- p-persistent CSMA / CD -

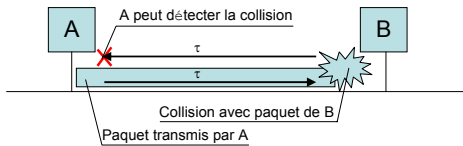
Dans ce cas les émetteurs potentiels ne parlent que pendant des Slots de temps si personne ne parle déjà. Et avec une probabilité p ($p=0,01$ par exemple).



utilisation d'un codage de base type Manchester pour pouvoir détecter les collisions
(en bande de base 0+0=0...).

Au mieux : 37% de succès, 37% de slots vides, 26% de collisions.

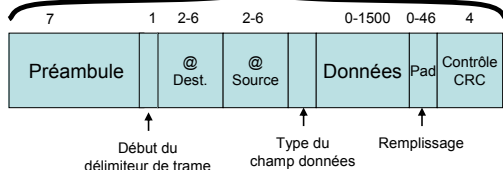
Trame Ethernet



La distance max entre deux stations ethernet est de 2500m.
à 200 000 km/s = 12,5µs + 3,5µs par repeteur et 4 repeteurs pour 2500 m.
Soit 52µs pour l' A/R.
Or 10Mbps : 1 bit = 0,1 µs.
Donc la longueur minimale d'un message et de 500bits (~ 64octets)

Trame Ethernet

Nombre d'octets



Format d'une trame IEEE 802.3

Avec : octet de préambule: 10101010
octet délimiteur de trame : 10101011
@source : identifiant unique intégré à la carte sur 6 octets.
@dest. : 6 octets intégrés à la carte réseau (broadcast : ff:ff:ff:ff:ff:ff)

Adresses physiques :

unicast, broadcast, multicast.

Les cartes réseaux de type Ethernet disposent d'un code identifiant (théoriquement) unique à chaque carte. Cet identifiant est codé sur 6 octets.

Une trame peut avoir pour destinataire le code identifiant d'une autre carte. Il s'agit d'un message **unicast** (un vers un).

Une trame peut avoir pour destinataires toutes les machines présentes sur le lien physique. Il s'agit d'un message **broadcast** (un vers tous).

Une trame peut avoir pour destinataire une adresse virtuelle qui concerne un groupe de récepteur sur le lien physique. Il s'agit d'un message **multicast** (un vers plusieurs).

Répéteur, pont et commutateur

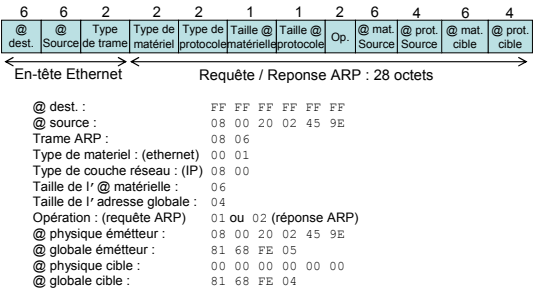
Répéteur : recevoir et amplifier et retransmettre un signal depuis un port vers un(des) autre(s). Un *Hub* est un répéteur 10BaseT multi port.

Pont : un pont relie des liaisons disjointes en filtrant les paquets selon l'adresse physique de leur destinataire.

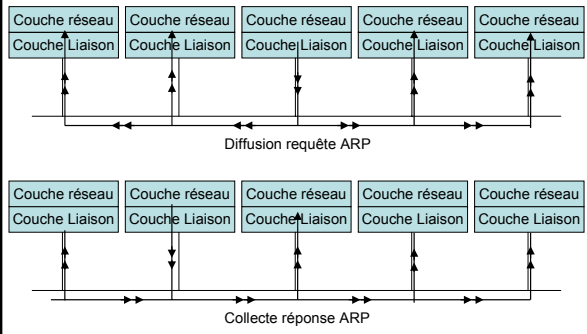
Commutateur : un commutateur est un pont multi port qui est capable de faire de la conversion de protocole entre différentes liaisons. Il ne s'intéresse donc pas aux adresses physique des paquets transmis mais aux adresses globales (indépendantes du matériel).

Ethernet et ARP

Rechercher la présence d'une machine associée à une adresse globale (@ IP) sur un réseau local.



Ethernet et ARP



Réseaux et routage

Rôle et fonctionnement des logiciels associables à la couche 3 du modèle OSI.

La couche réseau

La couche réseau a pour objectif de permettre à une machine X de dialoguer avec une machine Y par l'intermédiaire d'un chemin. Ce chemin est constitué d'un ensemble de supports physiques de communications hétérogènes, de machines nœuds du réseau supportant des protocoles de liaison et relayant les messages échangés de liaison en liaison.

Adressage des machines - exemple de l'OSI (NSAP) -

AFI	IDI	DSP
-----	-----	-----

← GDP →

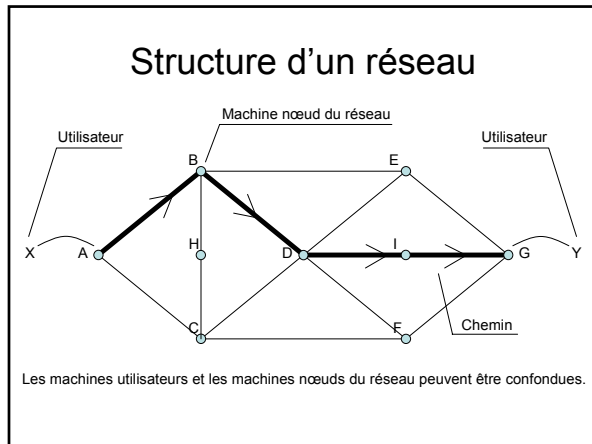
G.D.P. : *Global Domain Part*
(Définit la forme des adresses présentent dans D.S.P...)
A.F.I. : *Authority and Format Identifier*
I.D.I. : *Initial Domain Identifier*
D.S.P. : *Domain Specific Part*
(L'adresse selon le mécanisme d'adressage du réseau visé...)

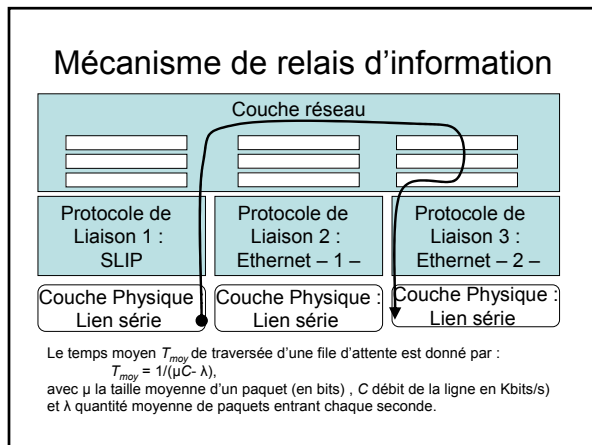
Analogie avec le téléphone :

(X) Y Z Z Z Z Z Z Z Z

← Région d'appel Identifiant de numéro à l'intérieur de la région

Identifiant de fournisseur de réseau global





Concevoir une couche réseau

Définir le modèle d'échange de l'information transportée :

- en mode connecté ;
- en mode non connecté.

Défenseurs du mode non connecté :
ARPA (Internet)

Défenseurs de mode connecté :
« les transporteurs »

Argumentaires		
Sujet	Service avec connexion	Service sans connexion
Initialisation :	Nécessaire.	Impossible.
@ du destinataire :	Nécessaire uniquement à l'installation.	Nécessaire dans chaque paquet.
Séquençement des paquets:	Garanti.	Non Garanti.
Contrôle d'erreur :	À la charge du réseau.	À la charge des utilisateurs.
Contrôle de flux :	À la charge du réseau.	A la charge des utilisateurs.
Possibilité de négociations :	Oui	Non

Propriétés souhaitables pour un algorithme de routage :

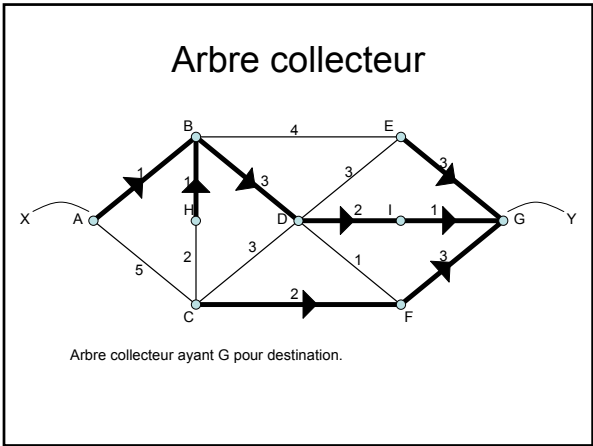
- Exactitude ;
- Simplicité ;
- Robustesse (aux MaJ & Défaillances des Machines) ;
- Stabilité (garantie de convergence) ;
- Justice (vis-à-vis des usagers) ;
- Optimisation (minimiser le temps de traversée, mais aussi, maximiser le flux de transmission) .

(Propriétés parfois contradictoires)

Algorithme du plus court chemin

Chemin retenu :
X→Y = XABDIGY

A partir du graphe du réseau, déterminer une valeur pour chaque liaison/arc (1 fois pour toutes ou périodiquement avec des messages de contrôle de la couche liaison par exemple), dans l'unité de mesure choisie (distance, temps de transmission, charge supportée,...).



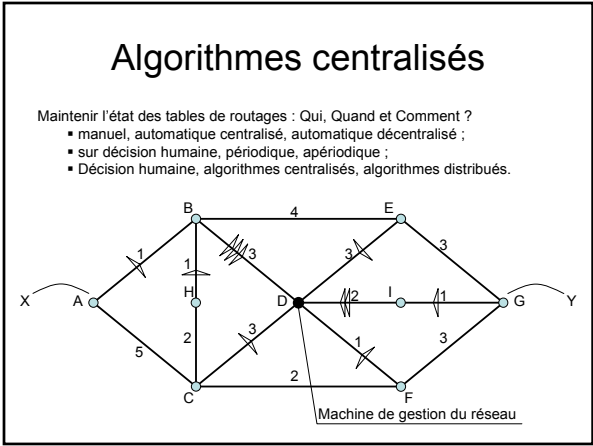
Algorithme de routage multi-chemin

Objectif : disposer de chemins de substitution,
• en cas de perturbation de certaines liaisons ;
• en cas de panne de certaines machines nœuds.

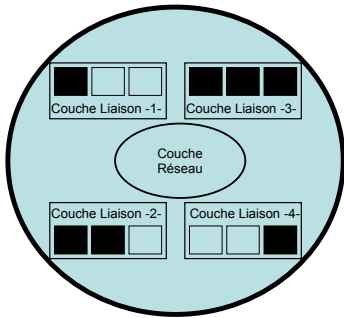
Stratégie : Dans les tables de routage de chaque machine nœud indiquer n liaisons possibles pour atteindre la machine cible. Chaque liaison pourra être pondérée avec un poids qui permet alors de définir la probabilité que les paquets soient routés par chacune des liaisons possibles pour sa destination.

Exemple : Table de routage de A :

	Id de liaison	Probabilité d'usage
@dest. : A liaisons :	{(-,1)}	
@dest. : B liaisons :	{(→B, 95), (→C; 0,05)}	
@dest. : C liaisons :	{(→C; 0,95), (→C; 0,05)}	
@dest. : * liaisons :	{(→B; 0,77), (→C; 0,23)}	



Algorithmes de routage local

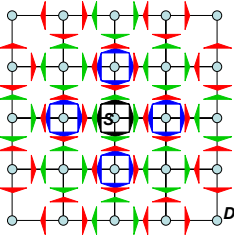


- **Local simple :**
exemple : choisir la file de sortie la moins chargée.
- **Local / statique :**
en cas de surcharge, abandonner la liaison prévue statiquement.
- **Savoir différé :**
Local+marquage des paquets avec infos Source+Distance. A chaque réception de paquet via une liaison on note que via cette liaison la source est à cette distance, on incrémente la distance parcourue par le paquet et on le réémet sur une autre liaison (appropriée).

Algorithmes de routage par inondation

Tout paquet reçu est réémis sur toutes les liaisons.
⇒ Génération d'un nombre infini de duplication de paquets!
Limiter le processus en utilisant par exemple un compteur de durée de vie du paquet. Cet algorithme ne nécessite aucune connaissance minimale du réseau. (la distance max entre deux nœud).

- ▶ Paquet initial Pack1 (transmit sur les 4 liaisons de S)
- ▶ Paquets générés à l'étape 1 (1ere retransmission du Pack1)
- ▶ Paquets générés à l'étape 2 (2ere retransmission du Pack1)
- ▶ Paquets générés à l'étape 3 (3ere retransmission du Pack1)



Algorithme de routage distribué

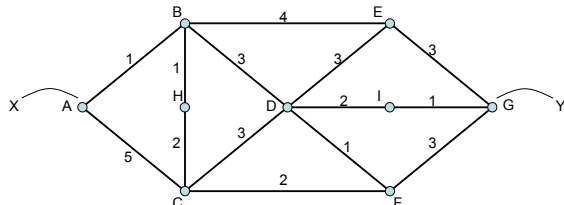


Table initiale de A :	Table initiale de B :	Table initiale de C :	Table de A après réception de B et C :
A→A : - A→B : (B) 1 A→C : (C) 5	B→A : (A) 1 B→H : (H) 1 B→D : (D) 3 B→E : (E) 4	C→A : (A) 5 C→H : (H) 2 C→D : (D) 3 C→F : (F) 2	A→A : - A→B : (B) 1 A→C : (C) 5 A→D : (D) 4 A→F : (F) 6
			A→B : (B) 1 A→H : (H) 2 A→E : (E) 5

Les algorithmes de routage hiérarchiques

Table de routage de 2C :

2A	→ 2B
2B	→ 2B
2C	→ -
2D	→ 2D
1*	→ 2B
3*	→ 2D
4*	→ 2D
5*	→ 2B

Algorithmes de gestion de la diffusion

Certains paquets sont destinés à plusieurs interlocuteurs présents sur le réseau global (multicast).

Les techniques de routage de ces paquets peuvent être :

- 1 paquet par dest. : rendement médiocre du réseau ;
- Inondation de ces paquets : largement sous optimal ;
- Routage multi destination :
Chaque paquet contient une liste d'@ destination. Le paquet est réémis sur les liaisons associées à au moins 1 adresse de la liste.
- Routage selon le chemin inverse :
Lorsqu'un paquet est reçu en mode multi destinataire, la machine de routage regarde l'adresse de l'émetteur. Si le paquet vient de la liaison qui aurait été utilisée pour retransmettre un paquet vers la source il le retransmet sur toutes les autres liaisons. Sinon il ne retransmet pas le paquet reçu...

Congestion du réseau

Gérer la congestion :

Proposer des mécanismes visant à garantir le bon fonctionnement du réseau lorsque le nombre de paquets entrant excède les capacités du réseau.

Gestion de la congestion

- Pré allocation des tampons -

L'excès de paquet se manifeste par une saturation des tampons d'émission des machines qui forment les nœuds du réseau.

Première idée : Pré réserver les tampons pour chaque chemin initié dans le réseau.

⇒ Connaître les chemins utilisés

- ⇒ Réseau en mode connecté
- ⇒ En cas de protocole « à fenêtre glissante » entre la source et la destination, il faut autant de tampons que le prévoit la fenêtre glissante dans chaque intermédiaire.

Gestion de la congestion

- Destruction des paquets -

Si une liaison monopolise l'ensemble des tampons disponibles, les paquets à destination d'autres liaisons sont perdus.

Il faut conserver un nombre minimum de tampons libres pour d'autres liaisons.

Tampons libres

Selon les résultats d'Irland (78), un maximum simple est :

$$m = k / \sqrt{s}$$

avec m le nombre max. de tampon pour une file, k le nombre de tampon dispos, et s le nombre de liaisons de sortie.

Les étreintes fatales

(DeadLock)

e.g. Liaison en « fenêtre glissante »

La couche réseau est bloquée, le réseau se congestionne.

- Contrôle de congestion isarithmique -

Objectif :

interdire l'émission de paquets lorsque le réseau a atteint sa charge de travail maximale.

Proposition :

Chaque paquet représente un jeton. Chaque machine nœud du réseau dispose initialement d'un certain nombre de jetons.

Lorsqu'une machine émet un paquet elle perd un jeton. Lorsqu'elle reçoit un paquet elle gagne un jeton.

Limite de la solution :

- ⇒ Problème de perte de performance du réseau lorsque des paquets sont perdus entre leur émission et leur réception.
- ⇒ n'empêche pas un nœud de recevoir plus de paquet qu'il ne peut en gérer (pas de garantie de flux).

Le contrôle de flux

Réduire la quantité de paquets échangés entre les machines nœuds chaque seconde lorsque le réseau se congestionne pour éviter la congestion.

Solution correcte pour éviter la surcharge :

- des liaisons physiques ;
- des capacités de traitement des machines nœuds ;

Solution médiocre pour répondre à une congestion.

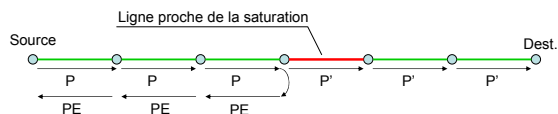
- ⇒ Inadaptée à un trafic irrégulier ;
- ⇒ Sous-exploite la capacité de transport du réseau.

Paquets d'engorgement

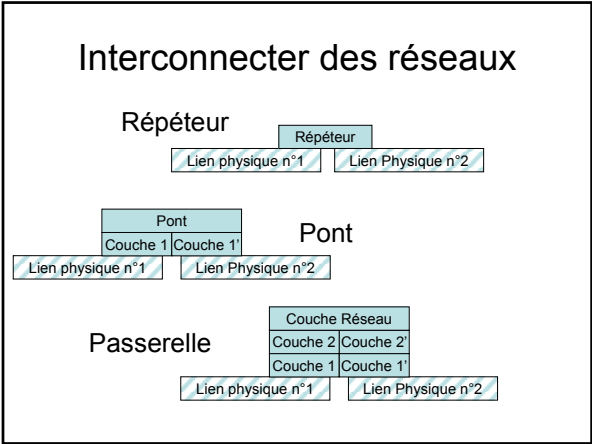
Calculer un taux d'occupation (u) maximum et décider d'un seuil d'occupation acceptable.

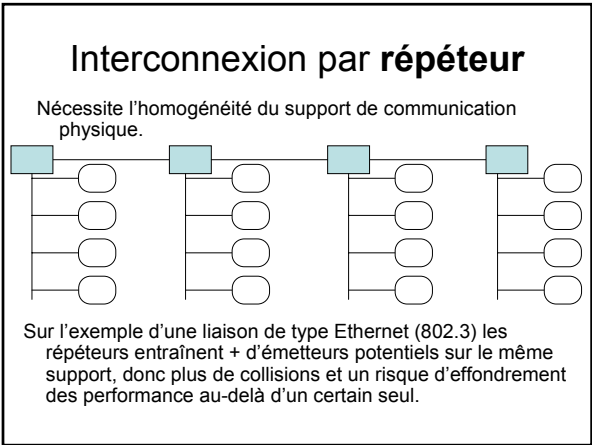
On peut calculer u avec : $u_{\text{nouveau}} = a \cdot u_{\text{ancien}} + (1-a)f$

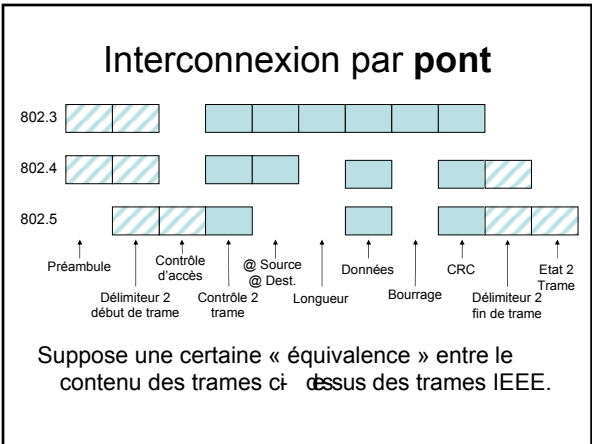
Où f est 0 ou 1 selon que la ligne est occupée lors de l'échantillonnage.
 a est la « faculté d'oublier » les échantillons anciens.



- P : Paquet émis par la Source (et réémis par les intermédiaires) ;
- P' : Paquet émis par la Source avec un tag de saturations ;
- PE : Paquet d'engorgement à destination de la source.
(pour que la source réduise son débit vers la Dest.)







Interconnexion par pont

Exemples des traitements informatiques à réaliser pour chaque type de conversion.

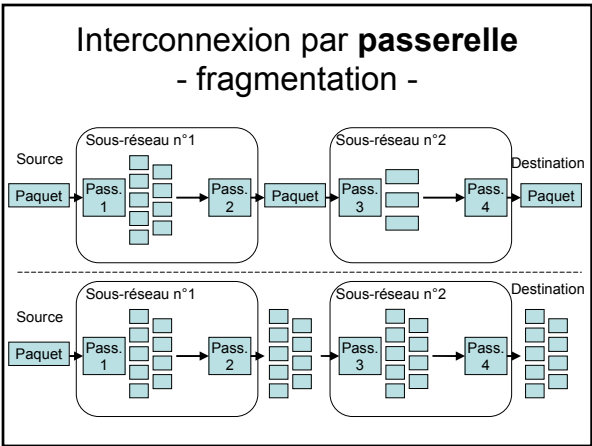
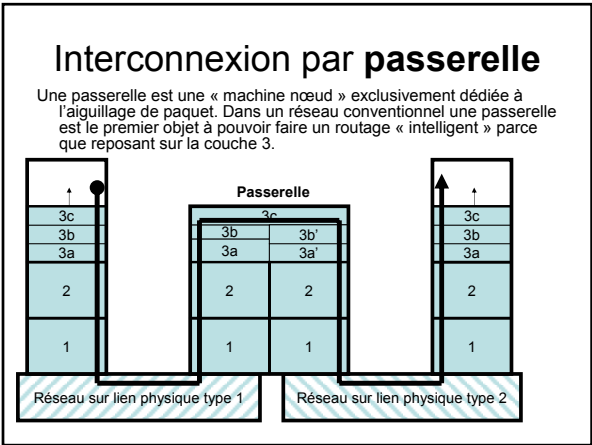
	802.3 « Ethernet »	802.4 « Token bus »	802.5 « Token ring »
802.3		1,4	1,2,4,8
802.4	1,5,8,9,10	9	1,2,3,8,9,10
802.5	1,2,5,6,7,10	1,2,3,6,7	6,7

Actions :

1. Reformater la trame et recalculer le CRC
2. Inverser l'ordre des bits
3. Copier la priorité qu'elle ait ou non un sens
4. Générer une priorité fictive
5. Annuler la priorité
6. Vider l'anneau
7. Positionner les bits A et C
8. Problème de congestion Lien rapide vers Lien Lent
9. Problème du jeton de transfert, ACK reporté ou annulé
10. Panique lorsque la trame est trop longue!

Annexes :

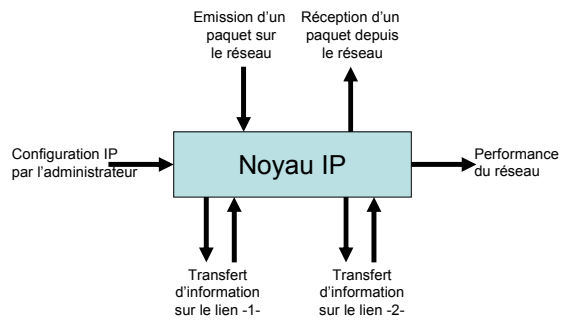
802.3	1518 octets	10Mbits/s
802.4	8191 octets	10Mbits/s
802.5	5000 octets	4Mbits/s



Internet Protocol

La couche IP du réseau
Internet

Rôle de la couche IP



Structure d'un réseau IP

Matériel hétérogène :

- Liaison série, parallèle ;
- Modems ;
- Liaison par bus (Ethernet 802.3, Tokenbus 802.4) ;
- Liaison en anneau (Tokenring 802.5) ;
- Liaison hertzienne (802.11).

Support réseau :

- Modèle non connecté ;
- Adresses sur 32bits ;
- Adressage / Routage hiérarchique ;
- Administration du routage / outils de mise à jour distribuée.

Adressage IP (IPv4)

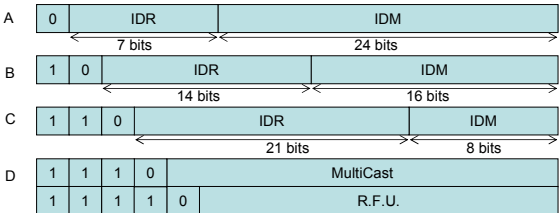
1 machine = 1 Code sur 32Bits = 4 octets
Exemple : « 134.206.11.2 » représentation @IP standard :
1 nombre par octet séparé par un point.
@IP décomposée en 2 parties : (id. réseau, id. machine).

Classe	Plage d'adresses	Nb de Réseaux	Nb. De machines
A	De 0.0.0.0 à 127.255.255.255	<127 (2 ⁷)	< 16 millions (2 ²⁴)
B	De 128.0.0.0 à 191.255.255.255	<16384 (2 ¹⁴)	< 65536 (2 ¹⁶)
C	De 192.0.0.0 à 223.255.255.255	<2millions (2 ²¹)	< 256 (2 ⁸)
D	De 224.0.0.0 à 239.255.255.255	MultiCast	
E	De 240.0.0.0 à 247.255.255.255	R.F.U.	

Adressage IP (IPv4)

Représentation interne d'une @ codée sur 32 bits, décomposée en deux parties :

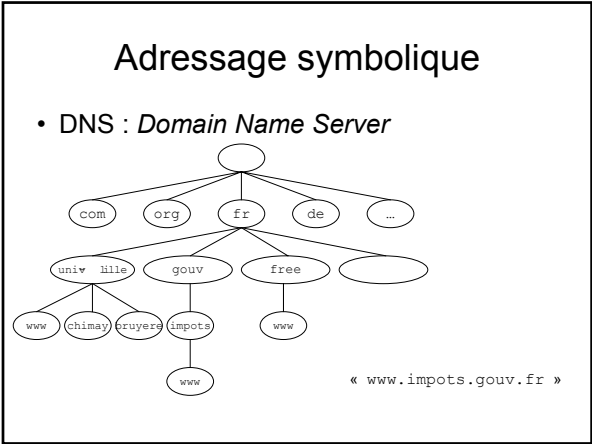
- Identifiant de réseau (IDR) ;
- Identifiant de machine dans le réseau (IDM).

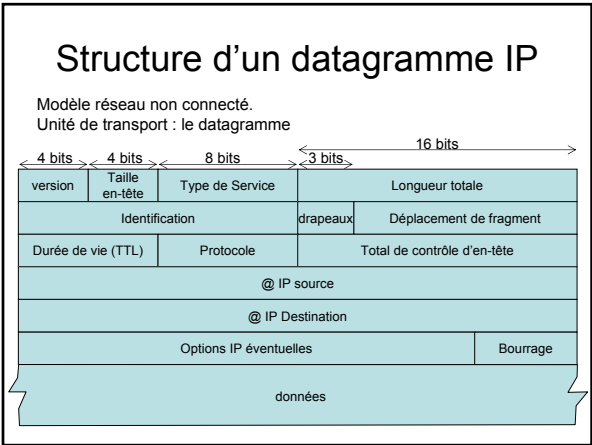


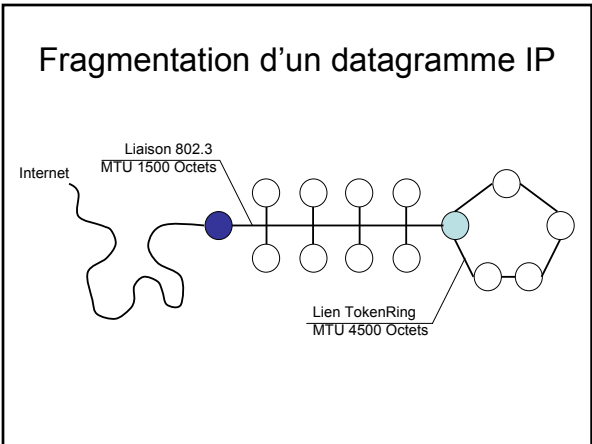
Adresses particulières

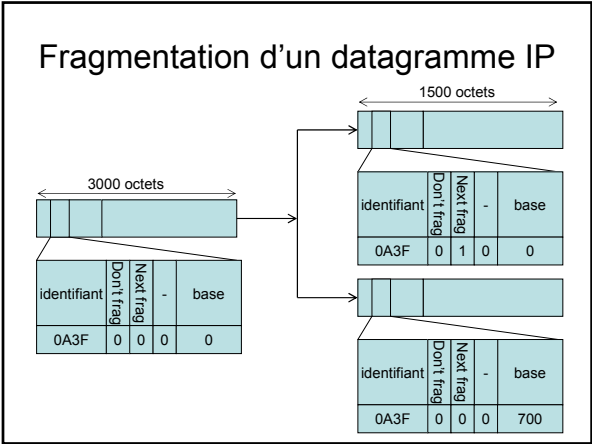
Adresses réservées à des usages particuliers :

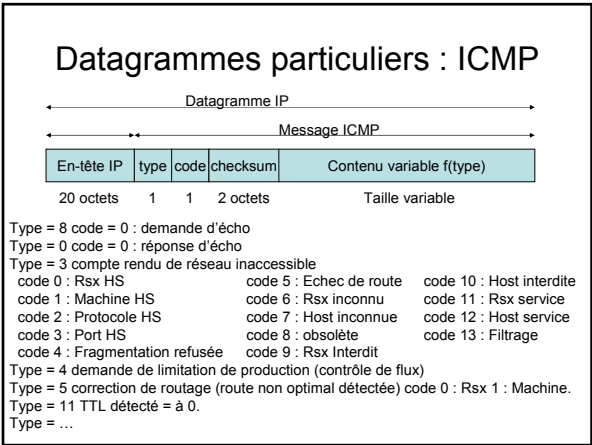
- « 0.0.0.0 »
- « idr nul, ∀idm », machine sur le réseau local ;
- « ∀idr , idm nul », désigne le réseau lui-même ;
- « 127.X.Y.Z », autre possibilité pour désigner la machine locale.
- « ∀idr , 11..1₍₂₎ », broadcast sur le réseau ;
- « 11..1 , 11..1₍₂₎ », broadcast sur le réseau de l'émetteur ;
- « 10.0.0.0 à 10.255.255.255
ou 172.16.0.0 à 172.31.255.255
ou 192.168.0.0 à 192.168.255.255 », réservé pour des intranets.

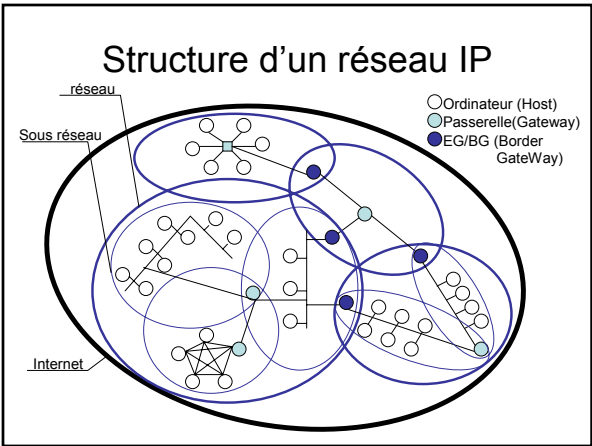












Structure d'un réseau IP

« 134.206.X.Y » Les machines A,B,C sont des machines classiques « Hosts » du réseau : « 134.206.X.X »

Les passerelles R1 et R2 sont des machines « Gateway ». Visibles dans 2 réseaux ⇒ 2 adresses.

« 134.207.X.Y »

« 134.206.X.Y » S1 est une machine passerelle de sous réseau dans le réseau « 134.206.X.Y ». ⇒ Définition d'un masque de sous réseau.

Routage IP

- Table de routage « administrative » :

@ réseau	masque réseau	@ passerelle	@ interface	métriques
0.0.0.0	0.0.0.0	134.206.3.1	134.206.11.2	1
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
134.206.0.0	255.255.0.0	134.206.11.2	134.206.11.2	1
134.206.11.2	255.255.255.255	127.0.0.1	127.0.0.1	1
134.206.255.255	255.255.255.255	134.206.11.2	134.206.11.2	1
224.0.0.0	224.0.0.0	134.206.11.2	134.206.11.2	1
255.255.255.255	255.255.255.255	134.206.11.2	134.206.11.2	1

Protocoles de type IGP¹ : exemple RIP²

Mise à jour des tables selon l'algorithme de routage distribué.

En-tête IP		
En-tête UDP		
Commande (1=Q;2=R)	Version (v1 ou v2)	= 0
Id. de famille d'adresse (=2)		= 0
@ IP		
v1 = 0 v2 = masque de sous réseau		= 0
Max 24 autres routes		

↑ Une route connue, avec le masque réseau et la métrique.

¹ IGP : Interior Gateway Protocol
² RIP : Routing Information Protocol

Les outils systèmes pour IP

```
Fichier des machines présentes sur le réseau :
/etc/hosts
127.0.0.1      localhost
134.206.11.7   bruyere.lifl.fr bruyere

# The following lines are desirable for IPv6 capable hosts
# (added automatically by netbase upgrade)

::1           ip6-localhost ip6-loopback
fe00::0       ip6-localnet
ff00::0       ip6-mcastprefix
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters
ff02::3       ip6-allhosts
Fichier de résolution symbolique :
/etc/resolv.conf
domain lifl.fr
search lifl.fr univ-lille1.fr
Nameserver 134.206.10.18
Nameserver 134.206.1.15
```

Les outils systèmes pour IP

```
Commande de test de la couche réseau :

>ping -f -n 4 -i 5 -l 248 bruyere
>ping -c 4 -t 5 -s 248 bruyere

Envoi d'une requête 'ping' sur 134.206.11.2 avec 248 octets de données:

Réponse de 134.206.11.2 : octets=248 temps<10 ms TTL=128
Réponse de 134.206.11.2 : octets=248 temps<10 ms TTL=128
Réponse de 134.206.11.2 : octets=248 temps<10 ms TTL=128
Réponse de 134.206.11.2 : octets=248 temps<10 ms TTL=128

Statistiques Ping pour 134.206.11.2:
Paquets : envoyés = 4, reçus = 10, perdus = 0 (perte 0%),
Durée approximative des boucles en milli-secondes :
    minimum = 0ms, maximum = 0ms, moyenne = 0ms
```

Les outils systèmes pour IP

```
Commande suivie de l'architecture réseau :

> tracert -d -h 20 www.impots.gouv.fr
> traceroute -m 20 www.impots.gouv.fr

Détermination de l'itinéraire vers www.impots.gouv.fr [195.101.154.66]
avec un maximum de 20 sauts :

 1  10 ms  <10 ms  10 ms  134.206.3.2
 2  <10 ms  <10 ms  10 ms  193.49.253.113
 3  10 ms  10 ms  <10 ms  193.54.138.117
 4  20 ms  <10 ms  10 ms  194.214.110.29
 5  10 ms  10 ms  10 ms  194.214.110.5
 6  10 ms  20 ms  10 ms  193.51.206.14
 7  10 ms  10 ms  20 ms  193.51.206.42
 8  10 ms  20 ms  10 ms  193.251.241.158
 9  10 ms  20 ms  30 ms  193.251.241.97
10  10 ms  20 ms  10 ms  193.251.126.23
11  20 ms  10 ms  10 ms  193.251.126.37
12  10 ms  10 ms  40 ms  194.51.159.206
13  10 ms  20 ms  10 ms  194.51.159.226
14  160 ms 150 ms 170 ms 194.51.195.6
15  *
```

Les outils systèmes pour IP

Commande d'accès à la couche liaison :

```
> arp -s 157.55.85.212 00-aa-00-62-c6-09

> arp -a
Interface : 134.206.11.2 on Interface 0x2
  Adresse Internet  Adresse physique  Type
  134.206.3.2       00-90-bf-5a-9c-8c  dynamique
  157.55.85.212     00-aa-00-62-c6-09  statique

> arp -d 157.55.85.212
```

Les outils systèmes pour IP

Commande de teste de la couche réseau :

```
>route PRINT
>route -n
=====
Liste d'Interfaces
0x1 ..... MS TCP Loopback interface
0x2 ...00 a0 24 50 0e 66 ..... 3Com EtherLink PCI
=====
Itinéraires actifs :
Destination réseau  Masque réseau  Adr. passerelle  Adr. interface  Métrique
0.0.0.0             0.0.0.0        134.206.3.1      134.206.11.6    1
0.0.0.0             0.0.0.0        134.206.3.2      134.206.11.6    1
127.0.0.0           255.0.0.0      127.0.0.1        127.0.0.1       1
134.206.0.0         255.255.0.0    134.206.11.6     134.206.11.6    1
134.206.11.6        255.255.255.255 127.0.0.1        127.0.0.1       1
134.206.255.255     255.255.255.255 134.206.11.6     134.206.11.6    1
224.0.0.0           224.0.0.0      134.206.11.6     134.206.11.6    1
255.255.255.255     255.255.255.255 134.206.11.6     134.206.11.6    1
Passerelle par défaut : 134.206.3.2
=====
Itinéraires persistants :
Aucun
```

Les outils systèmes pour IP

Commande de configuration du réseau :

```
> route ADD 157.0.0.0 MASK 255.0.0.0 157.55.80.1 IF 2 METRIC 3
> Route add -net 157.0.0.0 -netmask 255.0.0.0 -dev eth0

> route DELETE 157.0.0.0
> Route del 157.0.0.0

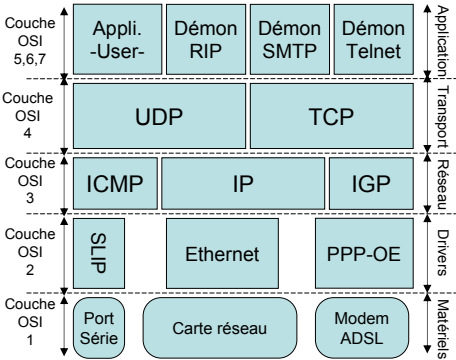
> route CHANGE 157.0.0.0 MASK 255.0.0.0 157.55.80.1 IF 3 METRIC 3
```

Les outils systèmes pour IP

Outil de mesure de l'activité d'une passerelle :

```
> netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 bruyere.lifl.fr:4810    zobe.linuxfr.org:www   ESTABLISHED
tcp        0      0 bruyere.lifl.fr:4809    zobe.linuxfr.org:www   CLOSE
tcp        0      0 bruyere.lifl.fr:4806    zobe.linuxfr.org:www   ESTABLISHED
tcp        0      0 bruyere.lifl.fr:888      lifl:32771              TIME_WAIT
tcp        0      0 bruyere.lifl.fr:823      lifl:32771              TIME_WAIT
tcp        0      0 bruyere.lifl.fr:www      barbar:1071             ESTABLISHED
tcp        0      0 bruyere.lifl.fr:ftp      regal:2278              ESTABLISHED
tcp        0      0 bruyere.lifl.fr:netbios  barbar:1030             ESTABLISHED
tcp        0  574 bruyere.lifl.fr:telnet   regal:2263              ESTABLISHED
tcp        0      0 bruyere.lifl.fr:netbios  kwak:1035              ESTABLISHED
```

Architecture des logiciels réseau IP



- Protocoles UDP & TCP -

Communications entre applications via TCP/IP.

Problématique de transport

La couche transport (couche 4) du modèle OSI a pour objectif de compléter l'activité de la couche réseau en réalisant des opérations de fragmentation, de dé fragmentation et de contrôle d'erreur ; afin d'offrir une interface de programmation exploitable par les applications présentes sur la machine.

La couche transport d'Internet gère deux protocoles :

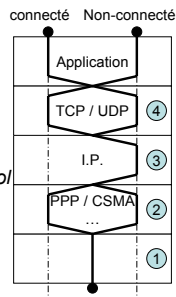
Interfaces avec les applications

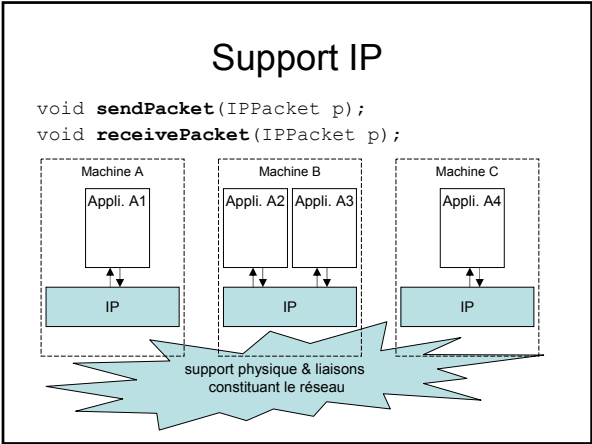
Modèle UDP : *User Datagram Protocol*

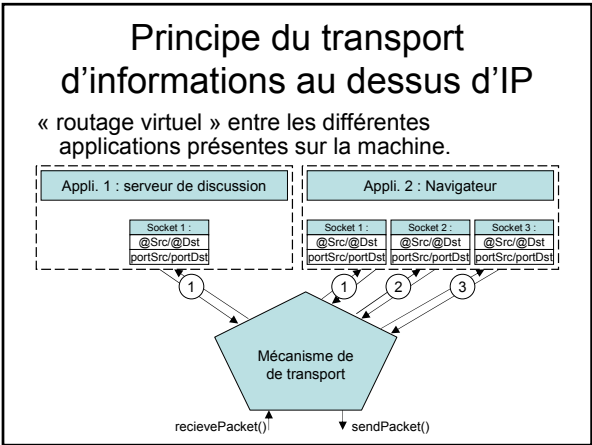
- Modèle de transmission sans connexion ;
- Pas de contrôle de séquence ;
- Pas de contrôle de flux ;
- Transport sous la forme d'un paquet IP (\Leftrightarrow tableau d'octet limité).

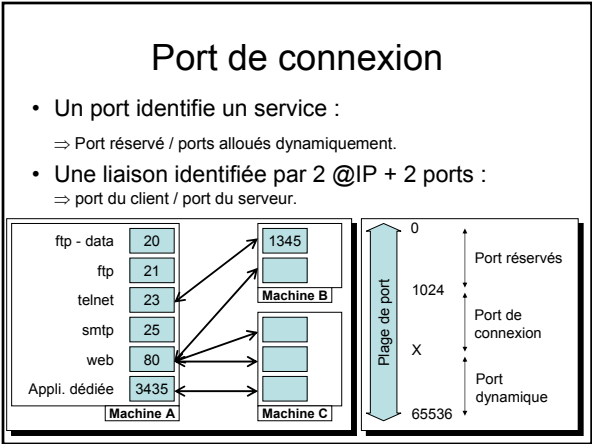
Modèle TCP : *Transmission Control Protocol*

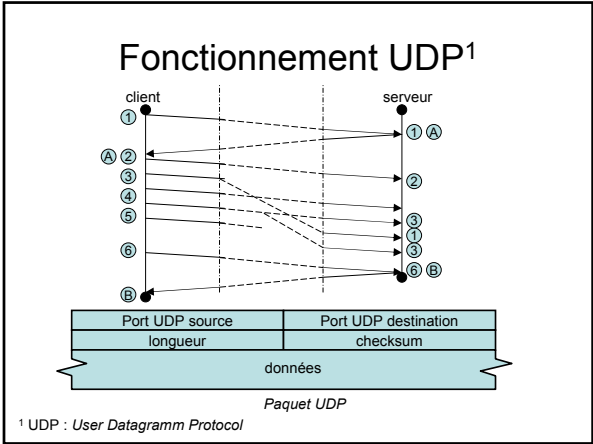
- Modèle de transmission avec connexion ;
- Gestion de la séquence ;
- Gestion du flux ;
- Restitution sous la forme d'un flot (\Leftrightarrow accès séquentiel d'un fichier).











émission d'un paquet UDP – avec Java –

```
DatagramPacket p;  
DatagramSocket s;  
InetAddress dst = InetAddress.getByName("brigant");  
int port = 1024 ;  
...  
p = new DatagramPacket (new byte[100],100, dst,  
    port);  
s = new DatagramSocket (/*ou port local spécifié*/);  
...  
s.send(p);
```

Réception d'un paquet UDP – avec Java –

```
import java.net.*;  
import java.io.*;  
...  
DatagramSocket s;  
DatagramPacket p;  
...  
s = DatagramSocket (/*port*/ 1024);  
p = DatagramPacket (/*buffer*/ new byte[512],512);  
...  
sock.receive(p);  
System.out.println("paquet reçu de :"+ p.getAddress()+  
    "port "+ p.getPort()+  
    "taille" + p.getLength());  
...  
s.close();
```

Exploitation UDP émission – en C –

```
int sock ;           // identifiant de socket
sockaddr_in name ;   // @ socket internet
hostent *hp ;        // identifiant IP

...
sock=socket( AF_INET, SOCK_DGRAM,0); // sock négatif => erreur
...
hp = gethostbyname("brigant.lifl.fr"); //convertir @alphanu => @IP
...
memcpy((char *)hp->h_addr, (char *)&name.sin_addr, hp->h_length);
name.sin_family = AF_INET;
name.sin_port = htons(5432);
...
gethostname(buf, max_buf); // obtenir le nom de la machine locale.
...
sendto(sock, buf, sizeof_buf, offset, (struct sockaddr*)&name,
        sizeof(name) ); // envoi du paquet buf vers name.
...
close(sock); // fermeture de la liaison.
```

Librairies nécessaires :
<sys/socket.h>
<netinet/in.h>
<netdb.h>

Exploitation UDP réception – en C –

```
Struct sockaddr_in adrSrv; // structure pour adresseIP
int s; // identifiant de socket

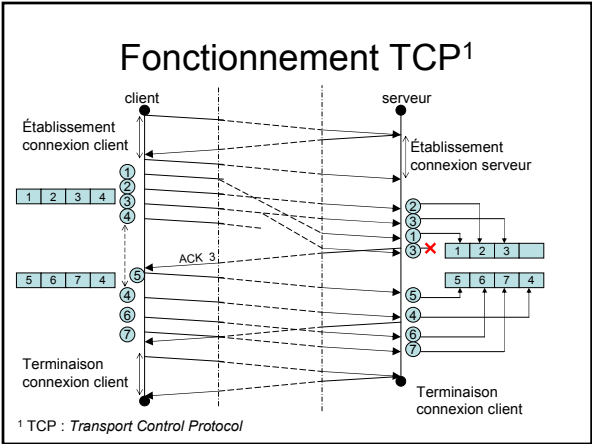
...
adrSrv.sin_family = AF_INET; // initialisation de l'adresse
adrSrv.sin_addr.s_addr = INADDR_ANY;
adrSrv.sin_port = htons(/* numero de port */);

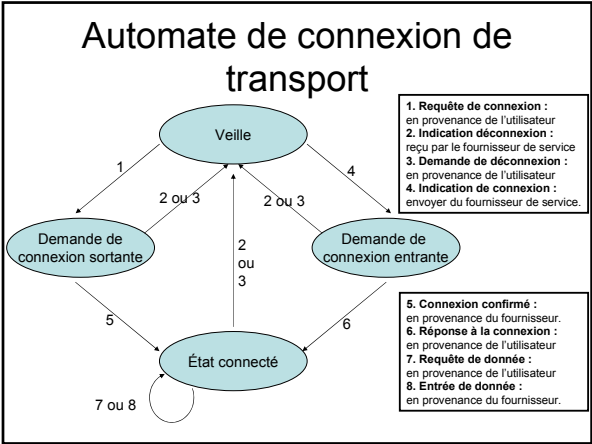
...
If (s = socket(AF_INET, SOCK_DGRAM, 0)) < 0 // création de la socket
// erreur de création de socket.

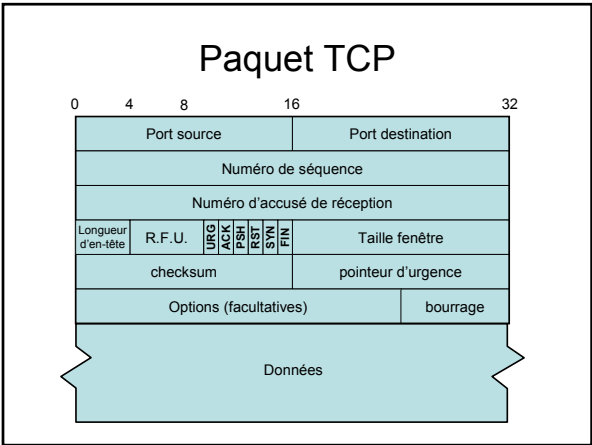
...
// liaison entre la socket et le système
if (bind(s, (struct sockaddr *)&adrSrv, sizeof(struct sockaddr_in)) < 0)
// erreur de liaison de socket.

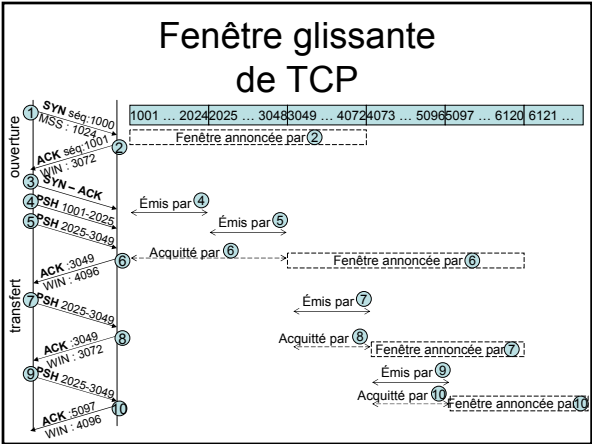
...
// émission et réception de paquets

Struct sockaddr_in addressClient;
Recvfrom(/*socket*/s, /*buffer de réception*/tampon, /*taille tampon*/ MAX_T,
/*offset*/0, /*adresse client*/ &addressClient, &taille);
```









Exploitation TCP
coté client – avec Java –

```
socket s; // déclaration de la liaison s.
InputStream iS; // déclaration du flux d'acquisition.
OutputStream oS; // déclaration du flux d'émission.
BufferedReader bF; // déclaration du support texte.
int v;
String str;

...
s = new socket("134.206.11.6",765); // établissement de la liaison s.
iS = s.getInputStream(); // obtention d'un flux d'acquisition.
oS = s.getOutputStream(); // obtention d'un flux d'émission.
bF = new BufferedReader(new InputStreamReader(iS)); // saisie texte.

...
v=iS.read(); // lecture d'un octet.
oS.write(v); // écriture d'un octet.

...
str = bF.readLine(); // lecture d'une ligne de texte.
s.close(); // fermeture de la liaison.
```

Exploitation TCP
coté serveur – avec Java –

```
import java.io.*;
import java.net.*;

...
ServerSocket sose; // serveur de socket.
Socket sock; // socket une fois la liaison établie.

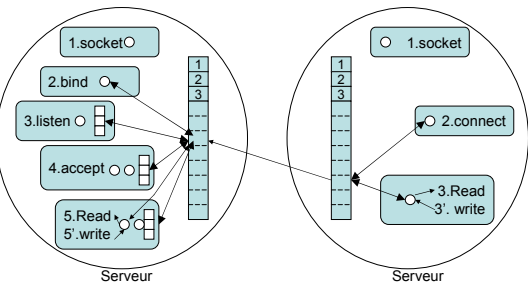
...
sose = new ServerSocket(7654); // création d'un serveur sur 7654.
sock = sose.accept(); // attente de connexion.
// ici on dispose d'une socket identique à celle du client.

...
InputStream in = sock.getInputStream();
OutputStream out = sock.getOutputStream();
idx = in.read();
...

```

Applications TCP/IP en C

Schéma de principe

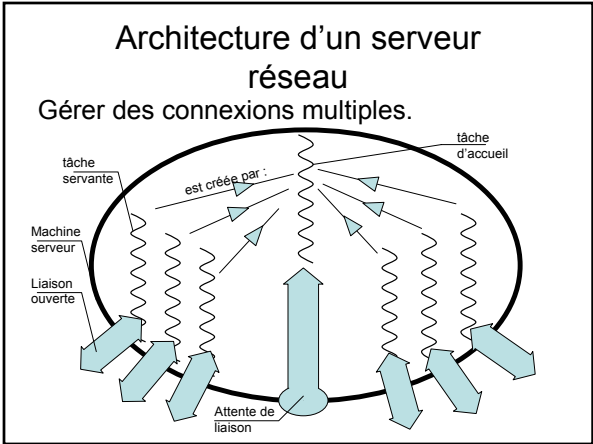


Exploitation TCP
coté client – en C –

```
Struct sockaddr_in saddr,addr;  
int s,s2,taille;  
  
...  
s = socket(PF_INET, SOCK_STREAM, 0);  
...  
// Initialiser saddr  
saddr.sin_family = AF_INET;  
saddr.sin_addr = getByName("brigant.lifl.fr");  
saddr.sin_port = htons(7654);  
...  
if(connect(s,(struct sockaddr *)&saddr,sizeof(saddr))<0)  
// erreur à la connexion ;  
...  
taille = read(s2,tampon,MAX_TAMPON) ; // valeur négative => erreur  
taille = write(s2,tampon,tailleA) ; // taille != tailleA => erreur.  
...
```

Exploitation TCP
coté serveur – en C –

```
Struct sockaddr_in saddr,addr;  
int s,s2,taille;  
  
...  
s = socket(PF_INET, SOCK_STREAM, 0);  
...  
// Initialiser saddr  
bind(df,(struct sockaddr *)&saddr,sizeof(saddr));  
...  
listen(df,MAX_CONNECTIONS); // initialisation de l'écoute.  
...  
s2 = accept(df,(struct sockaddr *)&addr,taille); // attente de  
connexion extérieure (S2 négatif => erreur).  
...  
taille = read(s2,tampon,MAX_TAMPON) ; // valeur négative => erreur  
taille = write(s2,tampon,tailleA) ; // taille != tailleA => erreur.
```



Outil socket générique

```
>telnet <nom-machine>[:port]
>telnet <nom-machine> [port]
```

```
/etc/services
nom      port/type  protocole # commentaire
...
ftp-data 21 / TCP
...
www      80 / TCP   http      # World Wide Web protocol
```

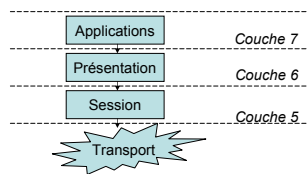
Les couches hautes du modèle OSI

Session
Présentation
Application

Le rôle des couches hautes

Le modèle OSI définit 3 couches qui n'ont pas d'équivalents « normalisé » dans le modèle TCP/IP. Pourtant les couche 5,6 et 7 assurent des fonctions qui répondent à des besoins applicatifs essentiels :

- (couche 5) gestion des échanges applicatifs ;
- (couche 6) homogénéisation des données ;
- (couche 7) normalisation des applications de base.

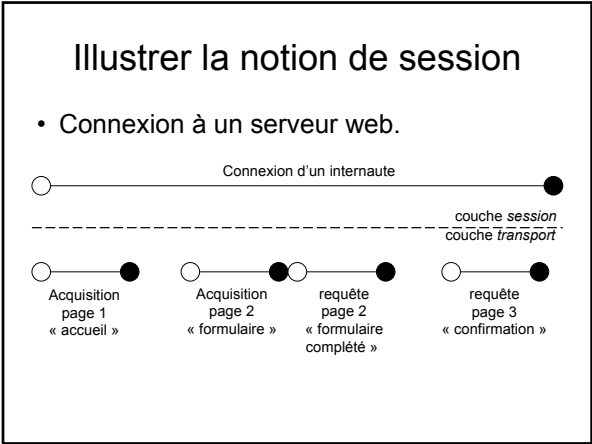


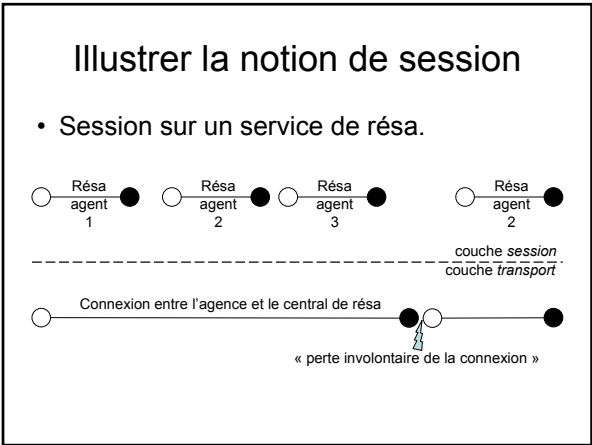
Notion de session

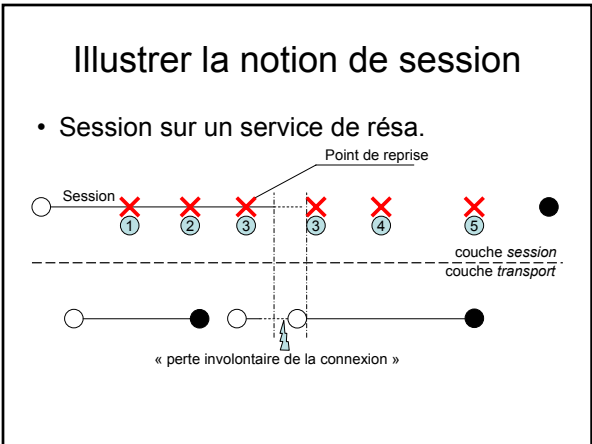
La couche de session fournit les moyens de synchroniser les échanges de données entre les applications.

La couche session a pour tâche de rendre cohérentes les dialogues et les changes de données au sein de l'application.

La couche session permet de définir le cadre d'un échange de donnée, avec un début , une fin, et un ordre de transmission pour chaque interlocuteur.







Implanter des points de reprise

Objectif : garantir la fiabilité de l'application en cas de problème dans la communication ou à cause de la machine distante. Exemple :

- Progression séquentielle des résultats \Rightarrow mémoriser la dernière étape pour pouvoir reprendre à partir de ce point ;
- Données obtenues/manipulées via un SGBD \Rightarrow Transaction ("begin", "commit", "rollback").

Notion de présentation

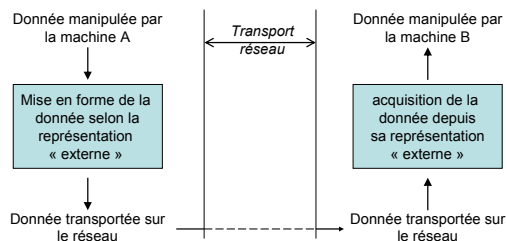
La couche présentation définit la syntaxe et la sémantique des informations transportées.

La couche présentation assure le transport de données dans un format homogène entre applications et ordinateurs hétérogènes.

Elle permet l'introduction de mécanismes de pré et post conditionnement de l'information en vue de son transport.

Notion de présentation

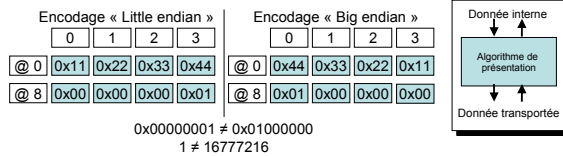
- Définir différentes manipulation de données.



Exemple de présentation

Format universel de données :

⇒ entiers 32 bits, codage des caractères, ...
e.g. Little endian, Unicode,...



⇒ Compression de données (sans perte, avec perte)
e.g. Gif, png, Jpg, mp3, mpg ...

Exemple de présentation

Format de documents :

⇒ définition **syntactique** et sémantique.

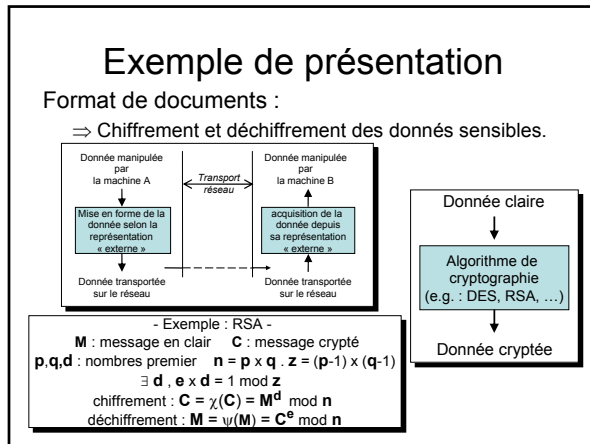
```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion> <major>1</major> <minor>0</minor> </specVersion>
  <actionList>
    <action> <name>RequestConnection</name> </action>
    <action> <name>ForceTermination</name> </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="no">
      <name>LastConnectionError</name>
      <dataType>string</dataType>
      <defaultValue>ERROR_NONE</defaultValue>
      <allowedValueList>
        <allowedValue>ERROR_NONE</allowedValue>
        <allowedValue>ERROR_ISP_TIME_OUT</allowedValue>
        <allowedValue>ERROR_COMMAND_ABORTED</allowedValue>
      </allowedValueList>
    </stateVariable>
  </serviceStateTable>
</scpd>
```

Exemple de présentation

Format de documents :

⇒ définition syntaxique et **sémantique**.

```
<html>
<head> <title>Cour</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000">
  <div align="center">
    <table width="600" border="0">
      <tr bgcolor="#000099">
        <td colspan="3">
          <div align="center"><font size="5"><b><font size="6"
            color="#FFFFFF">Enseignements</font></b></font></div>
          </td>
        </tr><tr>
          <td colspan="3">
            <p>Vous trouverez ci-dessous des documents relatifs au cours de :</p>
            <ul>
              <li><a href="#Rex">Recherche</a> - RSX- (Licence, Maîtrise, IUP);</li>
              <li><a href="#JC2">JavaCard</a> 2.1.1 (Seconds et troisiemes cycles).</li>
            </ul>
          </td>
        </tr>
      </table>
    </div>
  </body>
```



Exemple de présentation

RSA avec Java :

```
// Bibliothèques nécessaires :
import java.math.BigInteger;
import java.util.Random;

BigInteger un = new BigInteger("1");
BigInteger m, c, d, e, p, q, z, n;
// générer les clefs
p = new BigInteger(512, 10, new Random());
q = new BigInteger(512, 10, new Random());
z = p.subtract(un).multiply(q.subtract(un));
n = p.multiply(q);
d = new BigInteger(512, 10, new Random());
e = d.modInverse(z);

// chiffrer avec e et n
// byte[] ba message clair
m = new BigInteger(ba);
c = m.modPow(e, n);
bb = c.toByteArray();
// byte[] bb message crypté

// déchiffrer avec d et n
// byte[] bb message crypté
m = new BigInteger(bb);
m = c.modPow(d, n);
bb = m.toByteArray();
// byte[] ba message clair
```

Notion d'application

La couche application donne au processus d'application le moyen d'accéder à l'environnement OSI et fournit tous les services directement utilisables par l'application, à savoir :

- Le transfert d'informations ;
- L'allocation de ressources ;
- L'intégrité et la cohérence des données accédées ;
- La synchronisation des applications coopérantes.

Exemple d'application

Terminaux virtuels, deux catégories :

1. Texte ;

2. Graphique.

Difficulté à normaliser les code d'échappement pour le contrôle du terminal texte.

Différentes normes pour le graphisme : exemple X-window 11 (X11).

Exemple d'application

Systèmes de fichiers virtuels :

Objectifs :

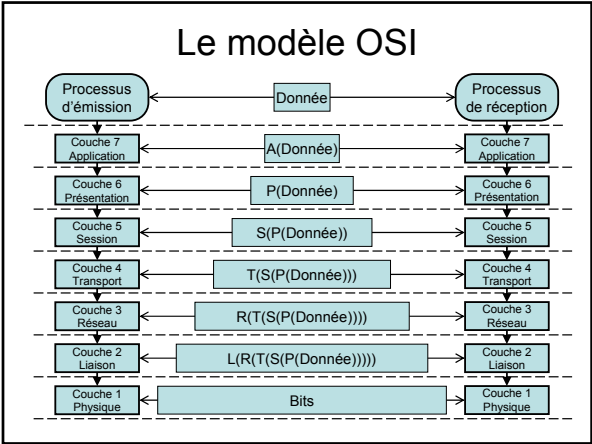
- administration centralisé ;
- accès simultanés ;
- répartition de charge ;
- duplicata de sécurité ;

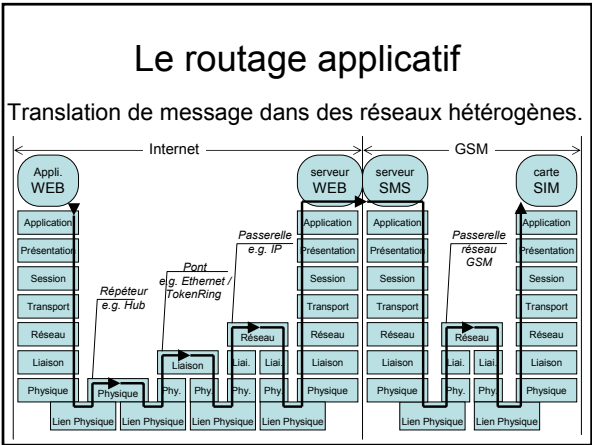
Exemple d'application

Service de courrier électronique

nom : Gilles.grimaud
adresse : lif1.fr
priorité : normal
cryptage : non

De : jeanjac
adresse : gemplus.com
date : 10 mars 2002
Objet : weekly report about Cigal project





FTP & SMTP

Deux applications fondamentales
pour le réseau Internet.

File Transfert Protocol

Rapide Historique :

1971 : Première version du protocole définit par le M.I.T.

1973 : Première documentation officielle du protocole FTP.

1975 : Evolution de FTP pour pouvoir fonctionner au
dessus de TCP (jusqu'alors FTP utilisé NCP).

1982 : Finalisation de la définition du rôle de FTP : « *Le File
Transfert Protocol* est désormais définit comme un
protocole de transfert de fichier entre des hôtes d'un
ARPANET, afin de profiter de l'utilisation d'une
capacité de stockage de données distante »

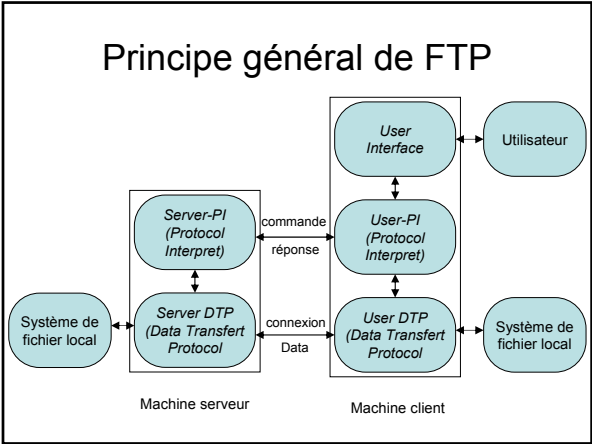
File Transfert Protocol

Un protocole d'échange de fichier « au
dessus » de TCP :

⇒ Définition d'un client et d'un serveur

Le client FTP est la machine de l'utilisateur. Le serveur
est la machine sur laquelle est placée le système de
fichier.

⇒ Prévu pour être exploité par l'intermédiaire de clients
dédiés, mais disponible à une exploitation directe (via un
client `telnet`).



Etablissement du canal de contrôle

Client « localhost » → **Canal de contrôle** → **Serveur FTP « pater.lifl.fr »** (ports 20, 21)

```
> telnet pater.lifl.fr 21
220 G6 FTP Server ready ...
USER toto
331 Password required for toto.
PASS otot
230 User toto logged in.
HELP
214-Supported Commands :
214-PORT STOR APPE RETR CWD
214-PWD  XPWD USER PASS LIST
214-NLST TYPE SYST QUIT DELE
214-SIZE REST RNFR RNTD XMKD
214-MKD  RMD ABOR PASV NOOP
214-CDUP SITE HELP STAT STOU*
214-MDTM STRU SMNT XCUP ACCT
```

Fonctionnement du canal de contrôle :

- ⇒ connexion de type telnet ;
- ⇒ le serveur FTP attend des connexions clients ;
- ⇒ les échanges respectent un ordre établi :
 1. le serveur attend des commandes : format ASCII, 4 caractères + arguments.
 2. le serveur retourne une réponse :
 - 3 chars formant un nombre : code retour
 - 1 char ' ' ou '-' format fin de la réponse ou pas
 - x chars commentaires « lisibles ».
- ⇒ le client termine les échanges en clôturant la socket, ou avec une commande LOUGOUT.

Le canal de contrôle

Client « localhost » → **Canal de contrôle** → **Serveur FTP « pater.lifl.fr »** (ports 20, 21)

Commandes de gestion du canal de contrôle :

USER <username>
Définir l'utilisateur de la session en cours.

PASS <password>
Envoyer le mot de passe associé à l'utilisateur.

REIN
Réinitialiser la session en cours, sans perdre la connexion.

QUIT
Termine la connexion en cours.
Le serveur rompt la Socket.

Etablissement du canal de données...

Client

« localhost »

yyy

xxx

canal de données

canal de contrôle

20

21

Serveur FTP

« pater.lifl.fr »

Cas par défaut :

Paramétrage du canal de donnée :
PORT 134,206,10,153,20,0
200 Port command successful.

Ouverture du canal de donnée :
Lorsque le canal de donné doit être ouvert, il l'est sous l'initiative du serveur.
Dans ce cas le serveur ouvre le port et l'IP indiqué par le client via la commande PORT.

Fermeture :
Sur l'initiative du serveur, lorsque les transferts associées à la commande en cours sont terminés.

Etablissement du canal de données...

Client

« localhost »

yyy

xxx

canal de données

canal de contrôle

zzz

21

Serveur FTP

« pater.lifl.fr »

Mode passif :

Paramétrage du canal de donnée (demande de mode passif) :
PASV
227 Entering Passive Mode (127,0,0,1,55,12).

Ouverture du canal de donnée :
Le canal de donné est ouvert sur l'initiative du client. Le port que le client doit ouvrir est indiqué dans la réponse à la commande PASV. Une fois le canal ouvert, le serveur peut transférer les données associées à la commande suivante (si cette commande déclenche des données sortantes...).

Fermeture :
Sur l'initiative du serveur, lorsque les transferts associés à la commande en cours sont terminés.

Le canal de contrôle

Client

« localhost »

yyy

xxx

canal de données

canal de contrôle

zzz

21

Serveur FTP

« pater.lifl.fr »

Commandes du paramétrage des transferts :
PORT <IP1>,<IP2>,<IP3>,<IP4>,<PORT1>,<PORT2>
Configurer la cible d'émission pour le canal de contrôle avec :
• IP, les 4 octets de l'adresse IP de la cible ;
• Port, deux nombres entre 0 et 255 qui donne le port sur 16bits.

PASV
Positionne le serveur en mode passif. Il retourne l'adresse et le port sur lequel il attend une connexion du client.

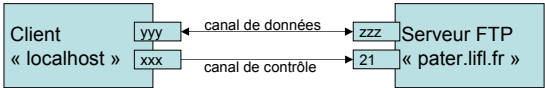
TYPE <param>
Définit le type d'échange réalisé via la socket de donnée :
<param> = **A** ASCII, **E** EBCDIC, **I** Image.

MODE <param>
Détermine le monde d'encodage des données.
<param> = **s** Flux, **B** Block, **c** Compressé.

par Gilles Grimaud

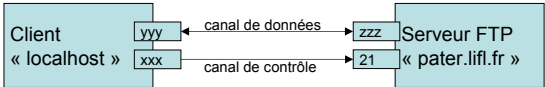
66

Exploitation d'un serveur FTP



Commandes de transfert FTP :
RETR <filename>
Déclanche la transmission par le serveur du fichier <filename> sur le canal de données.
STOR <filename>
Déclanche la réception d'un fichier qui sera enregistré sur le disque sous le nom <filename>. Si un fichier avec le même nom existe déjà il est remplacé par un nouveau avec les données transmises.
APPE <filename>
Déclanche la réception d'un fichier qui sera enregistré sur le disque sous le nom <filename>. Si un fichier avec le même nom existe déjà, les nouvelles données lui sont concaténées.
REST <offset>
Redémarrage en cas d'échec d'un transfert précédent. L'offset précise le numéro du dernier octet reçu.
ABOR : abandon d'un transfert en cours.

Exploitation d'un serveur FTP



Commandes de gestion du système de fichier distant et du serveur FTP :
PWD : impression du répertoire courant.
LIST : catalogue du répertoire courant (canal donnée).
NLST : catalogue succinct (canal donnée).
CWD <repname> : changement de répertoire courant pour <repname>.
MKD <repname> : création du nouveau répertoire <repname>.
RMD <repname> : suppression du répertoire <repname>.
DELE <filename> : suppression du fichier <filename>.
RNFR <filename1> : définit le nom actuel d'un fichier à renommer.
RNTO <filename2> : définit le nouveau nom d'un fichier à renommer.
STAT : status courant de la session FTP.
STAT <repname> : équivalent à LIST mais réponse sur le canal de contrôle.
HELP : affiche l'aide sur les opérations du site.
NOOP : no operation.

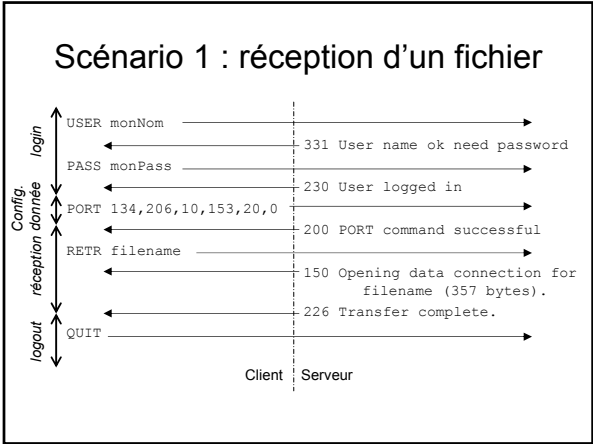
Code de retour

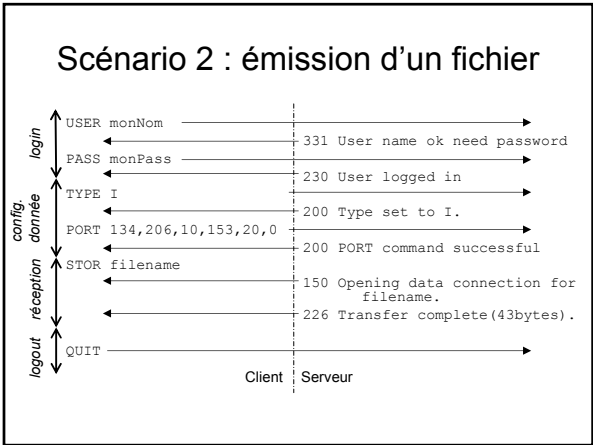
Les codes de retour :

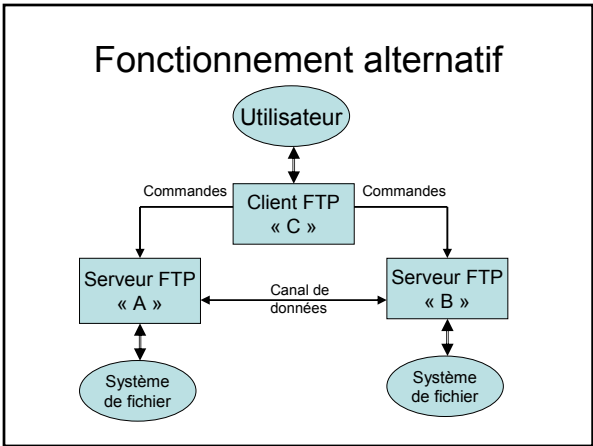
1yz : réponse positive préliminaire	x0z : erreur de syntaxe
2yz : réponse positive définitive	x1z : réponse contenant des informations
3yz : réponse positive intermédiaire	x2z : réponse vis à vis de la connexion
4yz : réponse négative transitoire	x3z : identification et authentification
5yz : réponse négative définitive	x4z : non spécifié
	x5z : système de fichier

La troisième digit raffine la description d'erreur, exemple :

- 501 Erreur de syntaxe, commande non connue.
- 502 Erreur de syntaxe dans les paramètres/arguments.
- 503 Erreur de syntaxe, commande non implémentée.
- 504 Erreur de syntaxe, mauvaise séquence de commandes.
- 505 Erreur de syntaxe, commande non implémentée pour ce param.







Simple Mail Transfert Protocol

Rapide Historique :

1971 : FTP utilisé en temps que support d'échange de mail.

1982 : Première documentation officielle du protocole
SMTP : RFC 821 & RFC 822.

1986 : Normalisation de X400/CCITT devenu MOTIS/ISO

Simple Mail Transfert Protocol

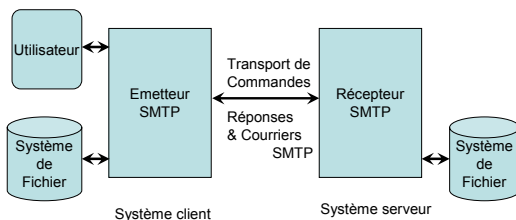
Protocole d'échange de messages
électroniques indépendant du protocole de
transport sous jacent.

N.B. : Pour les réseaux IP, SMTP est implanté au
dessus de TCP, port 25.

⇒ Un **serveur SMTP** est une machine « cible » qui se
présente comme un « bureau de poste » vis à vis de
clients SMTP.

⇒ Protocole directement accessible vis `telnet`.

SMTP : principes fondateurs



Récepteur SMTP :
⇒ bureau de poste ;
⇒ centre de tri.

SMTP : les commandes de base

HELO <domaine> : Initialisation de la session SMTP
MAIL FROM:<route-inverse> : déclaration de l'émetteur du mail
RCPT TO:<route-directe> : déclaration du destinataire du mail
DATA : initialisation de la séquence de saisie des données
RSET : initialisation de la séquence de saisie des données
SEND FROM:<route-inverse> : message direct plutôt que postage.
SOML FROM:<route-inverse> : message direct OU postage.
SAML FROM:<route-inverse> : message direct ET postage.
VRFY <chaîne> : vérification de l'existence d'un destinataire
EXPN <chaîne> : extraction des destinataires inscrits dans une liste de diffusion
HELP [<chaîne>] : demande d'aide (éventuellement sur une commande)
NOOP : aucune opération
QUIT : clôture de la session SMTP
TURN : demande d'inversion des rôles d'émetteur et de récepteur

Les réponses respectent un format comparable à celui de FTP.

Scénario 1 :
validation d'une adresse mail

>telnet monserveursmtp 25	←	220 lifl.lifl.fr ESMTP Sendmail 8.9.3/jtpda-5.3.3 ready at Mon 22 Apr 2002 11:36:22 +0200<EOL>
HELO pater.lifl.fr<EOL>	→	250 lifl.lifl.fr Hello pater [134.206.10.153], pleased to meet you<EOL>
VRFY gigrimaud<EOL>	→	550 gigrimaud... User unknown<EOL>
VRFY grimaud<EOL>	→	250 Gilles.Grimaud <grimaud@lifl.lifl.fr><EOL>
QUIT<EOL>	→	221 lifl.lifl.fr closing connection<EOL>
	Client : Serveur	

Scénario 2 : envoi d'un message

	←	220 lifl.lifl.fr ESMTP Sendmail 8.9.3/jtpda-5.3.3 ready at Mon 22 Apr 2002 11:36:22 +0200 <EOL>
HELO pater.lifl.fr<EOL>	→	250 lifl.lifl.fr Hello pater [134.206.10.153], pleased to meet you <EOL>
MAIL FROM:grimaud@lifl.fr<EOL>	→	250 grimaud@lifl.fr... Sender ok <EOL>
RCPT TO: grimaud@lifl.fr<EOL>	→	250 grimaud@lifl.fr... Recipient ok <EOL>
RCPT TO:jean@lifl.fr<EOL>	→	250 jean@lifl.fr... Recipient ok <EOL>
DATA<EOL>	→	354 Enter mail, end with "." on a line by itself <EOL>
Bla bla bla bla<EOL>	→	
Bla bla bla bla<EOL>	→	
.<EOL>	→	250 LAA21250 Message accepted for delivery <EOL>
QUIT<EOL>	→	221 lifl.lifl.fr closing connection <EOL>

Chemin

entête SMTP

Décl.

corps

1ère Partie

2ème Partie

Corps d'un message SMTP

Return-Path: <pieter@cc.uttente.nl>
Received: from malonne.lifl.fr by lifl.lifl.fr for <grimaud@lifl.fr>
Received: from netlx010.civ.uttente.nl by malonne.lifl.fr for <grimaud@lifl.fr>
Received: from utiw32.cc.uttente.nl by netlx010.civ.uttente.nl; <grimaud@lifl.fr>
Message-ID: <00bd01c1b62187d2b99f08830f59828@utiw32>
From: "Pieter Bartel" <pieter@cc.uttente.nl>
To: "Gilles Grimaud" <grimaud@lifl.fr>
Subject: Re: Strategic Roadmap for Smart Card Research
Date: Fri, 15 Feb 2002 14:05:48 +0100
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="-----_NextPart_000_0344_01C1D021.24302E00"
Status:
This is a multi-part message in MIME format.
-----_NextPart_000_0344_01C1D021.24302E00
Content-Type: text/plain;
charset="iso-8859-1"
Content-Transfer-Encoding: 7bit
Dear RESET partner,
Can I please have your comments on the table, no later than monday morning?
Please don't forget to send your A3 to ERCIM.
I attach the latest draft of the proposal for your information.
--pieter
-----_NextPart_000_0344_01C1D021.24302E00
Content-Type: application/msword;
name="RESET Part B-v0.4.doc"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename="RESET Part B-v0.4.doc"
QMSR4KXGGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAAAAAAAAAAAAAAAAhgEAAAAAAAAA
EAAAIAEAAAEAAAD+////AAAAAH4BAAB/AQAjAEAAICBAAD//////////
AAAAJBByAnF7BAK1YAACtWAAKpMAAGAFADJAAAAAAAAAAAAAAAAAAAAAbwAAAAAAAAAD//wEAAAAA

Supports de références sur les
protocoles de l'Internet : RFC¹

Disponibles sur le Web :
<http://www.ietf.org/rfc>
Tr. français <http://abcdrfc.free.fr>

RFC relatifs à FTP :
RFC 765, dernière version : **RFC 959**

RFC relatifs à SMTP :
RFC 772, RFC 780, RFC 788,
dernière version : **RFC 821 & RFC 822**

1 - RFC : *Request For Comments*

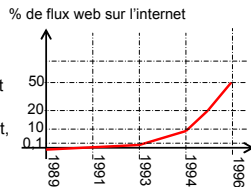
Le Web

Présentation du langage HTML et
du protocole HTTP.

Le web

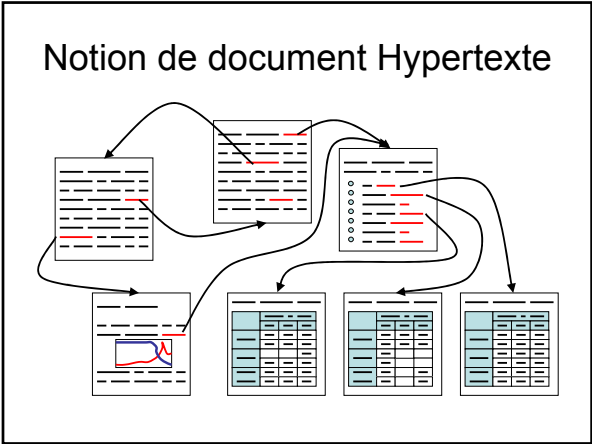
Rapide historique :

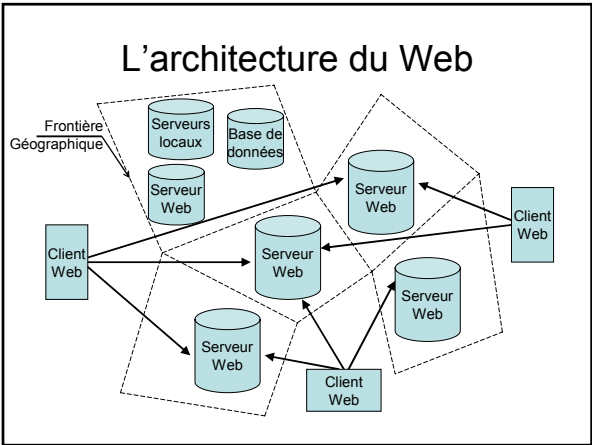
1989 : première note relative au web «
hypertexte et ke Cern » qui jette les
base du World Wide Web.
1990 : disparition du réseau ARPAnet et
apparition de l'Internet.
1991 : premier serveur web sur l'Internet,
celui du CERN.
1993 : première version « libre » d'un
navigateur web (celui utilisé par le
CERN).
1994 : création du WWW Consortium par le CERN, le MIT et l'INRIA.
1995 : Java s'intègre au Web (et au navigateur Netscape).
1996 : début de l'exploitation commerciale du Web.



Objectif initial du web

Pourquoi le Web et inventé au CERN ?
(Centre Européen de Recherche Nucléaire)
⇒ un grand nombre d'universités et de
laboratoires européen.
⇒ des documents produits sur une grande variété
de site, dans une grande variété de formats.
⇒ volonté de constituer une base unique, et
globale composée de l'ensemble de ces
documents qui sont néanmoins éparpillés sur le
réseau.





Notion d'hyperlien et Uniform Ressource Locator

`http://www.unsite.com:80/unRep/unDoc.cgi?arguments`
← protocole ← utilisateur ← Adresse du serveur ← port ← Chemin du document ← arguments →

Protocoles permettant de transférer des données sur le réseau :

- Serveur Web distant : `http://`
- Serveur FTP distant : `ftp://`
- Système de fichier local : `file://`
- Adresse Mail : `mailto://`
- Session interactives : `telnet://`
- Serveur de forum : `:news://`
- Serveur de forum nntp : `:nntp://`
- Gopher : `:gopher://`
- ...

Utilisateur et mot de passe local :
[<nomUtilisateur>[:<motDePasse>]@]
Exemple : `ftp://toto:otot@ftp.free.fr/img1.jpg`

Adresse du serveur :
Adresse symbolique convertible en adresse IP via un DNS.

Chemin du document :
Chemin symbolique à partir du répertoire racine de la connexion.

Arguments :
Chaîne de texte passée en argument par le client au serveur pour qu'il retourne un document paramétrable.

Documents hypertexte et Langage HTML

Objectif :

Présentation d'un texte « formaté » qui
contient différents supports de média dont :

- Du texte, des tables, des listes, ...
- Des images, des sons, des vidéos, ...
- Des fichiers binaires ;
- Des références vers des hyperliens.

Documents hypertexte et Langage HTML

Principe du langage :

Section de textes à afficher structurés à l'aide de *tag HTML*
notés `<tagName> </tagName>`.

Le premier tag (`<tagName>`) définit le début de la zone de
texte sur laquelle il porte, le second (`</tagName>`) définit
la fin de la zone de texte concernée.

Les tags HTML peuvent être encapsulés les uns dans les
autres.

Exemple :

```
<p> texte affiché à l'intérieur d'un  
paragraphe.</p>
```

Structure générale d'une page HTML

```
<html>  
<head>  
  <title>toto</title>  
  <meta http-equiv="Content-Type"  
    content="text/html; charset=iso-8859-1">  
  <meta name="keywords" content="motclef1 motclef2">  
  <meta name="description"  
    content="Résumé de la page...">  
  <meta http-equiv="refresh"  
    content="13;URL=http://www.lifl.fr">  
</head>  
<body bgcolor="#FFFFFF" text="#000000">  
  bla bla bla...  
  bla bla bla...  
  bla bla bla...  
  bla bla bla...  
</body>  
</html>
```

Tags clefs du langage HTML

Format du texte :

```
<h1> titre du document </h1>
<h2> sous-titre </h2>
<h3> titre de section </h3> (jusqu'à <h6> </h6>)

<p> Corps d'un paragraphe, paragraphe qui peut contenir
plusieur lignes, comme tout texte entre deux tags. </p>
<br> pour passer &agrave; la ligne à l'intérieur d'un
paragraphe.

<i> texte en italique </i>
<b> texte en gras </b>
<font color="#FF0000" face= "Courier New, Courier, mono">
texte avec une couleur et une fonte
particuli&agrave;re</font>

<a href="http://www.lifl.fr"> texte associ&eacute; à un
hyperlien vers www.lifl.fr .<a>
```

Tags clefs du langage HTML

Liste de puces et listes de points :

Exemple de code :

```
<p>Liste de puces et de points :</p>
<ul>
<li> puce numero 1 ;</li>
<li> puce numero 2 ;</li>
<li> puce numero 3 dont deux
points principaux :
<ol>
<li> point 3.1 ;</li>
<li> point 3.2 .</li>
</ol>
</li>
<li> puce numero 4.</li>
</ul>
```

Rendu par le navigateur :

Liste de puces et de points :

- puce numero 1 ;
- puce numero 2 ;
- puce numero 3 dont deux points principaux :
 - 1. point 3.1 ;
 - 2. point 3.2 .
- puce numero 4.

Tags clefs du langage HTML

Tableaux est tables HTML

Exemple de code :

```
<p>Exemple de table :</p>
<table width="400" height="92" border="1">
<tr>
<td>T (1,1)</td> <td>T (2,1)</td>
<td>T (3,1)</td>
</tr>
<tr>
<td rowspan="2">T (1,2) et T (1,3)</td>
<td>T (2,2)</td> <td>T (3,2)</td>
</tr>
<tr>
<td colspan="2" rowspan="2">T (2,3) et T (3,3)</td>
</tr>
<tr>
<td colspan="2" rowspan="2">T (2,3) et T (3,3)</td>
</tr>
</table>
```

Rendu par le navigateur :

Exemple de table :

T(1,1)	T(2,1)	T(3,1)
T(1,2) et T(1,3)	T(2,2)	T(3,2)
	T(2,3) et T(3,3)	

HTML est les zones d'images

Placer une image dans un texte :

```

```

Placer un programme java (« applet ») dans le document :

```
<applet code="testFTP2.class" codebase =  
"http://www.lifl.fr/applets" width="320" height="200">  
  <param name="arg1" value="13">  
  <param name="arg2" value="texte dans une String">  
</applet>
```

Un protocole pour transporter des documents Hypertextes

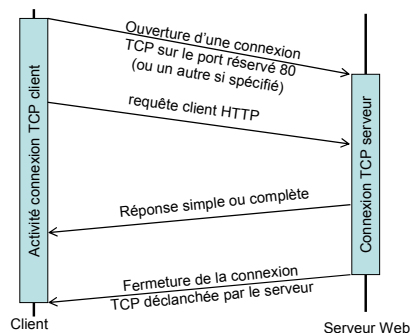
Pourquoi définir un nouveau protocole :

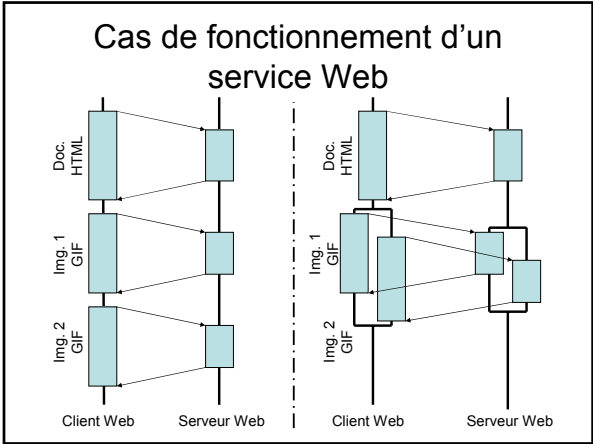
Initialement, permettre que n'importe quel serveur connecté au réseau IP puisse délivrer des documents textes de la manière la plus simple possible aux clients qui les réclament.
⇒ HTTP 0.9

Aujourd'hui HTTP est pensé comme un support pour la transmission de documents distribués et multimédia à travers un système d'information multi-utilisateurs.

HTTP : *HyperText Transfert Protocol* : Protocole de transfert d'hypertextes

HTTP sur TCP/IP : architecture des serveurs Web





les requêtes du client HTTP

Format des requêtes :

Trois requêtes de base :

```
GET <SP> <URL> <SP> [HTTP/1.0 <CRLF>
("Authorization: Basic QWxhZGRpbjpvY2FtZQ==" | // utilisateur / mot de passe
"From:" address_mail | // adresse de l'utilisateur
"If-modified-since:" date | // notification 2 changement ?
"Refered:" URL | // source du lien.
"User-Agent:" id de client Web] ] // nom du client (navigateur)

HEAD <SP> <URL> <SP> [HTTP/1.0 <CRLF>
("Authorization: Basic QWxhZGRpbjpvY2FtZQ==" | // utilisateur / mot de passe
"From:" address_mail | // adresse de l'utilisateur
"Refered:" URL | // source du lien.
"User-Agent:" id de client Web] ] // nom du client (navigateur)

POST <SP> <URL> <SP> [HTTP/1.0 <CRLF>
("Authorization: Basic QWxhZGRpbjpvY2FtZQ==" | // utilisateur / mot de passe
"From:" address_mail | // adresse de l'utilisateur
"If-modified-since:" date | // notification 2 changement ?
"Refered:" URL | // source du lien.
"User-Agent:" id de client Web | // nom du client (navigateur)
"Content-Type:" TypeDeMedia | // type des données transmises
"Content-Length:" TailleDuContenu | // taille des données
"Content-Encoding:" typeDeCodage ] <CRLF>
[OCTET] // exemple x-gzip
// corps des données transmises
```

les requêtes du client HTTP

Format des requêtes :

Autres commandes parfois implantées :

PUT <SP> URL <CRLF> document au format MIME

DELETE <SP> URL <CRLF> HTTP/1.0 <CRLF> en-têtes

Requêtes génériques :

Méthode <SP> URL <SP> HTTP/1.0 <CRLF> en-têtes et document MIME

Format des réponses du serveur Web

Pour les serveurs « HTTP 0.9 » :
Document HTML

Depuis « HTTP 1.0 » :
"HTTP/" une DIGIT "." <SP> trois chiffres <SP> raison <CRLF>
en-tête générale <CRLF>
en-tête réponse <CRLF>
en-tête entité <CRLF>
corps entité

Les trois chiffres désignent un code d'erreur.

En-tête entité pour un document HTML :
Content-Type:text/html
Le corps est alors un document HTML.

Codes d'erreur HTTP

Listes des erreurs prédéfinies :

Code ; Raison	description
"200" ; OK	OK
"201" ; Created	Créé
"202" ; Accepted	Accepté
"204" ; No Content	pas de contenu
"301" ; Moved Permanently	Changement définitif
"302" ; Moved temporarily	Changement temporaire
"304" ; Not modified	non modifié
"400" ; Bad request	requête incorrecte
"401" ; Unauthorized	non autorisé
"403" ; Forbidden	Interdit
"404" ; Not Found	Non trouvé
"500" ; Internal server error	Erreur interne du serveur
"501" ; Not implemented	Non implanté
"502" ; Bad Gateway	Erreur de routeur
"503" ; Service Unavailable	Indisponible

Ou autres code 3 digits + texte retour d'erreur

Entête de réponse d'un serveur HTTP

Entête HTTP

↑

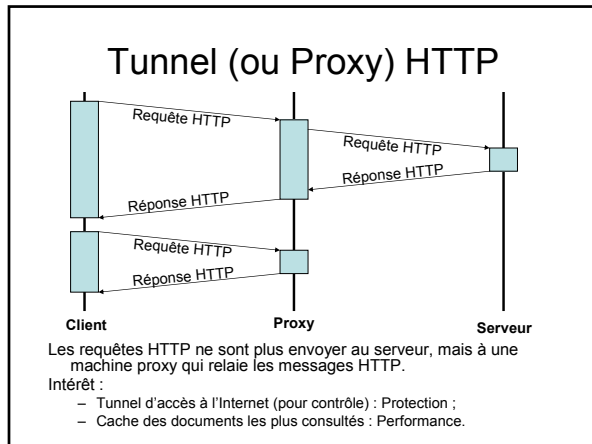
↓

corps

```
HTTP/1.0 <SP> 200 <SP> OK <CRLF>
Server: <SP> Apache/1.3.22 (Unix) PHP/4.0.6 <CRLF>
Date: <SP> Sun, 28 Apr 2002 20:57:46 GMT <CRLF>
Expires: <SP> Tue, 08 May 2000 13:41:16 GMT <CRLF>
Last-Modified: <SP> Sun, 02 Feb 2002 20:57:46 GMT <CRLF>
Content-Type: <SP> text/html <CRLF>
Content-Length: <SP> 2717 <CRLF>

<html>
<head>
  <title>toto</title>
  <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1">
  <meta name="keywords" content="motclef1 motclef2">
  ...
```

Le corps est une suite d'octets. Si on à dans l'en-tête la ligne content-Type: image/gif
Le corps du document sera la suite des octets qui constitue l'image au format GIF.



Documentation relative au web

- Protocole HTTP 1.0 et HTTP 1.1
RFC 1945 et RFC 2616
- grammaire des URL, URI et URN :
RFC 1738
- Le Web et le Langage HTML :
<http://www.w3c.org>
- Validateur de code en ligne :
<http://validator.w3.org/>

Distribution d'applications via le WEB

Documents actifs, applets
& Scripts cotés serveur, Servlet

Introduction à la notion de documents actifs

Produire des documents en fonction de différentes
données.

- Objectif :
- personnaliser les documents en fonction des utilisateurs ;
 - présenter des données extraites d'une requête dans une base de donnée ;
 - Déclancher des opérations à distance et obtenir des résultats ;
 - Afficher des comptes rendues de calculs, de mesure en temps réel.

- Source de données :
- documents construit en fonction des paramètres d'une requête ;
 - documents construit en fonction d'une base de données ;
 - documents construit en fonction d'un outil de mesure externe ;
 - documents construit en fonction de résultats externes au système.

Formulaires HTML

Objectif : permettre à l'utilisateur
d'envoyer des informations vers le
serveur.

- Exemple :
- Chaîne de caractère,
 - nombres,
 - choix parmi une liste,
 - cases à cocher,
 - fichiers,
 - choix d'un point sur une image...

Transfert des informations saisies via le
bouton « soumettre ».

2 méthodes de transfert :
GET & POST



Déclaration d'un formulaire en HTML

Déclaration du formulaire :

```
<form name="form2" method="get" action="http://anURL">
```

Corps du formulaire

```
</form>
```

```
<form name="form1" method="post" action="http://anURL"
  enctype="multipart/form-data">
```

Corps du formulaire

```
</form>
```

Method="get": Les arguments fournis par l'utilisateur sont passer directement dans l'URL qui suit la commande GET (la taille des arguments doit rester modeste)

Method="post": Les arguments fournis par l'utilisateur sont passer dans le corps de la requête, en utilisant par exemple le format MIME (cf. contenu d'un courrier électronique).

Formatage des requêtes HTTP : via la commande GET

Le formulaire en HTML :

```
<form name="f1" method="get" action="http://monsite.com/monprog.cgi">
<p> nom <input type="text" name="lenom" value="un nom">
      prenom <input type="text" name="leprenom" value="un prenom"> </p>
<p>
  <input type="submit" name="Submit" value="DoIt">
  <input type="submit" name="Cancel" value="Cancel">
  <input type="reset" name="Reset" value="Remise à Zero">
</p>
</form>
```

Requête envoyée au serveur sur click de soumettre :

```
GET /d?leprenom=gilles&lenom=grimaud&Submit=DoIt HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

Règle de formatage d'une requête « GET »

Les arguments du formulaire commencent après le "?" Chaque argument est identifié par un couple : nomDArgument=ValueDeLArgument
Le caractère "&" est utilisé pour séparer les arguments.

- Pour les champs de saisie de texte :

saisie de texte:

Code HTML: <input type="text" name="arg1" value="unNom">

Format dans la requête HTTP: ...?arg1=unNom&...

- Pour les listes de valeur :

Code HTML: <select name="arg2">

<option value="OUI">oui OK!</option>

<option value="NON">Non Merci...</option>

<option value="BLANC">Blanc</option>

</select>

Format dans la requête HTTP: ...&arg2=OUI&...

Case 1: ☐

Case 2: ☒

Case 3: ☐

- Pour les cases à cocher :

Code HTML: <input type="checkbox" name="Arg3" value="C1">

Format dans la requête HTTP: ...&ChoixA=C1&...

Règle de formatage d'une requête « GET »

- Pour les champs de saisie de texte :

Code HTML :

```
<input type="radio" name="Arg4" value="C1">
```

Format dans la requête HTTP :

```
...?arg4=C1&...
Choix 2- C
Choix 3- #
```

- Pour un point sur une image :

Code HTML :

```
<input type="image" border="0" name="arg5"
width="200" height="184" src="monImage.gif">
```

Format dans la requête HTTP :

```
...&arg5.x=110&arg5.y=13&...
```

- Pour les boutons (autre que reset) :

Code HTML :

```
<input type="submit" name="arg6" value="DoIt">
```

Format dans la requête HTTP :

```
...&arg6=DoIt
```

Règle de formatage d'une requête « POST »

⇒ Limitation de la commande GET (doit pouvoir être contenu dans une URL) inadapté aux requêtes volumineuses (exemple : transfert d'un fichier)

Déclarer un formulaire selon la méthode «post» :

```
<form name="f1" method="POST"
action="http://monsite.com/monprog.cgi"
enctype="multipart/form-data">
```

Contenu du formulaire équivalent à celui d'un formulaire GET sauf qu'il accepte les fichiers :

```
<input type="file" name="unFichier" maxlength="60">
```

Formatage des requêtes HTTP : via la commande POST

```
POST /bidon.cgi HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, */*
Content-Type: multipart/form-data; boundary=-----
```

```
7d29cf5045a
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Length: 1073
```

```
-----7d29cf5045a
Content-Disposition: form-data; name="arg1"
Champs de saisie de texte: Nom de l'utilisateur
```

```
Nom de l'utilisateur
-----7d29cf5045a
Content-Disposition: form-data; name="arg2"
Choix parmi une liste: [oui en effet]
```

```
OUI
-----7d29cf5045a
Content-Disposition: form-data; name="arg3"
Case à cocher:
Case 1: ☐
Case 2: ☒
Case 3: ☐
```

```
C2
-----7d29cf5045a
Content-Disposition: form-data; name="arg4"
Liste de choix:
Choix 1- C
Choix 2- C
Choix 3- #
```

3

Formatage des requêtes HTTP :
via la commande POST

```
----- 7d9cf5045a
Content-Disposition: form-data; name="arg5"; filename="C:\Demos\fr019_party\file_id.diz"
Content-Type: text/plain

.farbrausch consumer consulting
fr (R9: poem to a horse- party version
.64k demo production
.mekkasymposium 2002
.code      chaos .graphics   fiver2
.music     rp     .synth      kb
.packers   rpg
.wait for the final-   ww.farb  rausch.com

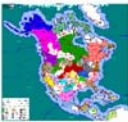
----- 7d9cf5045a
Content-Disposition: form-data; name="imageField.x"

108
----- 7d9cf5045a
Content-Disposition: form-data; name="imageField.y"

135
----- 7d9cf5045a -
```

C:\Demos\fr019_party\ Browse

Fichier indiqué



Traitement de la requête sur le
Serveur

Déclancher une action, un calcul, ...

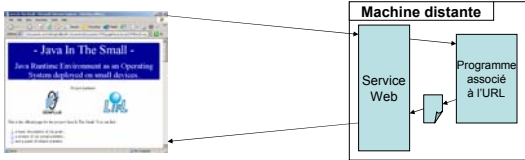
Exécuter un programme

- CGI, ISAPI ;
- Servlets ;
- Documents contenant des scripts exécuter par le serveur.

Retourner un document, produit par l'exécution de la requête.

- Une page HTML ;
- Une image ;
- Un renvoi vers une autre page...

Production de documents



Documents actifs CGI

```
#include <stdio.h>
int getNextCount();
int main(int argc, char **argv)
{ printf("<html>\n\n");
  printf("<head> <title> mon titre </title> </head>\n\n");
  printf("<body> <h1> Hello World </h1> \n\n");
  printf("<p> - %d</p> </body>\n\n", getNextCount());
  printf("</html>\n\n");
  return 1;}
```

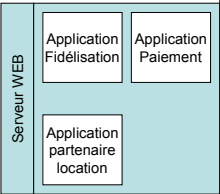
Intégrer les applications cotés serveur dans un cadre de travail

- Dépasser le schéma applicatif :
Un serveur Web pour une application

Intégrer les applications cotés serveur dans le Serveur Web lui-même :

Pour améliorer les performances du serveur :
Par exemple « DLL ISAPI sous IIS »

Pour faciliter le déploiement / administration et la maintenance d'applications WEB :
Exemple : JSP & Servlets sous TomCat Apache.



Production de documents via des Servlets

Déclaration d'une Servlet dans le fichier de mapping web.xml :

```
<web-app>
  <servlet>
    <servlet-name>MaServlet1</servlet-name>
    <servlet-class>coreservlets.entryClass</servlet-class>
    <init-param>
      <param-name>Message</param-name>
      <param-value>Bienvenu en ce mois de Mai</param-value>
    </init-param>
    <init-param>
      <param-name>compteur</param-name>
      <param-value>134</param-value>
    </init-param>
  </servlet>
</web-app>
```

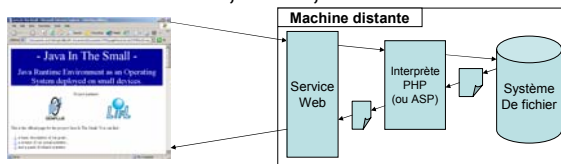
Production de documents via des Servlets

```
Public class EntryClass extends HttpServlet {
  private String message = "pas de message";
  private int compteur = 1;
  public void init() throws ServletException {
    ServletConfig cfg = this.getServletConfig();
    String s = cfg.getInitParameter("message");
    if(s!=null) message = s;
    s = cfg.getInitParameter("compteur");
    try { compteur = Integer.parseInt(s); } catch (NumberFormatException e) {}
  }
  public void doGet(HttpServletRequest req,HttpServletResponse res) {
    Response.setContentType("text/html");
    PrintWriter pw = res.getWriter();
    pw.println("<html> <head> <title> EntryClass document </title> </head>");
    pw.println("<body> <h1> The EntryClass message </h1> <p>");
    for(i=0;i<compteur;i++)
      pw.println(message);
    pw.println("</p> </body>");
  }
}
```

Cycle de vie d'une Servlet

- **Init**
 - Exécuter une fois lorsque le serveur web charge la servlet dans son espace de travail. Elle devrait être redéfini par la sous-classe de `Servlet` pour initialiser l'application.
- **Service**
 - Cette méthode est appelée (dans une nouvelle thread) pour chaque réception d'une requête HTTP. Elle décode les requêtes et redistribue les messages vers les méthodes « `doGet` », « `doPost` », ... (ne pas surcharger cette méthode sauf pour décodage des requêtes très particulières.)
- **doGet, doPost, ...**
 - Il faut surcharger ces méthodes pour définir les réponses du document actif aux requêtes http transmises par le client.
- **Destroy**
 - Appelée lorsque le serveur « retire » la servlet de son environnement d'exécution. Il faut surcharger cette méthode pour « libérer » les ressources monopolisées par la servlet...

Plus simple et plus facile à produire : PHP, ASP, JSP...



• Extrait de code PHP

```
<html>
<head> <title> MonTitre </title> </head>
<body> <h1> Hello World <h1>
<p> - <? getNextCount() ?> (Ou <? echo getNextCount(); ?>)
</body>
</html>
```

Problème liés aux génération de documents coté serveur :

Surcharge de calcul :

- création de processus CGI ;
- traitement de la requête...

Surcharge de la bande passante :

- Réception de requêtes invalides ;
- Transfert de données non compactes...

⇒ Répartir la charge de travail.

Quand le serveur devient un info centre

Architecture d'un Info Centre

Bande passante

Puissance des serveurs

Répartir la charge de travail

Des millions de clients pour quelques serveurs :

- ⇒ minimiser le nombre et la taille des requêtes ;
- ⇒ éviter la transmission de requêtes infondées ;
- ⇒ réduire le nombre de connexions simultanées au(x) serveur(s) ;
- ⇒ Transférer les données sous formes compactes plutôt que complètement formatées (exemple : des listes de valeurs plutôt que des images GIF ou JPG)

Solution :

- ⇒ Déployer des applications sur les clients.

Problème d'hétérogénéité des plateformes, des environnements, ...

JavaScript est les documents réactifs sur le navigateur client

```
***
<SCRIPT language=JavaScript>
<!--
window.onerror = new Function("return false;");
window.onload = init;

var eActiveButton = null;
function init() {
  this.SiteToolBar.onselectstart = new Function("return false;");
}
//-->
</SCRIPT>
***
```

Java et les applets intégrées aux documents HTML

```
...
<applet
  codebase="." code="Clock2.class"
  width=170 height=150
  alt="alternative message" >
  <param name=bgcolor value="000000">
  <param name=unusedp value="Un Texte">
</applet>
...
```

Définir une applet Java

- Hériter de `Java.applet.Applet` qui est une sorte de `java.awt.panel`.
- Redéfinir la méthode `void init()` appelé lorsque l'applet viens d'être chargés.
- Redéfinir la méthode `void start()` appelé à chaque (re)démarrage de l'applet.
- Redéfinir la méthode `void stop()` appelé lorsque le document contenant l'applet est quitté.
- Redéfinir la méthode `void destroy()` appelé lorsque l'applet est retirée du navigateur.
- Redéfinir la méthode `String [] [] getParameterInfo()`

Exemple d'Applet

```
public class Exemples extends Applet implements Runnable {
    Thread timer;
    public void init() {
        try {setBackground(new Color(Integer.parseInt(getParameter("bgcolor"),16));}
        catch (Exception E) { }
    }
    public void paint(Graphics g) { ... }
    public void start() { timer = new Thread(this); timer.start(); }
    public void stop() { timer = null; }
    public void run() { Thread me = Thread.currentThread();
        while (timer == me) {
            try { Thread.currentThread().sleep(100);}
            catch (InterruptedException e) {}
            repaint();}
    }
    public String getAppletInfo() { return "Title: An Applet \n By Me!"; }
    public String[][] getParameterInfo() {
        return {{"bgcolor", "hexadecimal RGB number", "The background color.
        Default is the color of your browser."}};
    }
}
```

Interaction Applet / Navigateur et Applet / Navigateur / Applet

Les applets peuvent interagir (dans un cadre limité par les problèmes de sécurité) avec le navigateur sur lequel elles sont déployées :

```
public void init() {  
    AppletContext aC = this.getAppletContext();  
    ac.showDocument(new URL("une URL"), "target");  
    ac.showStatus("a status text");  
}
```

Elles peuvent interagir entre elles via le navigateur :

```
Applet a = ac.getApplet("leNomDeLApplet");
```

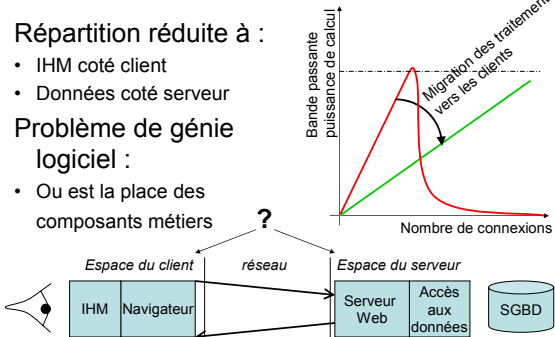
Bénéfice de la répartition des tâches

Répartition réduite à :

- IHM coté client
- Données coté serveur

Problème de génie logiciel :

- Ou est la place des composants métiers



Appel de procédures distantes

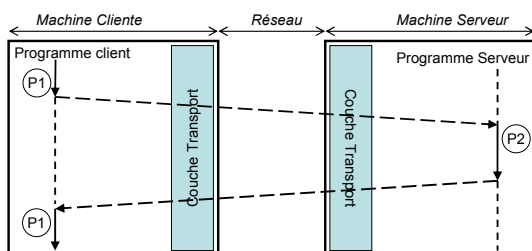
Introduction aux systèmes à base
d'objets distribués,
Illustration sur RMI

Du modèle client/serveur ...

- **Application Client/Serveur :**
Déf. : Application qui fait appel à des services
distants au travers d'un échange de messages
 - Le Client envoie une requête ;
 - Le Serveur retourne une requête.
- **Les clients** sont les programmes qui sollicitent
des services disponibles sur une machine
distante.
- **Le serveur** est le programme qui fournit un
service à un ensemble de clients.

Du modèle client/serveur ...

- Selon le modèle client/serveur, deux
messages au moins sont échangés :



Du modèle client/serveur ...

- Vue du client
 - Client
- Vue du serveur
 - Gestion des requêtes (priorité)
 - Exécution du service (séquentiel, concurrent)
 - Mémorisation ou non de l'état du client

Du modèle client/serveur ...

Exemples d'application client/serveur

- Serveur de fichiers (aufs, nfsd)
- Serveur d'impressions (lpd)
- Serveur de calcul
- Serveur de base de données
- Serveur de noms (annuaire des services)

Du modèle client/serveur ...

- 1 processus par service

1 processus par requête		2 processus pour le service		n processus pour le service	
Processus veilleur	Processus de service	Processus veilleur	Processus de service	Processus veilleur	Processus de service

par Gilles Grimaud

90

Du modèle client/serveur ...

Service sans données persistantes :

- Situation idéale où le service s'exécute uniquement en fonction des paramètres d'entrée
- Modèle client/serveur optimal
 - Pour la tolérance aux pannes
 - Pour le contrôle de la concurrence
- Exemple :
 - le calcul scientifique

Du modèle client/serveur ...

Service avec données persistantes :

- Les exécutions successives manipulent des données persistantes
 - Modification du contexte d'exécution sur le site distant
 - Problème de contrôle de concurrence
 - Difficultés en cas de panne en cours d'exécution
- Exemple :
 - Serveur de fichier réparti (lectures / écritures)

Du modèle client/serveur ...

Service en mode sans état :

- Les différentes requêtes peuvent être traitées sans lien entre elles.
 - Il peut y avoir modification de données globales mais l'opération s'effectue sans lien avec celles qui l'ont précédé.
- Exemple
 - Serveur de fichier réparti : accès aléatoire

Du modèle client/serveur ...

Service en mode avec état :

- Les différentes requêtes sont nécessairement traitées séquentiellement.
 - Il peut y avoir modification de données globales ou pas mais l'opération s'effectue en liaison avec celles qui l'ont précédé.
- Exemple
 - Serveur de fichier réparti : accès séquentiel

... au modèle d'appel de procédure à distance.

Outil idéal pour les applications conçues selon le modèle client / serveur.

- L'opération à réaliser est présentée sous la forme d'une procédure que le client peut appeler. Ce faisant il déclenche l'exécution du traitement associé à cette procédure, mais sur la machine distante.
 - Simplicité (en l'absence de panne)
 - Sémantique identique à celle de l'appel local
- Opération de base
 - Client
 1. doOp(IN ServiceID s, IN Name opName, IN Msg *args, OUT Msg *result)
 - Serveur
 1. getRequest(OUT ServiceID s, OUT Msg *args)
 2. opName(IN Msg *args, OUT Msg *result)
 3. sendReply(IN ServiceID s, IN Msg *result)

... au modèle d'appel de procédure à distance.

- Objectifs
 - Retrouver la sémantique habituelle de l'appel de procédure
 - Sans se préoccuper de la localisation de la procédure
 - Sans se préoccuper du traitement des défaillances
 - Objectifs très difficiles à atteindre
 - Réalisation peu conviviale
 - Sémantique différente de l'appel de procédure même en l'absence de panne.

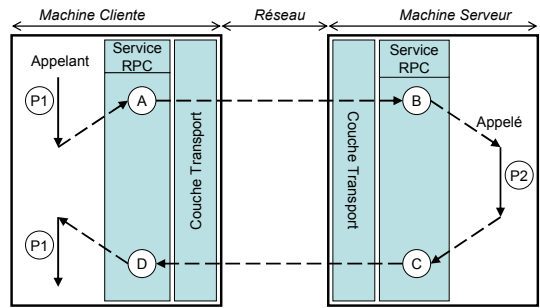
... au modèle d'appel de
procédure à distance.

Les pièges des appels de procédure à distance :

Appel de procédure	Appel de procédure à distance
Appelant et appelé sont dans le même espace de travail ⇒ Environnement d'exécution de l'appelant partagé avec l'appelé. • même mode de pannes ; • appel et retour de procédure considéré comme fiable ; • temps d'appel très faible ; • concurrence des appels dans une minorité de cas.	Appelant et appelé sont dans des espaces virtuels différents : ⇒ Environnement d'exécution de l'appelant distinct de celui de l'appelé. • pannes du client et du serveur « indépendantes » • pannes du réseau de communication ⇒ appel de procédure distante considéré comme non fiable • temps d'appel non négligeable ; • concurrence des appels dans la majorité des cas.

Principe de fonctionnement d'un
appel distant

• Principe de Birrel & Nelson (84)



Rôle des talons

Le talon client
- Stub -

C'est la procédure d'interface
du site client :

- qui reçoit l'appel local ;
- le transforme en appel distant (encodage des arguments dans un message « réseau »)
- attend réception des résultats de l'exécution distante
- décode et retourne la réponse à celui qui a appelé (localement) le Stub.

Le talon serveur
- Skeleton -

C'est la procédure sur le site
serveur :

- Qui reçoit l'appel sous forme de message (décode les arguments)
- Appelle « localement » la procédure serveur
- Encode la réponse fournie par la procédure serveur sous la forme d'un message « réseau ».

RPC

Les problèmes

Traitement des défaillances

- congestion du réseau ou du serveur
 - la réponse ne parvient pas en temps utile
- panne du client pendant le traitement de la requête
- panne du serveur avant ou pendant le traitement de la requête
- erreur de communication

Problèmes de sécurité

- authentification du client
- authentification du serveur
- confidentialité des échanges

Désignation et liaison

Aspect pratiques

- adaptation à des conditions multiples (protocoles, langages, matériels)
- gestion de l'hétérogénéité

RPC

Passage de paramètres

Valeur

- pas de problème particulier

Copie / restauration

- les valeurs des paramètres sont recopiées
- Pas de difficultés majeures
- optimisation des solutions pour RPC

Références

Impossible d'utiliser « l'adresse mémoire » de l'appelant !

Solution pour les refs.

- proposer une référence de l'objet indépendante de l'adresse mémoire.
- L'objet devient lui-même une référence distante pour le serveur qui pourra l'utiliser via des appels de procédures distantes.

Solution insuffisante

pour représenter des références « non objet » (adresse mémoire).
⇒ Seule alternative : mémoire distribuée

Système d'objets distribués

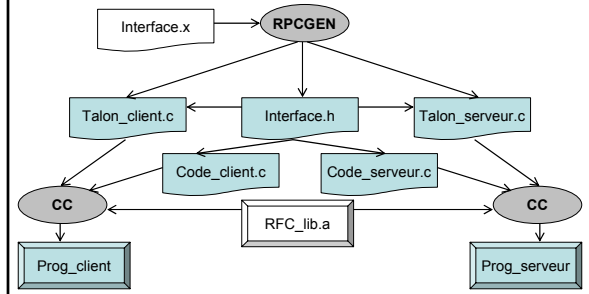
Espace machine A

Espace machine B

Espace machine C

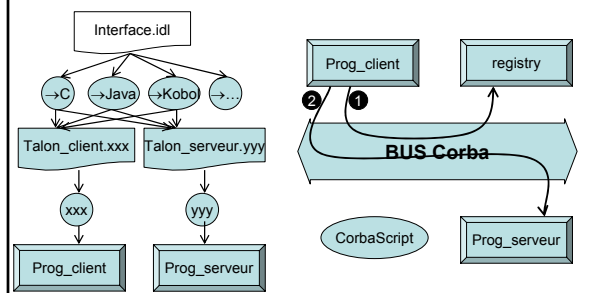
Les premières générations d'outils d'appel à distance

- RPC : langage C / Unix



Corba

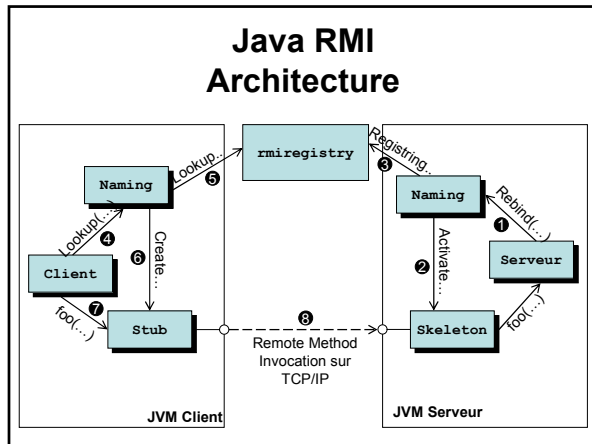
- Support de l'hétérogénéité des langages et des plateformes :
- 1 langage de description du service : IDL
 - 1 consortium garant de la représentation du plus grand nombre : OMG



Java et les applications distribuées Java-RMI

Java possède un RPC orienté objet intégré

- Interaction d'objets situés dans des espaces d'adressage différents sur des machines distinctes.
- Un objet distribué est un objet Java « comme les autres ». Il possède
 - Un proxy : représentant de l'objet coté client...
 - Un skeleton : coté serveur



Java RMI Mode opératoire

Codage

- Description de l'interface du service
- Ecriture du code du serveur implantant l'interface
- Ecriture du client qui utilise l'interface

Compilation

- Compilation des sources (`javac`)
- Génération des stub et skeleton (`rmic`)

Activation

- Lancement du serveur de noms (`rmiregistry`)
- Lancement du serveur
- Lancement du client

Java RMI écriture de l'interface

Simple déclaration d'une interface Java classique.

- L'interface doit être publique
- L'interface distante doit étendre l'interface `java.rmi.Remote`
- Chaque méthode doit déclarer au moins l'exception `java.rmi.RemoteException`
- Les objets passés en paramètre des méthodes doivent :
 - être une sorte d'interface `java.rmi.Remote` le paramètre est alors passé par référence ;
 - Supporter l'interface `java.io.Serializable`, dans ce cas l'objet est passé par valeur (sérialisation / désérialisation)

Exemple Java RMI 1/ création de l'interface

Hello.java

```
public interface Hello extends
    java.rmi.Remote
{
    String sayHello() throws
        java.rmi.RemoteException;
}
```

Exemple Java RMI 2/ création du serveur

HelloServeur.java

```
import java.rmi.*;
import java.rmi.server.*;

public class HelloServeur extends UnicastRemoteObject implements Hello {
    private String msg;

    public HelloServeur(String msg) throws java.rmi.RemoteException {
        super(); this.msg = msg; }

    public String sayHello() throws java.rmi.RemoteException {
        System.out.println("Hello world: " + msg);
        return "Hello world: " + msg; }

    public static void main(String args[]) {
        try {
            HelloServeur obj = new HelloServeur("HelloServeur");
            Naming.rebind("//localhost:8080/mon_serveur", obj);
            System.out.println("HelloServer bound in registry");
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

Exemple Java RMI 2/ création du serveur

HelloClient.java

```
import java.rmi.*;

public class HelloClient {
    public static void main(String args[]) {
        try {
            Hello obj = (Hello)
                Naming.lookup("//localhost:8080/mon_serveur");
            String msg = obj.sayHello();
            System.out.println(msg);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Exemple Java RMI 3/ génération du code

Compilation des sources :

```
javac Hello.java HelloServeur.java HelloClient.java
```

Génère les classes :

```
hello.class  
HelloServeur.class  
HelloClient.class
```

Production des talons associés au serveur :

```
rmic HelloServeur
```

Génère les classes :

```
HelloServeur_Stub.class  
HelloServeur_Skel.class
```

Exemple Java RMI 4/ activation du système

- Trois processus à démarrer :

```
RMIREGISTRY  
>  
> RMIREGISTRY 8080
```

```
JAVA HelloServeur  
> JAVA -Djava.rmi.server.codebase=ftp://localhost/JServ/ HelloServeur  
HelloServer bound in registry  
Hello world: HelloServeur
```

```
JAVA HelloClient  
>  
> JAVA HelloClient  
Hello world: HelloServeur  
>
```

Couche de transport alternative pour Java RMI

Définir une sous-classe de **Socket** et une sous-classe de **ServerSocket** adapté au support de transport visé.

Définir une classe implémentant **RMIClientSocketFactory** et une autre pour **RMI ServerSocketFactory** produisant des sortes de **Socket** et des **ServerSocket** qui seront utilisées par le stub et le skeleton.

Le serveur qui étend **UnicastRemoteObject** et qui implante l'interface RMI "remote" utilise

```
Super(0, // choix d'un port anonyme sinon numero du port  
      new RMIClientSocketFactory(),  
      new RMIServerSocketFactory())
```

Plutôt que
Super()

N.B. : Flexibilité limitée pour la couche de transport utilisée.
