Protocoles applicatifs

Pierre David
pda@unistra.fr

Université de Strasbourg – Master CSMI

2023 – 2024

Plan

De Telnet/FTP à SSH

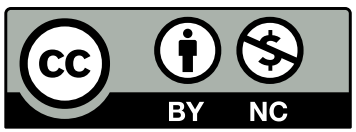
Courrier électronique

BOOTP et DHCP

RPC/XDR et NFS

X-Window

Licence d'utilisation



Avertissement

Un certain nombre des protocoles présentés ici ne sont plus utilisés aujourd'hui.

Ils constituent cependant des exemples intéressants dont les idées peuvent être reprises dans de nouveaux protocoles.

Plan

De Telnet/FTP à SSH

Courrier électronique

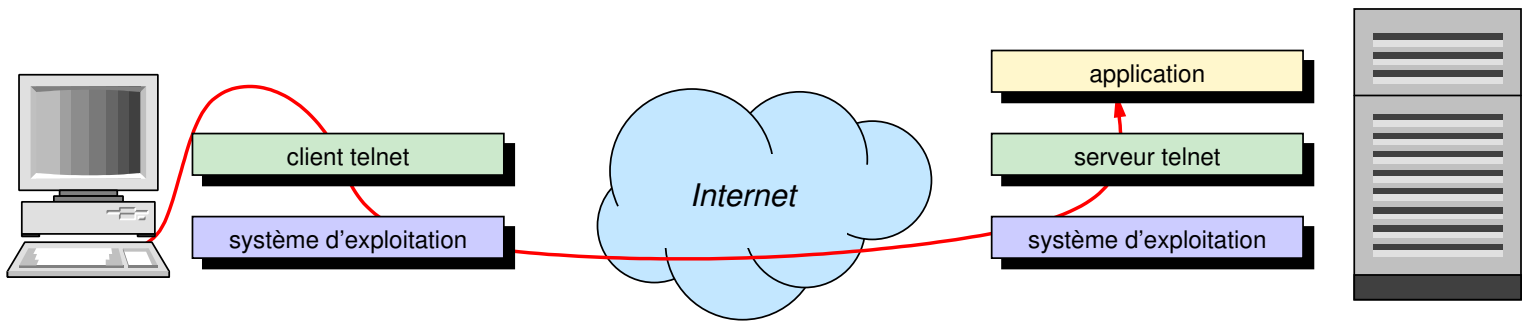
BOOTP et DHCP

RPC/XDR et NFS

X-Window

TELNET

Connexion à distance : un utilisateur accède à une application sur un serveur distant



TELNET

Avantages de TELNET :

- utilise des adresses IP aussi bien que des noms
- standard sur l'Internet (une des premières applications)
- indépendant du système (connexion depuis une machine Unix vers un mainframe IBM sous z/OS)
- 7 ou 8 bits

Inconvénients :

- frustre
- peu de sémantique liée au système
- non chiffré
- quasiment plus utilisé aujourd'hui
  $\Rightarrow$ protocole historique, mais intéressant
  $\Rightarrow$ notion de terminal virtuel, négociation d'options

TELNET

Problème : les programmes distants sont naïfs, ils croient que le terminal est directement connecté.

Solution : notion de *pseudo-terminal* (liée au système d'exploitation).

TELNET

Problème : interprétation des codes de contrôle différente suivant les systèmes.

Solution : définition du NVT (Network Virtual Terminal)

7	BEL	driiiing !
8	BS	←
9	HT	tabulation horizontale
10	LF	↓
11	VT	tabulation verticale
12	FF	page suivante
13	CR	retour en début de ligne
10+13	CR+LF	fin de ligne
32..126	ASCII	caractères imprimables
255	IAC	Interpret next As a Command

TELNET

Commandes : envoyées après le caractère IAC

...		
240	SE	fin des sous-options
241	NOP	aucune opération
242	DM	data mark
243	BRK	envoyer un *break*
244	IP	interrompre le programme
245	AO	abort output
246	AYT	are you there ?
...		
250	SB	début des sous-options
251	WILL	option acceptée
252	WON'T	option refusée
253	DO	accepter cette option
254	DON'T	ne pas accepter cette option
255		octet 255

Exemple :

1. $\rightarrow$ IAC AYT
2. $\leftarrow$ *un signal visuel* (ex : `machine.u-strasbg.fr here`)

TELNET

Cas particulier de commande : le signal *Synch*
$\Rightarrow$ moyen d'envoyer des commandes urgentes

Exemples :

- IP : interrupt process
- AO : abort output
- AYT : are you there ?

Méthode :

- l'une des deux parties envoie une ou des commandes importantes (IP, AO, AYT), puis envoie la commande DM en donnée *urgente* de TCP
- l'autre partie n'analyse que les commandes, jusqu'à la commande DM signalée par la donnée urgente.

TELNET

Options :

- échange d'informations
  (type de terminal, variables d'environnement, etc.)
- adaptation de comportements
  (mode caractère, ligne, édition locale, 8 bits, etc.)
- négociées entre les deux parties
- compatible même en cas d'option inconnue
  (auquel cas : refus de l'option)

Mécanisme :

- s'intègrent dans la connexion (commandes)
- négociation symétrique

TELNET

Négociation des options :

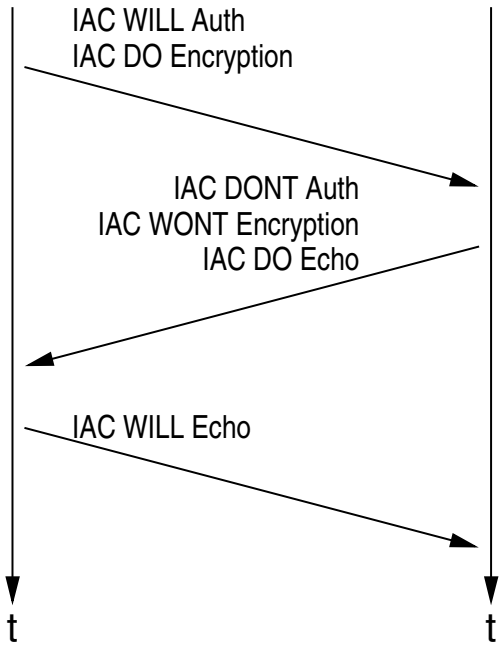
- je voudrais utiliser l'option *x* :
  1. → WILL *x*
  2. ← DO *x* / DON'T *x*
- je voudrais que vous utilisiez l'option *x* :
  1. → DO *x*
  2. ← WILL *x* / WON'T *x*

Cas particuliers :

- je ne veux pas utiliser l'option *x* :
  1. → WON'T *x*
  2. ← DON'T *x*
- je ne veux pas que vous utilisiez l'option *x* :
  1. → DON'T *x*
  2. ← WON'T *x*

TELNET

Exemple :



Problème : risque de désynchronisation

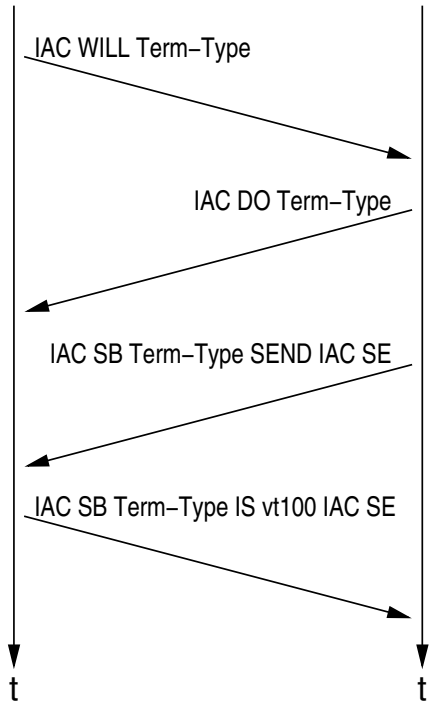
- risque de boucles d'acquittements
- règle = pas d'acquittement d'une requête pour une option déjà spécifiée

TELNET

Traitement des options non binaires : type de terminal, taille de la fenêtre, etc.

Méthode : extension via les sous-options (SB...SE)

0	IS
1	SEND
24	Term-Type



FTP

FTP : *File Transfer Protocol*
$\Rightarrow$ protocole historique de transfert de fichiers sur Internet

Avantages de FTP :

- standard sur l'Internet (une des premières applications)
- indépendant du système

Inconvénients :

- frustre
- non chiffré
- quasiment plus utilisé aujourd'hui
  $\Rightarrow$ protocole historique, mais intéressant
  $\Rightarrow$ protocole « en clair », introduction à SMTP

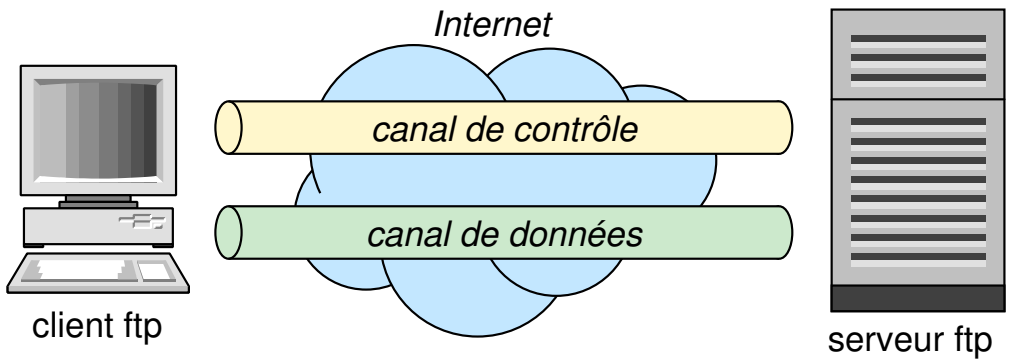
FTP

```
> ftp ftp.u-strasbg.fr                    # lancement de la commande ftp
Connected to anubis.u-strasbg.fr.
220 ProFTPD 1.3.5rc3 Server (Osiris IPv6) [2001:660:2402::6]

Name (ftp.u-strasbg.fr:pda): anonymous    # connexion anonyme
331 Anonymous login ok, send your complete email address as your password

Password: pda@unistra.fr                  # mot de passe = mon adresse électronique
230 Anonymous access granted, restrictions apply
Remote system type is UNIX.
Using binary mode to transfer files.

ftp> cd /pub/FreeBSD                      # changement de répertoire sur le serveur
250 CWD command successful

ftp> get README.TXT                       # récupération d'un fichier
local: README.TXT remote: README.TXT
200 EPRT command successful
150 Opening BINARY mode data connection for README.TXT (4259 bytes)
226 Transfer complete
4259 bytes received in 0.02 secs (174.6161 kB/s)

ftp> quit                                 # on s'en va
221 Goodbye.
```

FTP

FTP utilise deux connexions TCP :

connexion de *contrôle* :	envoi des commandes
connexion de *données* :	données à transférer



FTP

Connexion de contrôle :

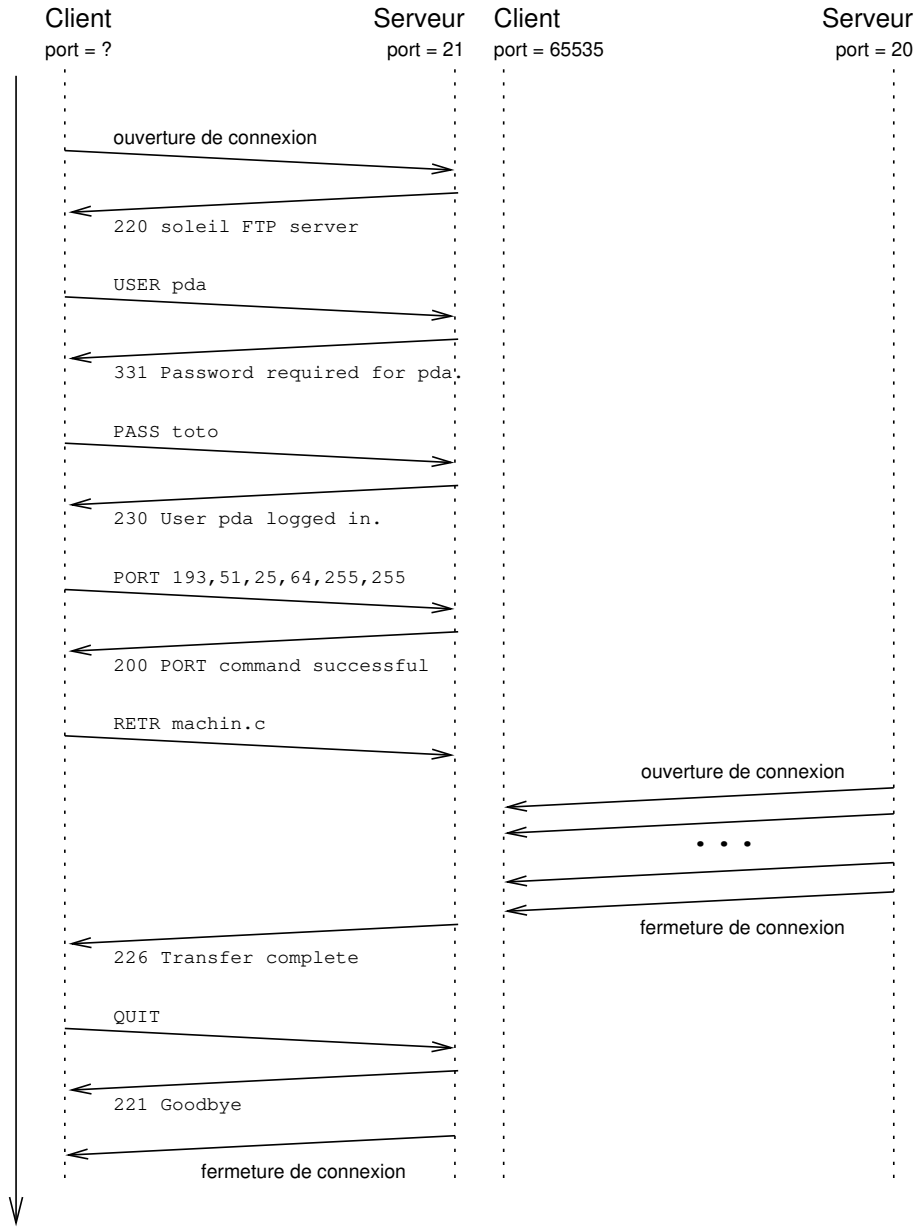
- le client envoie une commande
  sous forme d'un *verbe* suivi de paramètres
- le serveur effectue l'action
  et renvoie un code numérique (plus des messages)

Quelques « verbes » courants :

Verbe	Signification
USER	Authentification
PASS	Authentification
PORT	Numéro de la connexion de données
LIST	Lister les fichiers
ABOR	Arrêter le transfert en cours
PWD	Répertoire courant
CWD	Change le répertoire courant
DELE	Suppression de fichier
RETR	Lit le fichier
STOR	Écrit le fichier
TYPE	Type de transfert

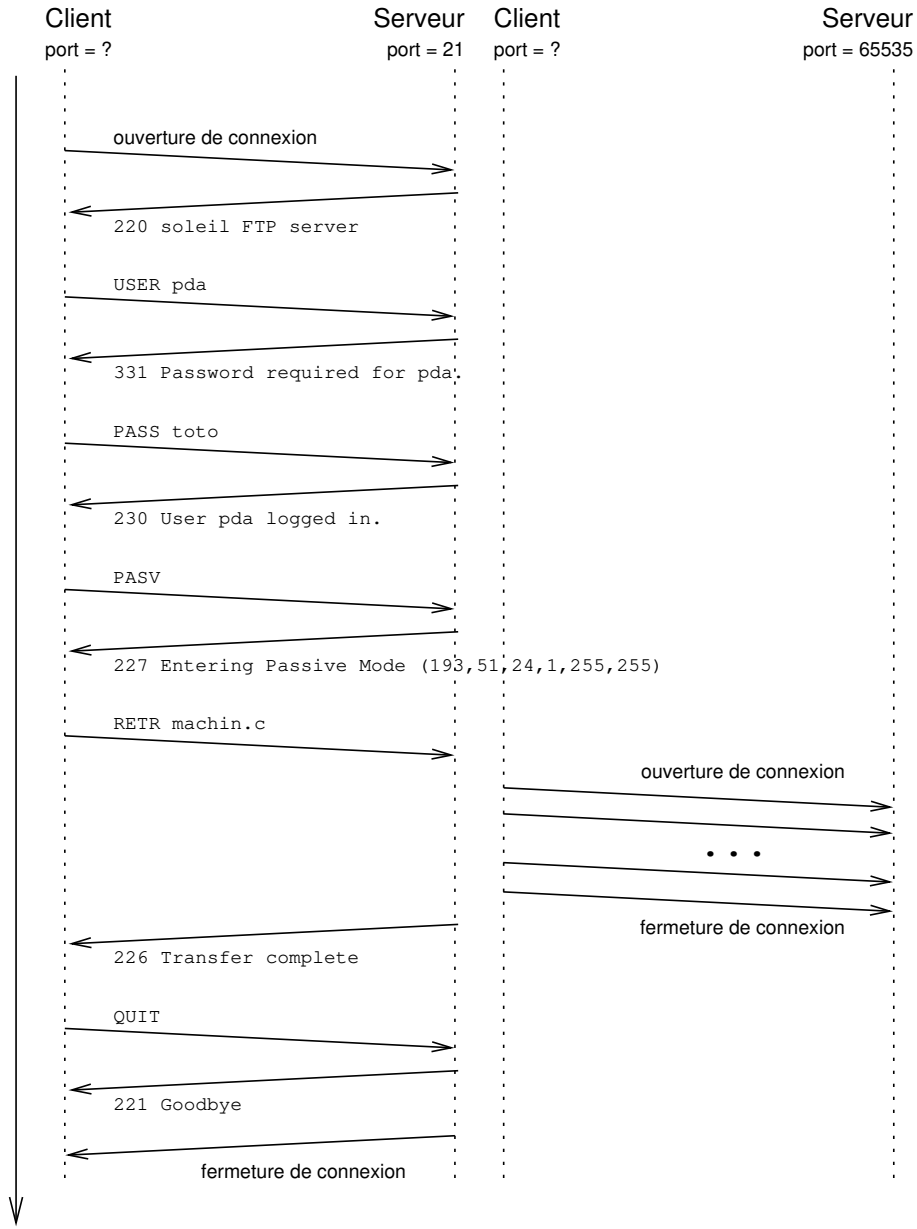
FTP

Échange de messages :



FTP

Connexion passive : facilite le travail des garde-barrières.



FTP

FTP anonyme : faciliter la distribution (de logiciels, de littérature, etc.)

Implémentation :

- serveur ftp reconnaît l'utilisateur spécial *anonymous*
- usage (politesse) : mettre son adresse électronique à la place du mot de passe
- accès à un sous-ensemble (contrôlé) de l'espace disque du serveur

Aujourd'hui supplanté par HTTP
$\Rightarrow$ beaucoup de sites ferment leur service FTP anonyme

TFTP

Problème : lourdeur de FTP

- utilise TCP
- beaucoup d'options
- authentification

$\Rightarrow$ trop lourd dans certaines situations (machines peu « intelligentes » : routeurs, terminaux X, cafetières, etc.)

Solution : protocole TFTP (*Trivial File Transfer Protocol*)

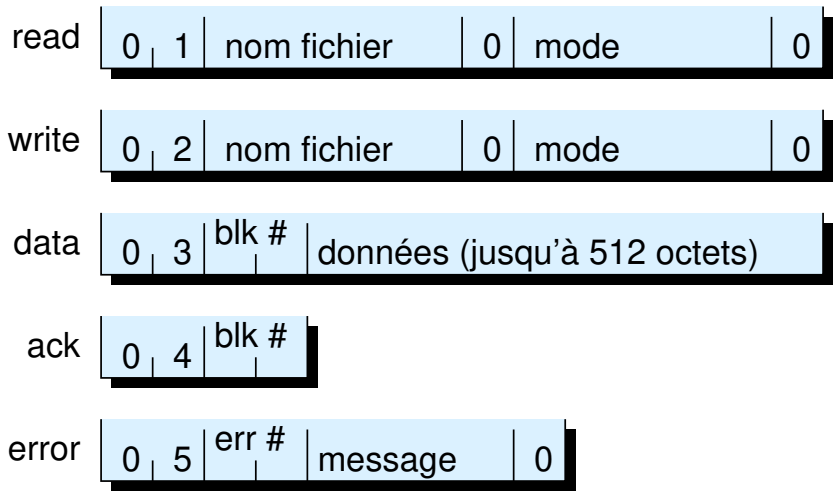
TFTP

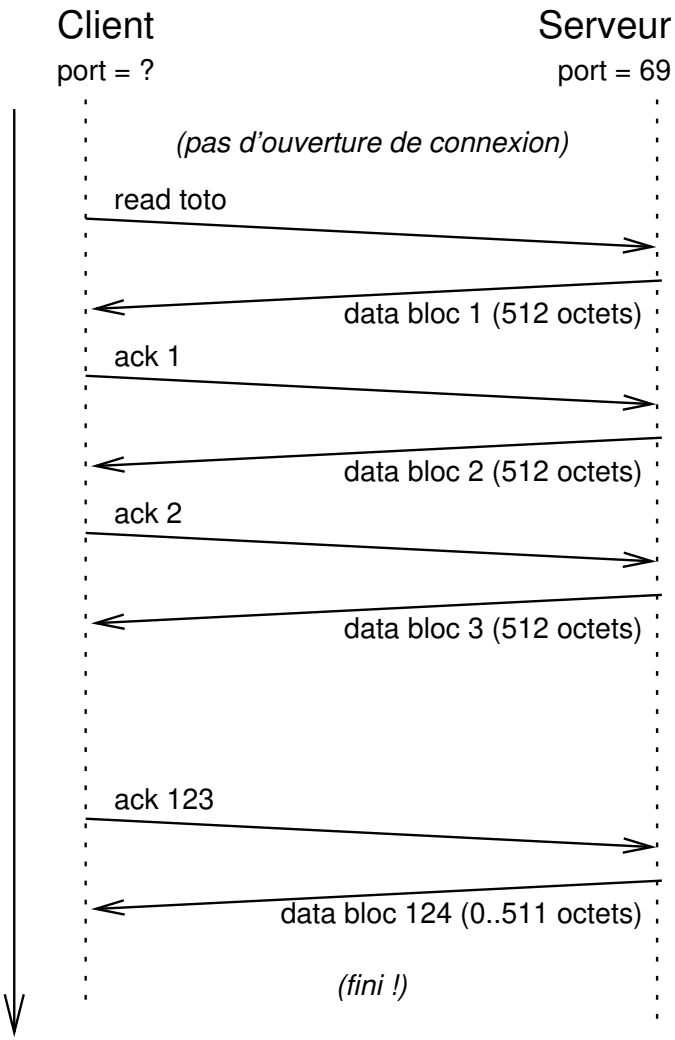
TFTP est très léger :

- utilise UDP
  $\Rightarrow$ intéressant sur réseaux locaux
  $\Rightarrow$ pas adapté aux réseaux longue distance
- pas d'authentification, pas de chiffrement
  $\Rightarrow$ pas utile pour démarrer un terminal X
- protocole simple, encodé en binaire
  $\Rightarrow$ très efficace

TFTP est toujours très utilisé dans certains contextes
$\Rightarrow$ démarrage de stations sans disque, sauvegarde de configurations d'équipements réseau, etc.

TFTP





Protocole très simple !

Rlogin

rlogin : connexion à distance (comparable à TELNET).

Avantages :

- protocole simple (beaucoup plus simple que TELNET)
- bien intégré dans l'environnement Unix
  - propage une partie de l'environnement (terminal, nom du terminal X, etc.)
  - authentification automatique
    - `/etc/hosts.equiv`
    - `$HOME/.rhosts`
  - *stdin*, *stdout*, *stderr*

Inconvénients :

- cf. TELNET
- disparu au profit de SSH

Rlogin

Protocole :

1. client envoie au serveur :
   - 1.1 octet nul (0)
   - 1.2 nom de login sur le client, suivi de 0
   - 1.3 nom de login sur le serveur, suivi de 0
   - 1.4 type de terminal, suivi de /
   - 1.5 vitesse de connexion, suivie de 0
2. serveur répond avec un octet nul
3. les données envoyées par le client sont transmises caractère par caractère (sans écho)
   Algorithme de Nagle pour *bufferiser*
4. les données envoyées par le serveur sont affichées sur l'écran

Rlogin

Commandes échangées :

- commandes du serveur vers le client :
  - via les données urgentes de TCP
  - 4 commandes définies :

2	flush output
16	arrêter de gérer Ctrl-S/Ctrl-Q
32	reprendre la gestion de Ctrl-S/Ctrl-Q
128	récupérer la taille de la fenêtre

- commandes du client vers le serveur :

  Une seule définie : envoi de la taille de la fenêtre.

  1. le client envoie 4 octets : 255+255+'s'+'s',
  2. puis 4 nombres sur 16 bits : taille de la fenêtre en caractères, puis en pixels

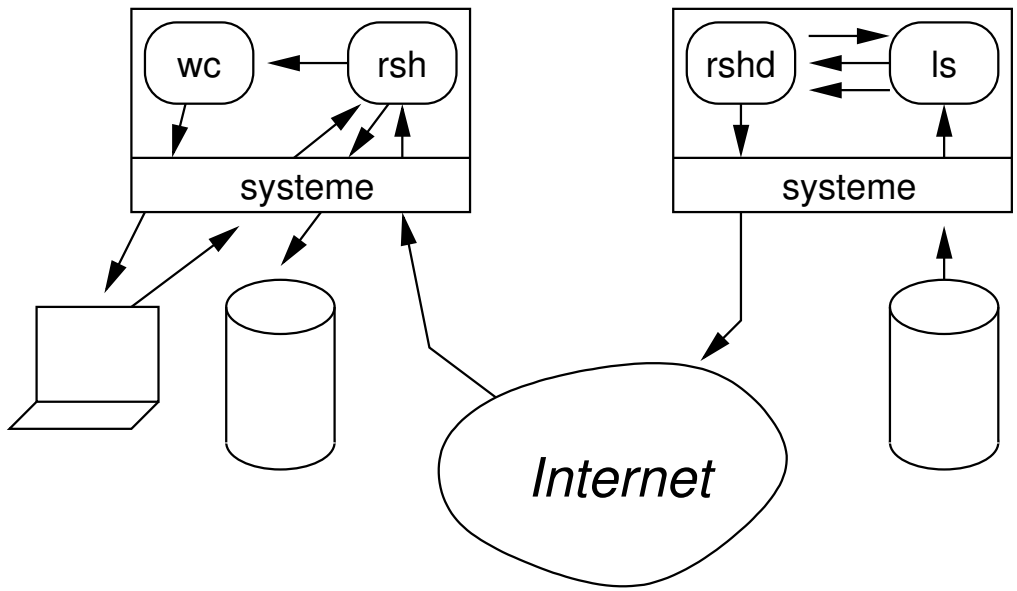
  Pas de caractère d'échappement pour 255 !

Rsh

`rsh` : exécution d'une commande à distance

Exemple :

```
$ rsh serveur.u-strasbg.fr ls | wc -l 2> toto
```



Rsh

Protocole :

1. le client envoie au serveur un numéro de port en ASCII, suivi par l'octet nul (0)
2. le serveur se connecte sur le port indiqué
   $\Rightarrow$ sortie d'erreur (`stderr`)
3. le client envoie au serveur :
   - 3.1 nom de login sur le client, suivi de 0
   - 3.2 nom de login sur le serveur, suivi de 0
   - 3.3 commande, suivie de 0
4. le serveur renvoie un octet nul
5. la connexion est ensuite utilisée pour la sortie standard (`stdout`) et l'entrée standard (`stdin`)

$\Rightarrow$ protocole à deux connexions, asymétrique

Rcp

`rcp` : copie de fichier

Exemple :

```
$ rcp vagabond.u-strasbg.fr:toto turing:titi
```

Possible si authentification réussie. Même sémantique que la commande `cp` classique.

SSH

Ssh (« *secure shell* ») est une application destinée à remplacer rlogin, rsh et rcp.

- 1995 : conception par Tatu Ylonen et formation de la société « SSH Communications Security »
- 1997 : groupe de travail IETF sur la version 2 du protocole
- 1999 : implémentation OpenSSH (basée sur ssh-1.x)
- 2000 : OpenSSH inclut le protocole v2
- 2006 : SSH-v2 = RFC 4251 à 4254

SSH

Fonctionnalités :

- facile d'emploi
- remplace complètement rlogin, rsh, telnet
- compatible avec rlogin et rsh
- communication chiffrée pour éviter l'espionnage
- encapsulation des flux X11 (tunnel)
- encapsulation de n'importe quel flux TCP (tunnel)

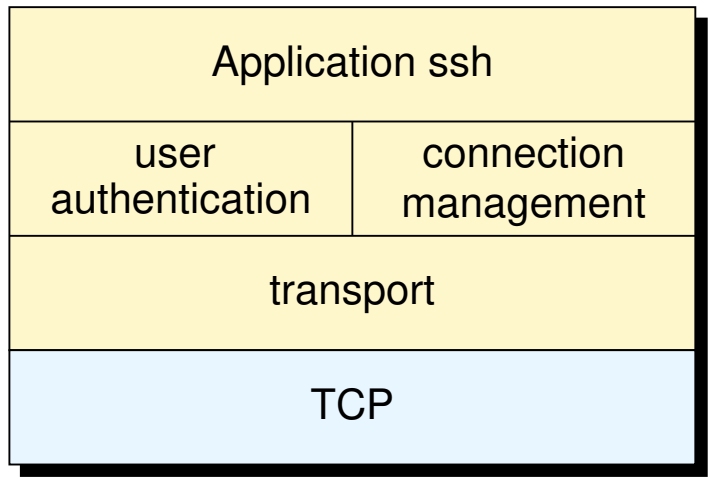
Chaque serveur (ou client) ssh dispose d'une clef de signature (clef privée et clef publique K_s) qui l'authentifie. Exemple pour RSA :

- `/etc/ssh/ssh_host_rsa_key`
- `/etc/ssh/ssh_host_rsa_key.pub`

SSH

Décomposition du protocole SSH v2 en couches :

- « transport » : authentification du serveur, confidentialité, intégrité, échange des clefs (toutes les 1 h ou 1 Go)
- « user authentication » : authentification de l'utilisateur auprès du serveur
- « connection management » : multiplexe le tunnel chiffré en plusieurs canaux logiques (ex : sessions interactives + port fowarding)

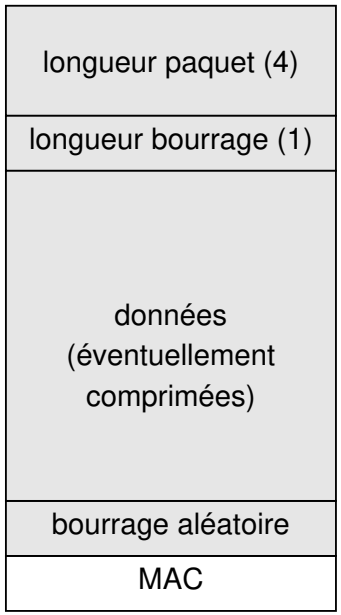


SSH – Protocole de transport

Protocole de transport de SSH v2 :

- longueur du paquet : longueur totale du paquet (hors longueur et MAC)
- longueur du bourrage : voir ci-dessous
- données : les données utiles du paquet, pouvant être éventuellement comprimées
- bourrage aléatoire : bourrage éventuel avec des données aléatoires pour arriver à une longueur multiple de 8 octets. Le bourrage doit faire au minimum 4 octets.
- MAC : code d'authentification du message non chiffré

Toutes les données (sauf le MAC) sont chiffrées.

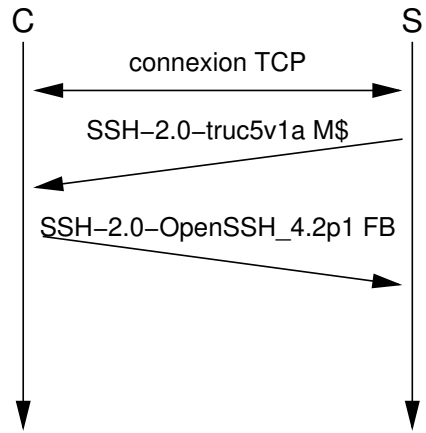


SSH – Protocole de transport

Le protocole SSH v2 (étape 1) :

- ouverture de la connexion TCP
- échange des bannières
  Forme : SSH-2.0-*logiciel commentaire*
  Les bannières permettent l'adaptation de la version.
  Le texte de la bannière (« *logiciel* ») est utilisé par la suite.

Note : l'échange des bannières est le seul échange ne passant pas par le format général du protocole de transport.

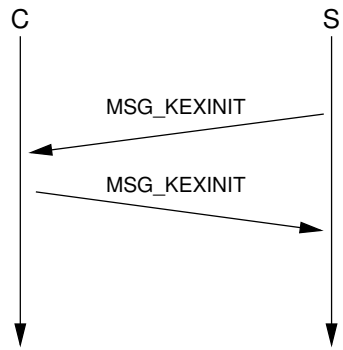


SSH – Protocole de transport

Le protocole SSH v2 (étape 2) :

Chaque partie annonce :

- un nombre aléatoire (*cookie*) R sur 128 bits
- les algorithmes pour l'échange des clefs
- les algorithmes pour la clef de serveur
- les algorithmes symétriques C→S
- les algorithmes symétriques S→C
- les algorithmes de MAC C→S
- les algorithmes de MAC S→C
- les algorithmes de compression C→S
- les algorithmes de compression S→C



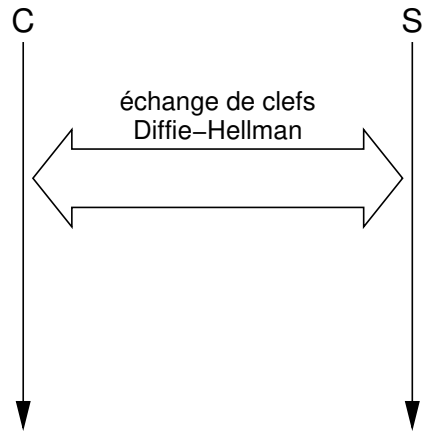
Algorithmes cités par ordre de préférence (premier = préféré)
Chaque partie détermine indépendamment l'intersection des algorithmes (préféré d'abord, priorité au client ensuite).
⇒ algorithmes obligatoires : éviter une intersection vide !

SSH – Protocole de transport

Le protocole SSH v2 (étape 3) :

L'échange de clef Diffie-Hellman conduit à :

- secret partagé k
  $k = Y^x \bmod p = X^y \bmod p$
- valeur de hachage h
  $h = \text{hash}(V_c + V_s + I_c + I_s + K_s + X + Y + k)$
- dans l'échange, le serveur envoie en plus (par rapport à DH classique) :
  - sa clef publique K_s
  - la signature $s = \text{sign}(h)$ avec sa clef privée



Note :

- V_c et V_s : versions citées dans les bannières initiales
- I_c et I_s : contenus des messages de l'étape 2

SSH – Protocole de transport

Le protocole SSH v2 (étape 3 suite) :

Le client reçoit K_s et $s \Rightarrow$ vérifier l'identité du serveur :

- grâce à la clef publique du serveur K_s, qu'il compare :
  - à la version stockée dans le fichier `known_hosts`
  - à la version stockée dans un annuaire de clefs (DNS)
  - manuellement (grâce à l'empreinte en clair de cette clef affichée à l'écran)
- par la signature s : c'est bien le possesseur de la clef K_s qui a signé

SSH – Protocole de transport

Le protocole SSH v2 (étape 3 fin) :
Le chiffrement et la vérification d'intégrité sont effectués grâce à des algorithmes potentiellement différents dans chaque sens.
Les deux parties se sont accordées sur un secret k et h :

Client	Serveur
$iv = \text{hash}(k + h + \text{"B"} + \text{sid})$	$iv = \text{hash}(k + h + \text{"A"} + \text{sid})$
$k_c = \text{hash}(k + h + \text{"D"} + \text{sid})$	$k_c = \text{hash}(k + h + \text{"C"} + \text{sid})$
$k_i = \text{hash}(k + h + \text{"F"} + \text{sid})$	$k_i = \text{hash}(k + h + \text{"E"} + \text{sid})$

Notes :

- iv : vecteur d'initialisation du chiffrement symétrique en mode CBC
- k_c : clef de chiffrement symétrique (dans chaque sens)
- k_i : clef d'intégrité pour le calcul HMAC (dans chaque sens)
- sid : *session id*, valeur initiale du hasard R

SSH – Protocole de transport

Le protocole SSH v2 :
Les étapes 2 et 3 peuvent être renouvelées à l'initiative du client ou du serveur :

- tous les Go échangés ou
- toutes les heures

$\Rightarrow$ valeurs de configuration paramétrables.
La valeur aléatoire R est recalculée, mais le *session id* reste inchangée (valeur initiale de R).

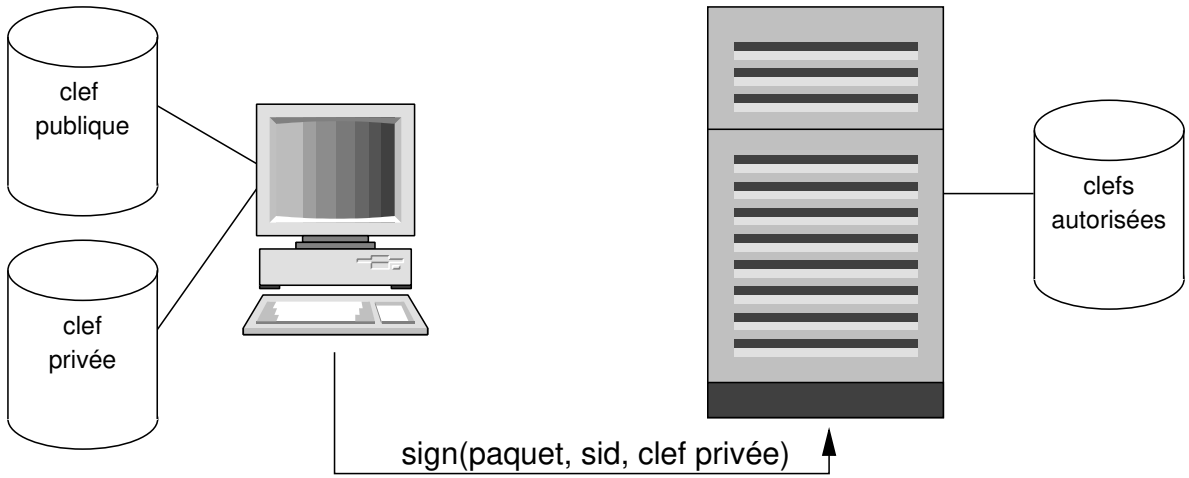
SSH – Protocole d'authentification

Le protocole d'authentification supporte trois mécanismes :

- signature à clef publique (« *publickey* »)
- mot de passe (« *password* »)
- basée sur le nom du client (« *host* »)

SSH – Protocole d'authentification

Authentification par signature à clef publique :



- L'utilisateur génère clef privée + clef publique sur le client :
  ⇒ commande `ssh-keygen`
  ⇒ fichiers `~/.ssh/id_rsa` et `id_rsa.pub` (ou `id_dsa`)
- L'utilisateur installe sa clef publique sur le serveur (par `ssh` par exemple) :
  ⇒ fichier `~/.ssh/authorized_keys2`

SSH – Protocole d'authentification

Authentification par signature à clef publique :
La requête d'authentification comprend :

- le nom d'utilisateur
- « publickey »
- l'algorithme de signature à clef publique (ex : `ssh-dss`)
- la clef publique
- la signature (sur tous les autres champs de la requête + *session id*)

La signature :

- certifie que le possesseur de la clef publique se connecte
- n'est pas rejouable (intègre le *session id*)

L'accès à la clef privée sur le client peut être protégé par un mot de passe local.
⇒ `ssh-agent` : évite d'avoir à re-saisir le mot de passe

SSH – Protocole d'authentification

Authentification par mot de passe :
Le client envoie :

- le nom d'utilisateur
- le mot de passe « en clair »

$\Rightarrow$ ces informations sont chiffrées par le protocole de transport

Méthode :

- la plus simple $\Rightarrow$ la plus utilisée
- support obligatoire dans toutes les implémentations
- peu conviviale

SSH – Protocole d'authentification

Authentification basée sur le nom du client : analogue aux `.rhosts` avec `rlogin`

Le client envoie :

- le nom d'utilisateur
- l'algorithme de signature à clef publique utilisé par le client
- la clef publique (K_s) du client
- le nom complet du client
- le nom d'utilisateur sur le client
- la signature (avec la clef privée du client, sur tous les autres champs de la requête + *session id*)

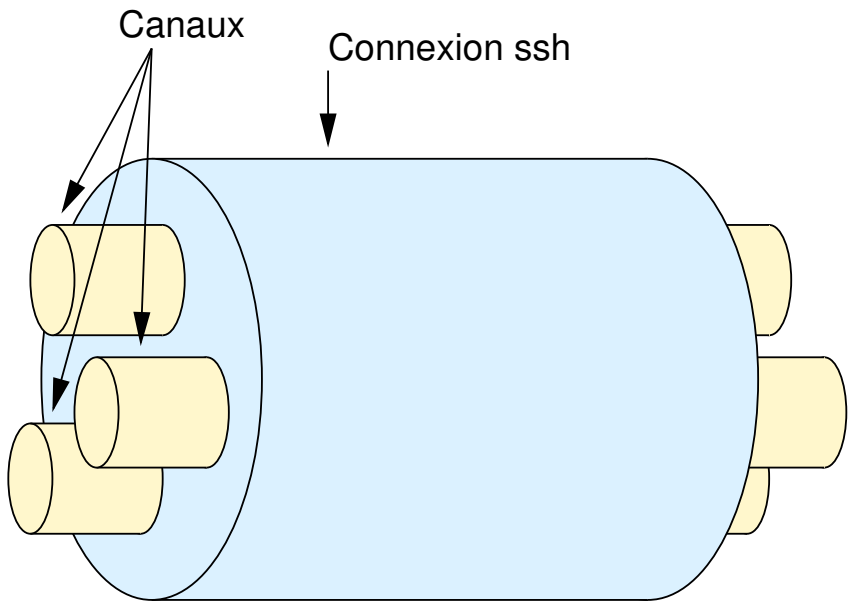
$\Rightarrow$ principe identique à `rlogin`, mais vérification plus rigoureuse du client.

Méthode :

- compatible avec `rlogin`
- peu recommandée dans des environnements sécurisés
- vulnérable à une divulgation de la clef privée du client

SSH – Protocole de connexion

Protocole de connexion : permet le multiplexage de plusieurs canaux indépendants dans une seule connexion ssh (avec le protocole de transport).



Exemples :

- encapsulation des flux X11
- encapsulation de sessions basées sur des redirections de ports

SSH – Protocole de connexion

Opérations du protocole :

- ouverture et fermeture d'un canal (avec un type)
- transfert de données
- fin de fichier sur un canal

Types de canaux :

- session interactive
- redirection X11
- changement de taille de fenêtre
- passage de variables d'environnements
- connexion TCP/IP redirigée
- etc.

SSH – Protocole v1

Version précédente du protocole : 1.0

- contrôle d'intégrité basé sur des CRC sur 32 bits
  $\Rightarrow$ insuffisant, facilement contournable
- vulnérable aux attaques « *man in the middle* »
  $\Rightarrow$ seulement lorsque la clef du serveur est inconnue
- moins documentée, moins standardisée

$\Rightarrow$ la version 2.0 est la version recommandée

SSH – Bilan

Ssh est l'exemple d'application de sécurité idéale :

- documentée
- simple d'utilisation
- bien conçue
- implémentations libres

Plan

De Telnet/FTP à SSH

Courrier électronique

BOOTP et DHCP

RPC/XDR et NFS

X-Window

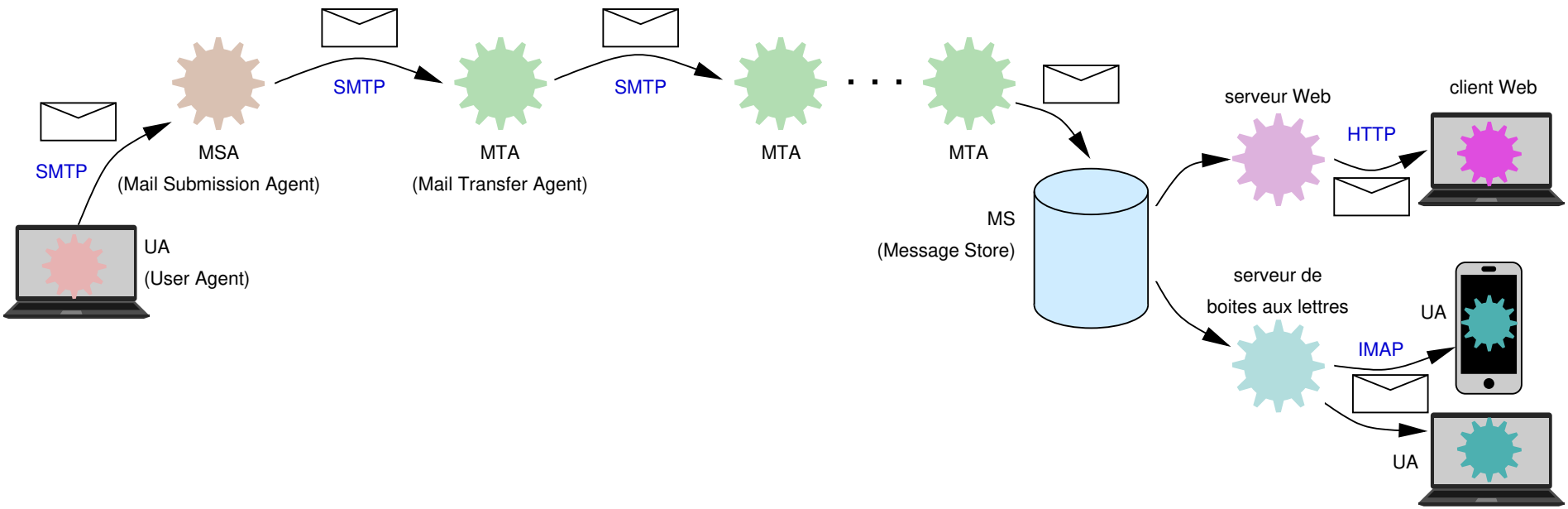
Courrier électronique

Dans les années 1980, deux protocoles principaux en concurrence :

- X.400
  - norme internationale de l'ISO
    - accès payant à la spécification
  - plusieurs version (X.400 1984, X.400 1988, etc.)
  - complexe
  - norme tombée en désuétude
- SMTP
  - **Simple** Mail Transfer Protocol
  - protocole standardisé par l'IETF
    - « standard » $\Rightarrow$ 2 implémentations interopérables
    - accès facile à la spécification
  - 115 pages pour la spécification originale (RFC 821 + 822, 1982)
  - omniprésente
  - évolutions majeures depuis

Courrier électronique

Architecture générale d'un système de messagerie :



- UA : User Agent (compose les messages, affiche les messages)
- MSA : Mail Submission Agent (accepte un message dans le système)
- MTA : Mail Transfer Agent (transfère les messages)
- MS : Message Store (boîte aux lettres contenant les messages)

Courrier électronique

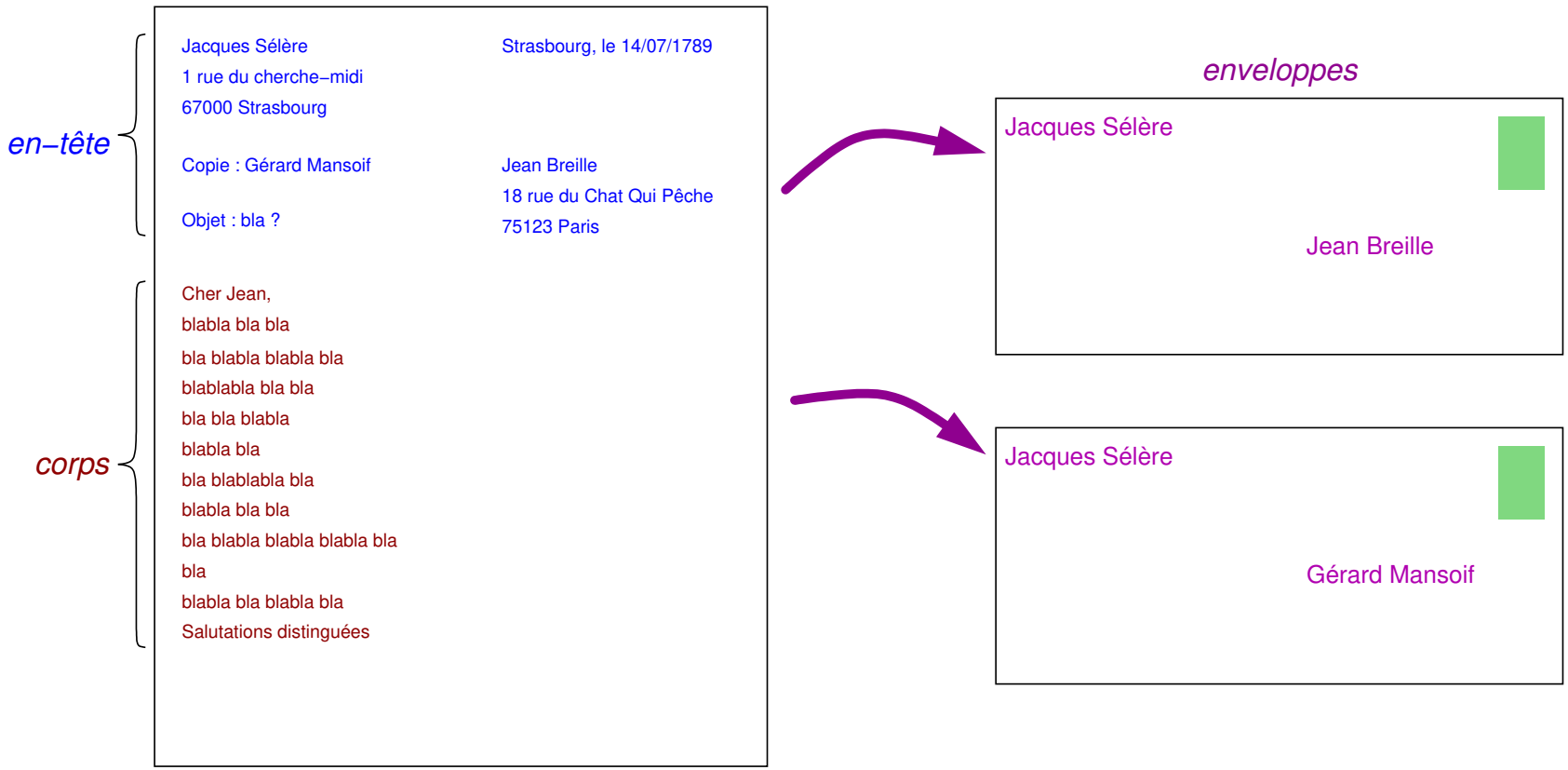
Principaux protocoles mis en œuvre :

SMTP	envoi initial du message par l'expéditeur
SMTP	transfert du message de proche en proche
IMAP	mise à disposition des messages pour le destinataire

- SMTP : Simple Mail Transfer Protocol
- IMAP : Internet Message Access Protocol
  - avant : POP (Post Office Protocol)

Constituants d'un message

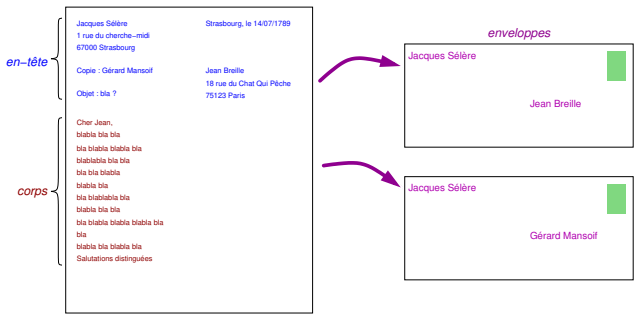
Analogie postale :



- en-tête : l'émetteur, le destinataire, la date, etc.
- corps : le contenu du message
- enveloppe : le message pendant son transfert

Constituants d'un message

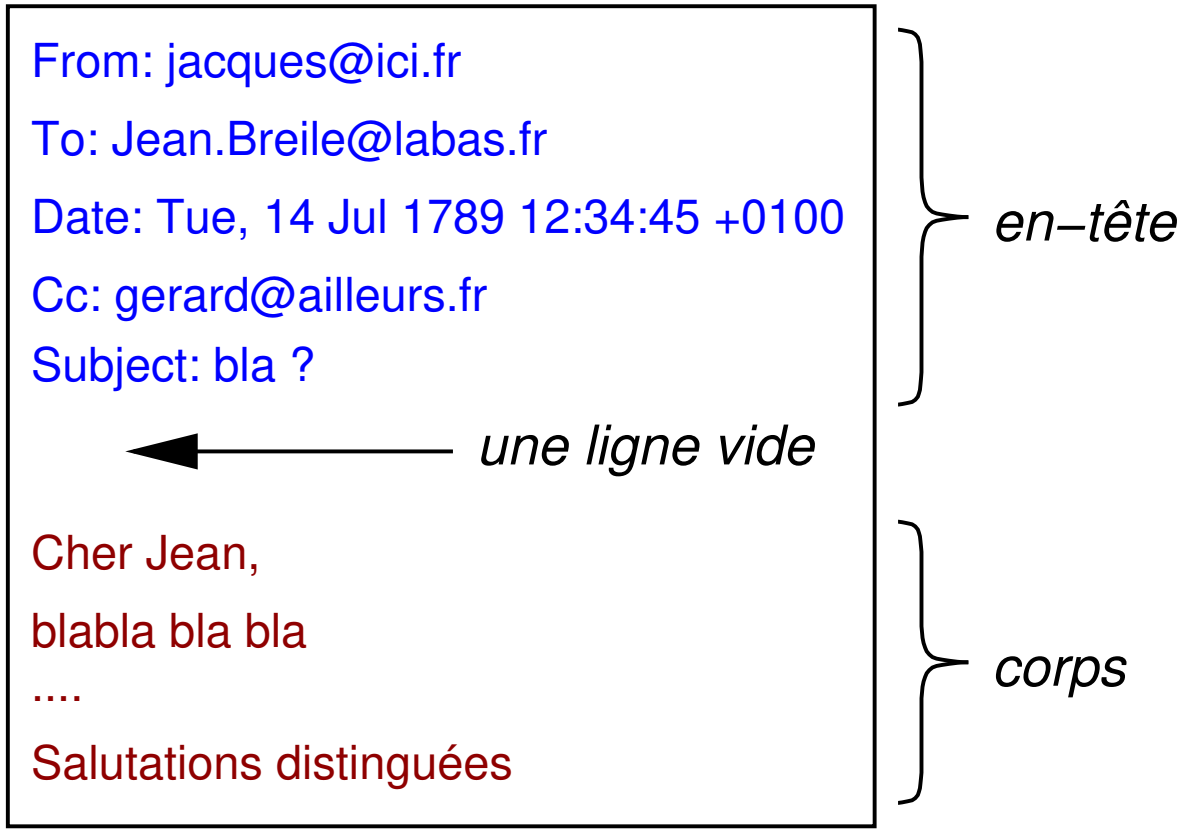
Analogie postale :



- le message est placé dans une enveloppe
  - ou plusieurs enveloppes s'il y a plusieurs destinataires
  - le facteur ne regarde pas le message
    - ni l'en-tête, ni le corps
  - rappel : le secret des correspondances est censé être inviolable
- l'enveloppe est utilisée par le facteur pour router le message
  - ou le re-router en cas de redirection ou d'erreur de destinataire
- l'enveloppe est détruite lorsque le message arrive à destination

Constituants d'un message

Avec SMTP :



⇒ la structuration des messages est simple :

- en-tête constituée de couples <clef, valeur>
  - exemple : `From: jacques@ici.fr`
- peu de contraintes sur le corps (= suite de lignes)

Constituants d'un message

Quelques champs d'en-tête :

`From`	Adresse d'expéditeur
`Sender`	Adresse de la personne qui a envoyé le message
`To`	Adresses des destinataires
`Cc`	Adresses des destinataires en copie (*carbon copy*)
`Bcc`	Adresses des destinataires en copie cachée (*blind cc*)
`Date`	Date d'envoi
`Message-Id`	Identifiant unique du message
`Subject`	Sujet du message
`In-Reply-To`	Identifiant du message original, pour les réponses
`Received`	Marquage opéré lors du transfert du message
`X-...`	Champs « expérimentaux »

Même si le format semble souple, on ne peut pas « inventer » un nouveau champ, ni utiliser n'importe quel format (cf. champ « date »)

Avec un client de messagerie « normal », on peut afficher tous les champs de l'en-tête d'un message

Constituants d'un message

Une adresse est constituée de :

partie-locale @ *nom-de-domaine*

Plusieurs formats possibles pour les adresses :

jean@labas.fr	classique...
jean@labas.fr (Jean Breille)	on peut ajouter des commentaires
jean (Jean Breille)@labas.fr	et même n'importe où
"jean breille"@labas.fr	caractère spécial dans l'adresse
<jean@labas.fr>	adresse facilement analysable par un programme
Jean <jean@labas.fr> Breille	tout ce qui n'est pas entre < > est un commentaire
<jean@labas.fr> "Jean Breille"	caractères spéciaux dans les commentaires

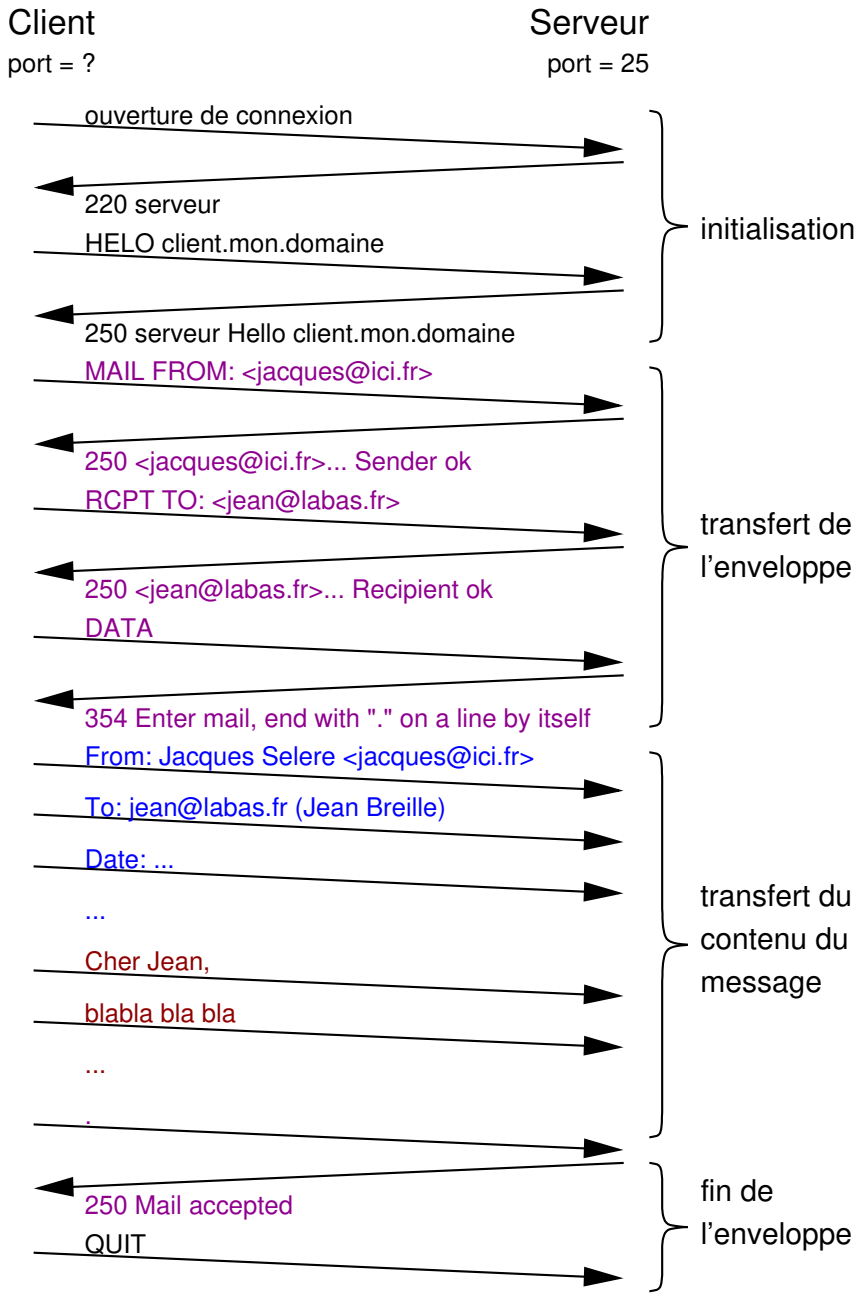
SMTP

Protocole SMTP (*Simple Mail Transfer Protocol*)

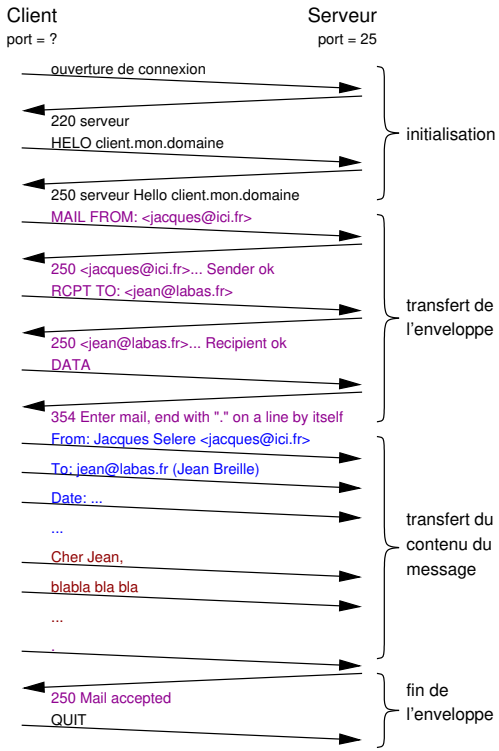
- simple
- commandes = texte
- basé sur TCP

Le protocole permet à l'émetteur d'envoyer l'enveloppe, puis le message

SMTP


Client
port = ?
Serveur
port = 25
ouverture de connexion
220 serveur
HELO client.mon.domaine
250 serveur Hello client.mon.domaine
initialisation
MAIL FROM: <jacques@ici.fr>
250 <jacques@ici.fr>... Sender ok
RCPT TO: <jean@labas.fr>
250 <jean@labas.fr>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
transfert de l'enveloppe
From: Jacques Selere <jacques@ici.fr>
To: jean@labas.fr (Jean Breille)
Date: ...
...
Cher Jean,
blabla bla bla
...
.
transfert du contenu du message
250 Mail accepted
QUIT
fin de l'enveloppe

SMTP



- protocole textuel
- commande : « verbe » suivi de paramètres
  - exemple : verbe = `MAIL FROM:`, paramètre = `<jacques@ici.fr>`
- le serveur répond par des codes
  - exploités par les programmes
  - le serveur peut ajouter des commentaires pour les humains (debug)
- algorithme du « point seul sur sa ligne » pour terminer le message
  - si une ligne du message commence par un point, il sera doublé

SMTP

Chaque commande traitée par le serveur donne lieu à un code en retour :

- 3 chiffres
- premier chiffre indique l'état de la demande :

2	demande traitée sans erreur
3	demande en cours d'exécution
4	erreur temporaire détectée, il faudra réessayer
5	erreur permanente détectée, pas la peine de réessayer

- les codes indiquent une transition dans l'automate d'états
- ces codes ont inspiré HTTP

SMTP

Protocole SMTP également utilisé pour la soumission d'un message par le client de messagerie (UA) :

- l'UA se comporte comme un client SMTP
- ajout de verbes pour identifier et authentifier l'utilisateur
- utilisation d'un port TCP spécifique (587) pour :
  - différencier les fonctionnalités
  - contourner les filtrages
    port SMTP (25) souvent filtré pour prévenir le spam

SMTP

SMTP tel que défini en 1982 comporte des limitations :

- caractères sur 7 bits
  - transport de messages avec des caractères ASCII uniquement
  - pas de jeux de caractères nationaux
    - pas d'Arabe, de Katakana, de Coréen, de Cyrillique, etc.
    - ... et pas d'accents en français !
  - pas de transport de fichiers binaires
  - pire encore : SMTP impose de tronquer le 8e bit de chaque octet
    - une implémentation n'a pas le droit d'être libérale et d'accepter quand même des caractères accentués
    - « â » (en alphabet ISO 8859-1) = 0xe2 = 1110 0010
    - après troncature : 0110 0010 = 0x62 = « b » (en ASCII)
- le corps des messages est composé de lignes
  - pas de structuration des messages
  - pas de transport d'images, de vidéos, etc.
- SMTP est figé, non extensible
  - pas moyen de le faire évoluer

de SMTP à ESMTP + MIME

À partir du milieu des années 1990, deux voies complémentaires d'évolution :

- ESMTP (*Extended SMTP*)
  - nouveau protocole rétro-compatible avec SMTP
  - transfert des 8 bits et des données binaires
  - autres extensions (accusés de réception, etc.)
  - extensible dans le futur
  - le déploiement va prendre du temps
- MIME (*Multi-Purpose Mail Extension*)
  - structuration et typage des messages
  - encodage des messages
    - pour les transférer via un canal SMTP à 7 bits...
    - ... ou via un canal ESMTP capable de transférer les 8 bits

Le déploiement a pris du temps (une décennie environ)

$\Rightarrow$ c'est maintenant accompli !

ESMTP

Principe : nouveau verbe EHLO

- *Extended HELO* : référence au verbe *HELO* de SMTP
- le client tente EHLO
- serveur SMTP classique : renvoie une erreur

  $\Rightarrow$ le client réessaye avec HELO et continue en SMTP

  220 serveur
  EHLO client.mon.domaine
  550 Command not recognized
  HELO client.mon.domaine *(ah pardon ! je m'étais trompé)*
  250 serveur Hello client.mon.domaine
- serveur ESMTP: il répond avec les extensions supportées

  $\Rightarrow$ l'échange continue avec ESMTP

  220 serveur
  EHLO client.mon.domaine
  250-serveur Hello client.mon.domaine
  250-8BITMIME
  250-SIZE
  ...
  250 HELP
  - option 8BITMIME : supporte le transport des caractères sur 8 bits

MIME

Ajout de trois champs dans l'en-tête des messages :

`MIME-Version`	valeur = 1.0 (inchangée depuis l'origine)
`Content-Transfer-Encoding`	encodage du message
`Content-Type`	typage du message

MIME – Encodage

Valeurs possibles du champ `Content-Transfer-Encoding` :

- `7bits` : le message ne contient que des caractères dont les codes sont compris entre 0 et 127
  - le message peut transiter sur un canal à 7 bits
- `8bits` : le message contient des caractères dont les codes sont compris entre 0 et 255
  - le message nécessite un canal à 8 bits
- `quoted-printable` : le message contient des caractères encodés suivant la méthode :
  - si le caractère a un code ≤ 127, il reste tel quel
  - si le caractère a un code ≥ 128, il est encodé en « $=c_1c_2$ » où c_1c_2 est la valeur en hexadécimal (exemple : « â » devient « =E2 »)

  D'où :
  - le message conserve une certaine lisibilité

    « cet été, j'étais à la plage » devient « cet =E9t=E9, j'=E9tais =E0 la plage »
  - le message peut transiter sur un canal à 7 bits

MIME – Encodage

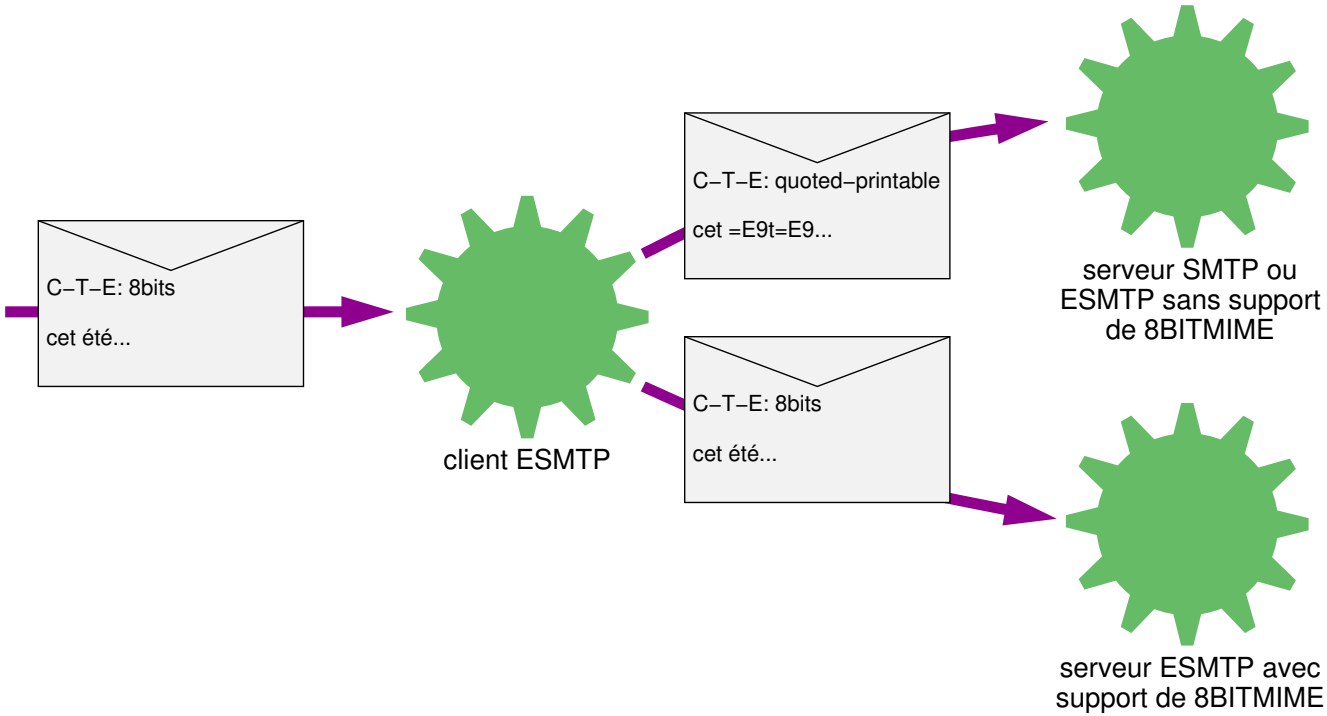
Valeurs possibles du champ `Content-Transfer-Encoding` :

- `base64` : le message contient tous les octets encodés suivant la méthode suivante
  - les octets sont regroupés par 3 $\Rightarrow$ 24 bits
  - le paquet de 24 bits est regroupé en 4 paquets de 6 bits
  - chacun des 6 bits contient une valeur entre 0 et $2^6 - 1 = 63$
  - cette valeur est un index dans une table de 64 caractères : [A, B..., Z, a, b..., z, 0, 1... 9, +, /]

  D'où :
  - le message n'est plus lisible par un humain
  - taille du message encodé = $\frac{4}{3} \times$ taille du message original
  - le message peut transiter sur un canal à 7 bits
- `binary` : le message reste tel quel
  - nécessite un canal adapté au transport d'informations binaires

MIME – Encodage

Grâce à ESMTP, un client ESMTP peut transcoder le message en fonction du serveur :



Si le premier client est SMTP, c'est à l'UA de faire le premier encodage.

MIME – Typage

Le champ d'en-tête `Content-Type` indique le type du message

Structure :

type / sous-type ; paramètres

Exemples :

- `text/plain ; charset=iso-8859-15 ; format=flowed`
- `text/html ; charset=utf-8`
- `image/jpeg ; name="vacances.jpg"`
- `image/png`
- `audio/basic`
- `video/mpeg`
- `application/pdf ; name="facture.pdf"`
- `application/vnd.openxml ; name="pres.pptx"`
- `message/delivery-status`

Le client de messagerie (UA) sait maintenant ce qu'il faut faire avec le message

MIME – Typage

Exemple complet :

```
From: Jacques.Selere@ici.fr
To: jean@labas.fr (Jean Breille)
Date: ...
Subject: superbe photo de vacances
MIME-Version: 1.0
Content-Type: image/jpeg ; name="vacances.jpg"
Content-Transfer-Encoding: base64

QqLzW+P/UReZE+tmfGWMI7/1finQkrvNKfAnv1E+xSpV7DNa9VITciDIUrey0L2g
In4T2HypZuxYwtzxwTz/0JT58SWAs4hRbW0C2aU/VtoEXvj+NcdqXv6sZTcbdKSD
DSok4bxNg9ydiguns00L96U+qFz5NX1/65l9+FgIKuP8SlIWh87Ivl/dDtTKzMvl
5O1bBMhXJV5CEwe31WOPhJwd0OKOaA4cu62dE3l8v/tkL2uQWN6hXzKhnLFNGkTr
...
```

MIME – Structuration

Le plus souvent, un message est constitué de plusieurs éléments :

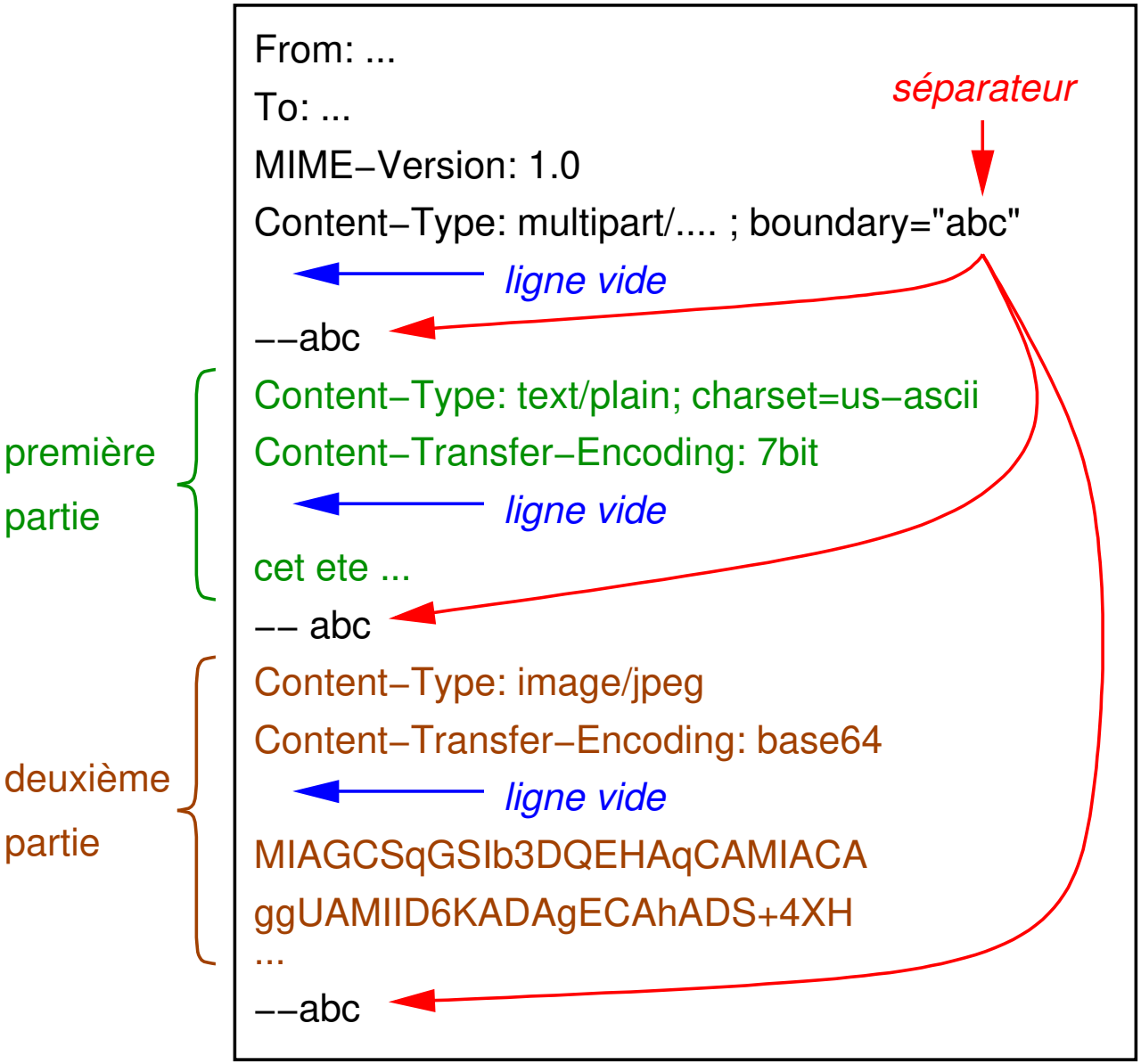
- un texte et plusieurs photos
- un texte avec un message transféré contenant lui-même plusieurs photos
- plusieurs versions d'un même document

$\Rightarrow$ type « **multipart** »

- permet de décomposer le message en sous-parties
  - séparées par un délimiteur
- chaque partie est un mini-message avec :
  - l'en-tête comprenant le type et l'encodage
  - le corps

MIME – Structuration

Exemple :



MIME – Structuration

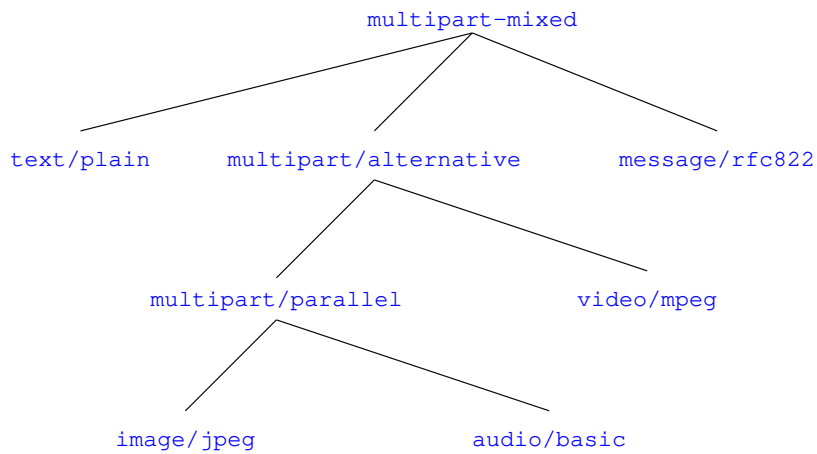
Différents sous-types :

- `multipart/alternative`
  - même contenu, plusieurs versions sous différents formats (texte brut, enrichi, etc.) dans l'ordre du plus simple vers le meilleur
  - l'UA du destinataire choisit le « meilleur » en fonction de ses capacités
- `multipart/mixed`
  - les différentes parties sont présentées dans l'ordre du message
  - exemple : une série de photos (de vacances...)
- `multipart/parallel`
  - les différentes parties sont présentées simultanément
  - exemple : une image et un son en même temps
- `multipart/related`
  - les différentes parties sont un seul et même document
  - exemple : un texte et la signature électronique, ou une page HTML et les images associées

MIME – Structuration

Un contenu « multipart » peut lui-même contenir des sous-parties de type « multipart »

- structuration forte du message
- exemple : un message comprenant plusieurs parties présentées dans l'ordre
  1. un texte
  2. plusieurs versions d'un même contenu
     - tout d'abord, une image et un son présentés simultanément
     - ou, mieux, une vidéo
  3. la copie d'un mail reçu dans la boîte aux lettres



Plan

De Telnet/FTP à SSH

Courrier électronique

BOOTP et DHCP

RPC/XDR et NFS

X-Window

BOOTP / DHCP

Problème : lorsqu'un terminal X ou une station sans disque démarre, il ou elle a besoin :

- de son adresse IP
- de la localisation du fichier (serveur, méthode et nom de fichier) contenant son système d'exploitation
- et d'autres informations (routeur, serveur DNS, etc.)

Même problème lorsqu'on connecte un ordinateur au réseau sans-fil de l'université ou derrière sa box.

BOOTP / DHCP

Solution partielle au problème : RARP

Inconvénients de RARP :

- n'utilise pas IP
  $\Rightarrow$ solution orthogonale au reste des protocoles
- nécessite d'accéder aux adresses Ethernet
  $\Rightarrow$ généralement inaccessibles aux programmes
- retourne peu d'information (i.e. adresse IP)
  $\Rightarrow$ nécessite autre protocole (ex. : bootparam/Sun)
- nécessite que le serveur soit sur le même réseau
  $\Rightarrow$ pas adapté aux grands réseaux

BOOTP / DHCP

Solution complète : protocoles BOOTP et DHCP

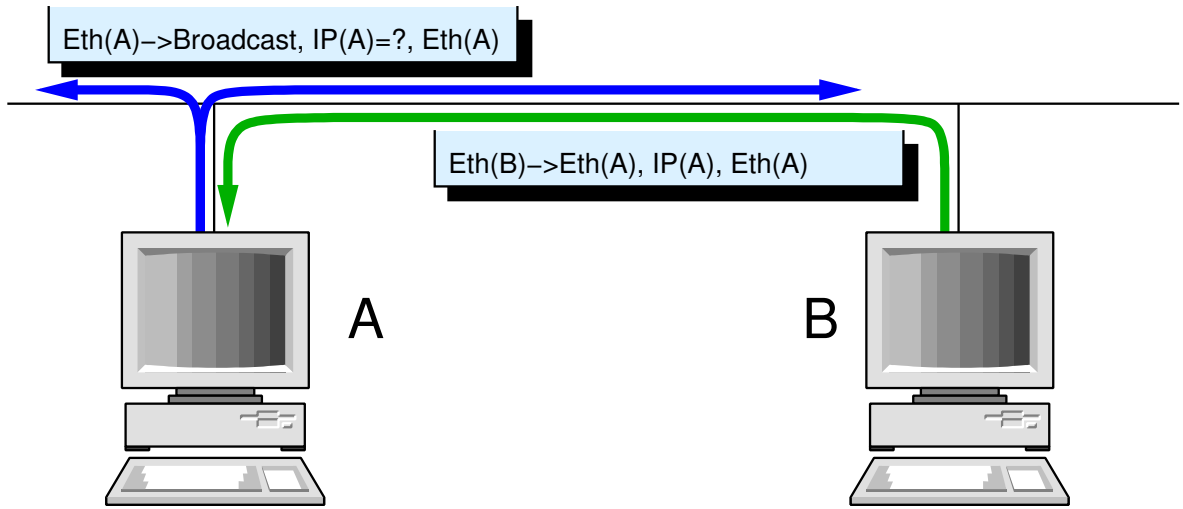
- utilisent UDP/IP
- peuvent fonctionner même si le serveur ne connaît pas l'adresse Ethernet de l'émetteur
- renvoient beaucoup plus d'informations (serveur de boot, routeur, serveur DNS, etc.)
- peuvent utiliser un *relais*

Deux protocoles :

- BOOTP (*Bootstrap Protocol*) :
  protocole original
- DHCP (*Dynamic Host Configuration Protocol*) :
  protocole postérieur à BOOTP, plus dynamique, compatible avec BOOTP

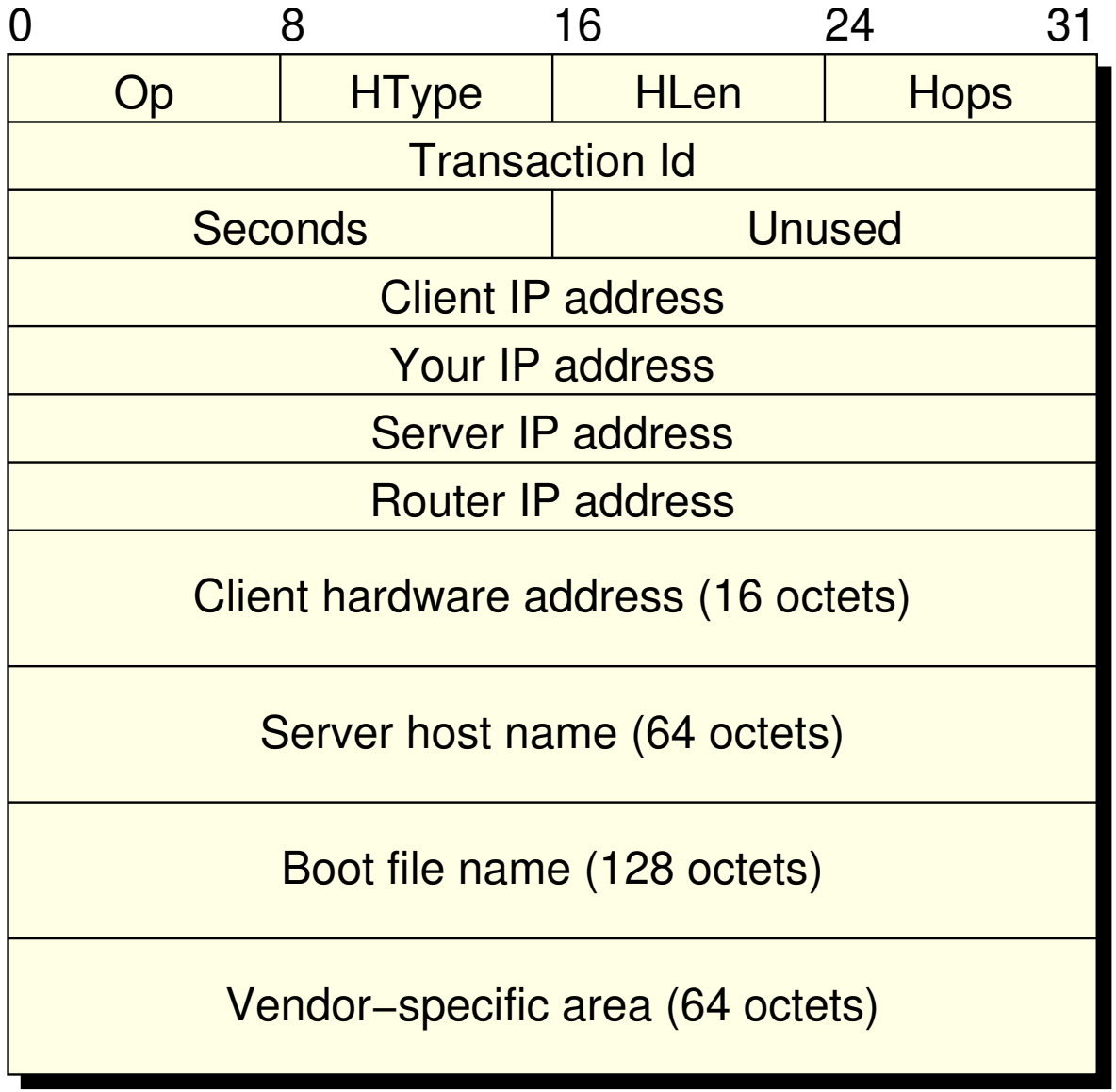
BOOTP

Principe de fonctionnement de BOOTP :



1. Message de A vers 255.255.255.255
   Indication = voici mon adresse Ethernet (Eth(A))
   Qui peut me donner les informations... ?
2. Message de B vers 255.255.255.255 (ou vers A)
   Réponse = voici les informations demandées

BOOTP


0
8
16
24
31
Op
HType
HLen
Hops
Transaction Id
Seconds
Unused
Client IP address
Your IP address
Server IP address
Router IP address
Client hardware address (16 octets)
Server host name (64 octets)
Boot file name (128 octets)
Vendor−specific area (64 octets)

BOOTP

Vendor Specific Area : information optionnelle passée au client

- 4 premiers octets : *magic cookie*
  $\Rightarrow$ définit le format de ce qui suit (dans la pratique, un seul *magic cookie* est utilisé : 99.130.83.99).
- après (si 99.130.83.99) viennent les options :

1 octet	type de l'option
1 octet	longueur éventuelle de ce qui suit
n octets	la valeur de l'option

  Exemples d'options :
  - masque de sous-réseau
  - adresses de routeurs supplémentaires
  - adresses de serveurs DNS
  - taille du fichier de boot
  - etc.

BOOTP

BOOTP utilise UDP $\Rightarrow$ perte possible de paquets

Cas pathologique : démarrage de 80 terminaux après une coupure de courant $\Rightarrow$ saturation du serveur

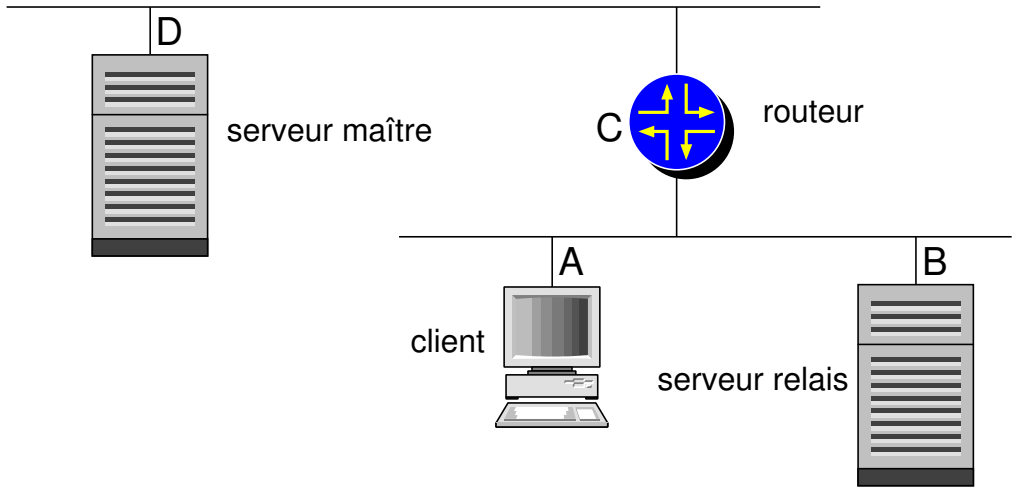
Solutions :

- le client démarre une horloge après l'émission d'une requête $\Rightarrow$ retransmission si perte
- le délai de retransmission initial est aléatoire (< 4 sec)
- les délais ultérieurs doublent à chaque retransmission
- au delà de 60 secondes, le délai n'est plus doublé

Toute la fiabilisation est à la charge du client
$\Rightarrow$ le serveur est très simple

BOOTP

Mécanisme de relais :



étape	de→vers	op	hops
1	A →255.255.255.255	1	0
2	B →D	1	1
3	D →B	2	?
4	B →255.255.255.255	2	?+1

Le serveur relais peut éventuellement être le routeur

DHCP

Avec BOOTP, l'adresse IP est fournie « définitivement »
$\Rightarrow$ affectation statique

Nouveaux besoins :

- terminaux mobiles
- raréfaction des adresses IP

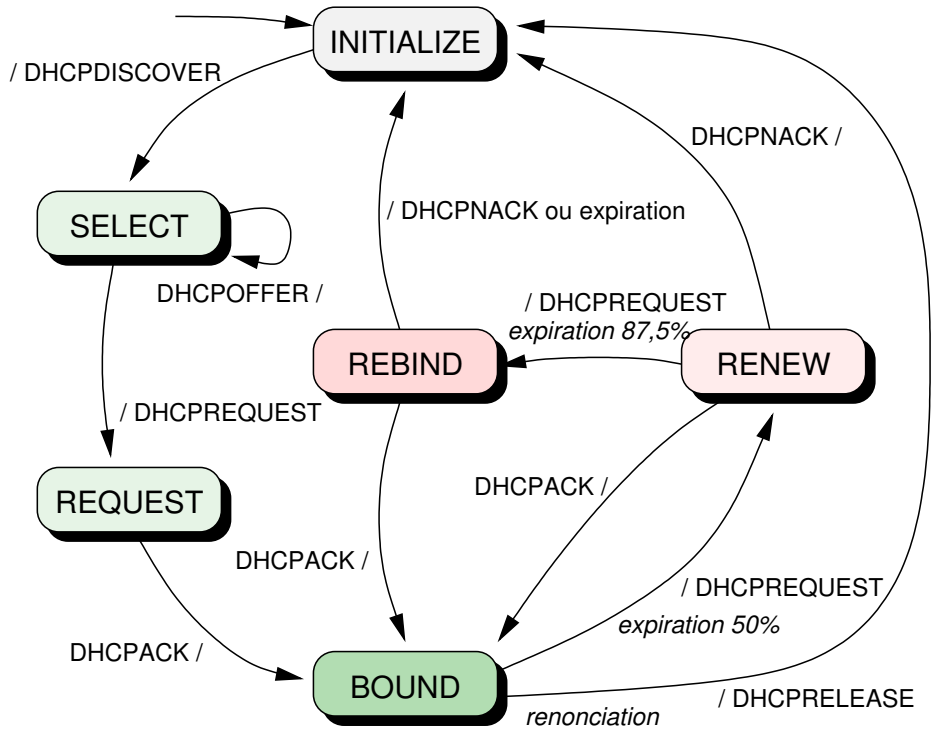
$\Rightarrow$ besoin de gestion plus dynamique des adresses : DHCP

Caractéristiques de DHCP :

- compatible avec BOOTP
- gestion des adresses Ethernet non répertoriées
- notion de « prêt » des adresses IP

DHCP

Prêt d'une adresse IP : diagramme d'états de DHCP



Notes :

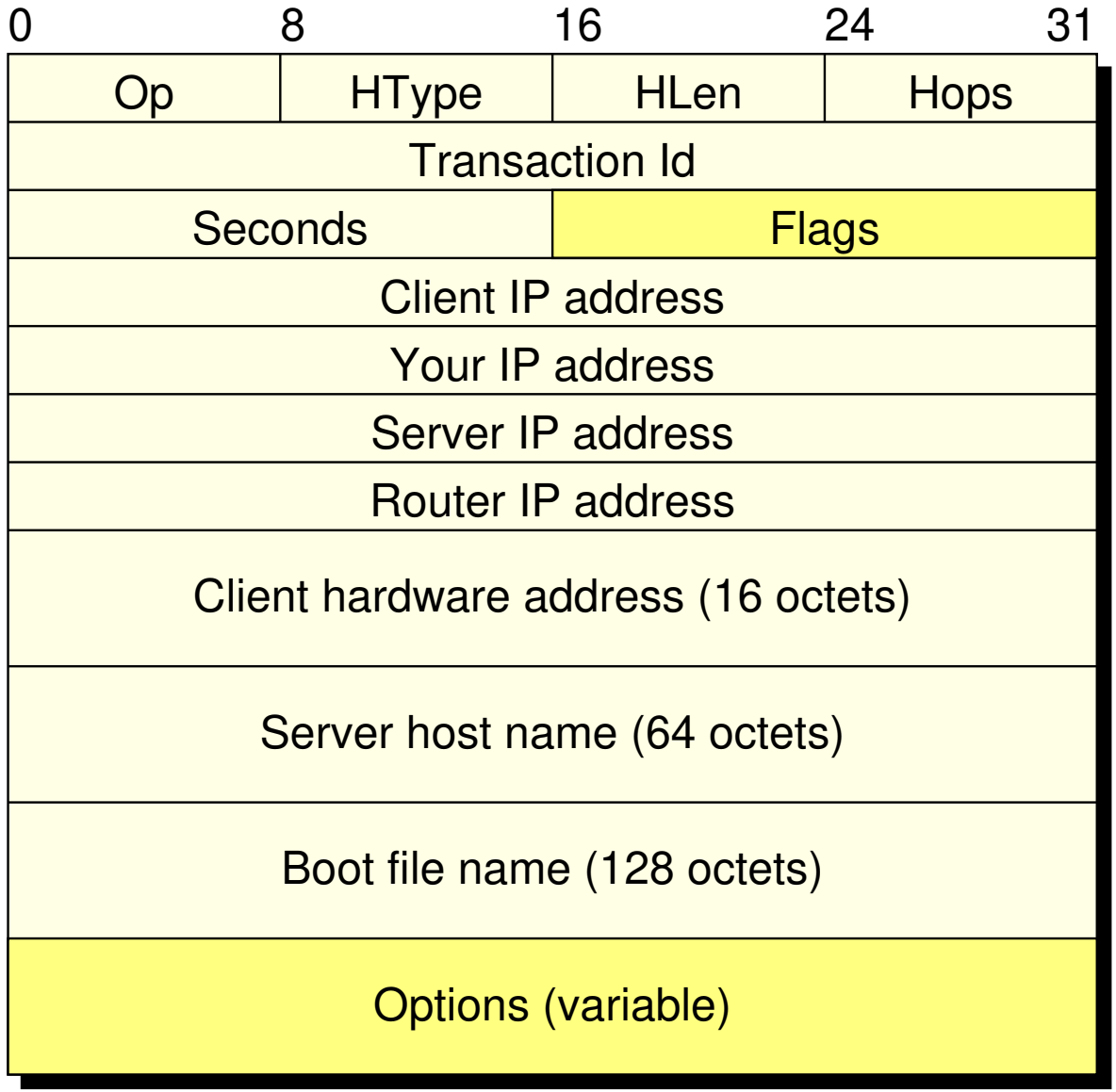
- *a*/*b* : réception de *a*, émission de *b*
- message de type *t* : *option* du message DHCP

DHCP

Problèmes ouverts :

- comment continuer les connexions TCP avec une adresse IP modifiée (suite à une expiration) ?
- comment nommer (dans le DNS) une machine avec une adresse dynamique ?
  - préallocation statique $\Rightarrow$ facile
  - enregistrement dynamique dans le DNS

DHCP


0
8
16
24
31
Op
HType
HLen
Hops
Transaction Id
Seconds
Flags
Client IP address
Your IP address
Server IP address
Router IP address
Client hardware address (16 octets)
Server host name (64 octets)
Boot file name (128 octets)
Options (variable)

Plan

De Telnet/FTP à SSH

Courrier électronique

BOOTP et DHCP

RPC/XDR et NFS

X-Window

RPC

- Présentation
- XDR
- RPC bas niveau
- Compilateur rpcgen

RPC - Présentation

Inconvénients des sockets :

- bas niveau (primitives `read` et `write`)
- oblige à définir et implémenter un protocole
  $\Rightarrow$ spécifier un format
- oblige à adapter les données
  $\Rightarrow$ gestion de l'ordre des octets

$\Rightarrow$ on aimerait avoir une abstraction de plus haut niveau

Modèle intéressant : l'appel de procédure à distance (RPC = *Remote Procedure Call*)

RPC - Présentation

RPC = appel de procédure à distance

Principe comparable à l'appel de procédure local :

- envoi des paramètres :
  $\Rightarrow$ par le réseau et non par la pile
- la procédure tourne :
  $\Rightarrow$ sur une autre machine
- l'appelant attend :
  $\Rightarrow$ sur la machine locale
- récupération de la valeur de retour
  $\Rightarrow$ par le réseau et non par un registre du processeur

$\Rightarrow$ la procédure distante ne doit pas utiliser de variables globales !

RPC – Présentation

Comment implémenter les RPC ?

- comment spécifier le protocole ?
  $\Rightarrow$ protocole = paramètres + valeur de retour
- comment spécifier le format des paramètres ?
  $\Rightarrow$ protocole de présentation des données
- comment nommer/contacter la procédure ?
  $\Rightarrow$ serveur de procédure
- comment « cacher » l'implémentation ?
  $\Rightarrow$ langage de description d'interface + compilateur
  $\Rightarrow$ « stubs »

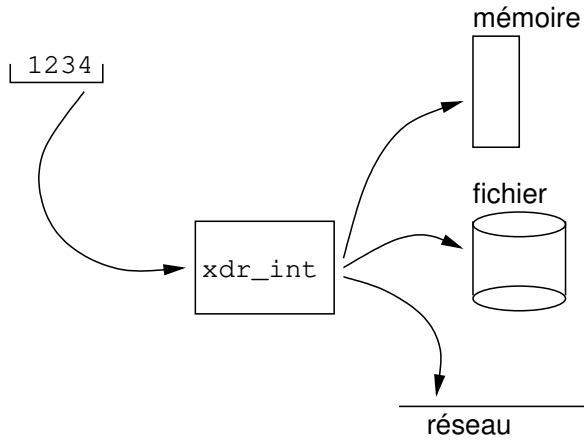
RPC – XDR

XDR = *eXternal Data Representation*

Principe : fonctions de conversion d'un type dans une représentation portable

Chaque type a sa fonction de conversion, qui :

- prend la valeur (son adresse) en paramètre
- envoie la valeur convertie dans un fichier (fichier sur disque, tube, socket, etc.)



RPC – XDR

Conversions des types de base :

- `bool`
- `enum`
- `char` `int` `long` `short` et leurs versions non signées
- `float` et `double`
- type « opaque »
- type spécial `void`

$\Rightarrow$ fonctions préprogrammées

RPC – XDR

Conversion des types complexes : chaînes de caractères, tableaux, structures, unions, pointeurs

$\Rightarrow$ construire les fonctions à partir des types de base.

Exemple :

```
struct machin { int a ; float b ; int c [MAX] ; } ;

bool_t xdr_machin (XDR *xdrs, struct machin *m)
{
    int n = MAX ;
    if (xdr_int (xdrs, & m->a) &&
            xdr_float (xdrs, & m->b) &&
            xdr_array (xdrs, & m->c, &n,
                       10, sizeof (int), xdr_int) )
        return TRUE ;
    return FALSE ;
}
```

RPC – Bas niveau

Nommage des procédures distantes :

- nom de machine
  (adresse IP ou nom symbolique)
- numéro de programme (groupe de procédures)
  entier sur 32 bits
- numéro de procédure dans le programme
  entier sur 32 bits
- numéro de version de la procédure
  entier sur 32 bits

RPC – Bas niveau

Liaison numéro de programme / port UDP ou TCP :

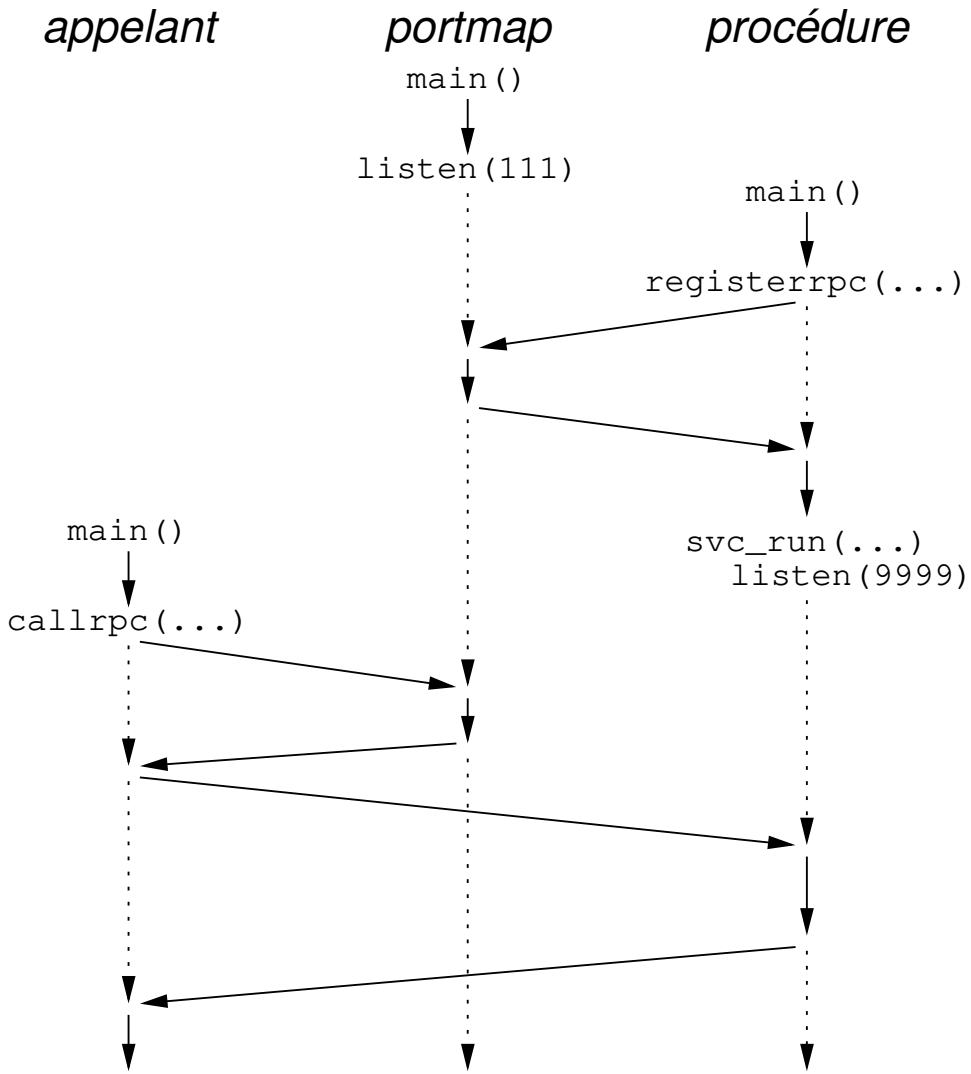
- chaque application connaît le numéro de programme mais pas le numéro de port
- il faut donc demander à quelqu'un qui sait...
  $\Rightarrow$ serveur de « renseignements »
  $\Rightarrow$ programme `portmap`, port 111 TCP et UDP

RPC – Bas niveau

Procédure en 2 temps :

- la procédure qui doit être appelée :
  - alloue un port TCP ou UDP
  - s'enregistre (avec le port) auprès de `portmap`,
  - attend une connexion sur ce port
- un programme voulant appeler cette procédure :
  - interroge `portmap`
    $\Rightarrow$ pour connaître le numéro de port de la procédure
  - envoie ensuite les données à ce port

RPC – Bas niveau



RPC – Bas niveau

Code de la procédure appelée :

```
main () {
    registerrpc (TOTOPROG, TOTOVERS, TOTOPROC_NUM,
        toto, xdr_int, xdr_double) ;
    svc_run () ;
}

char *toto (int *donnee) {
    static double s ;
    s = sqrt ((double) *donnee) ;
    return (char *) &s ;
}
```

RPC – Bas niveau

Code de l'appelant :

```
main (int argc, char *argv []) {
    double s ;
    callrpc (argv [1],
        TOTOPROG, TOTOVERS, TOTOPROC_NUM,
        xdr_int, atoi (argv [2]),
        xdr_double, &s) ;
    printf ("%lf", s) ;
}
```

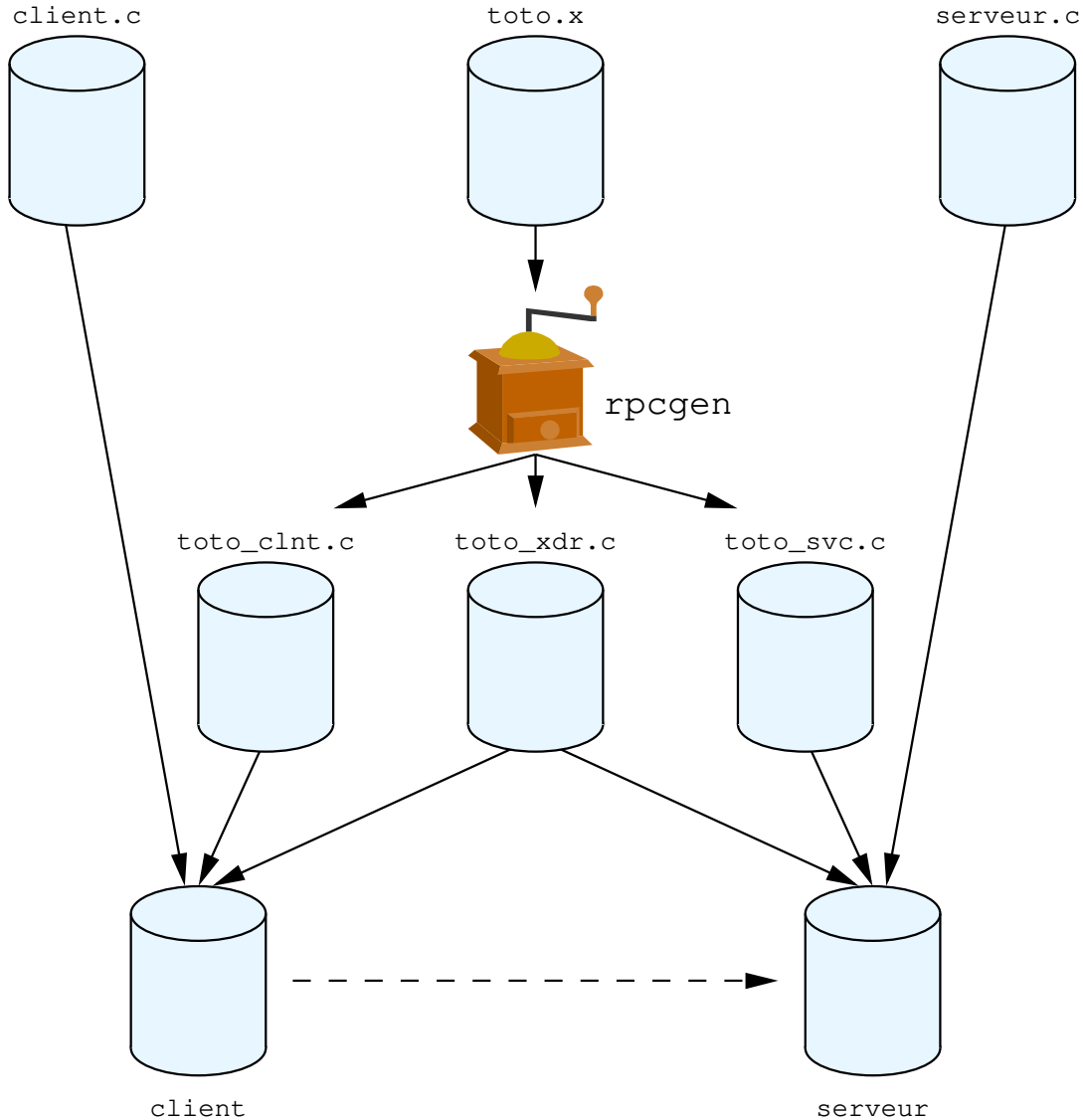
RPC – Compilateur

Problème : toute application utilisant les RPC doit réécrire du code :

- sur le serveur :
  - zéro, une ou plusieurs routines XDR
  - un `main` appelant `registerrpc`
- sur le client :
  - zéro, une ou plusieurs routines XDR
  - un `main` appelant `callrpc`

Idée : construire une seule spécification avec un langage de haut niveau, et ne coder en C que la partie « utile »

RPC – Compilateur



RPC - Compilateur

Exemple de spécification :

```
const MAX = 100 ;
typedef string machaine<MAX> ;

program TOTO {
    version TOTOVERS {
        double TOTO (int) = 1 ;
        int NOUVEAU (machaine) = 2 ;
    } = 1 ;
    version TOTOVERS2 {
        double TOTO (double) = 1 ;
    } = 2 ;
} = 99 ;
```

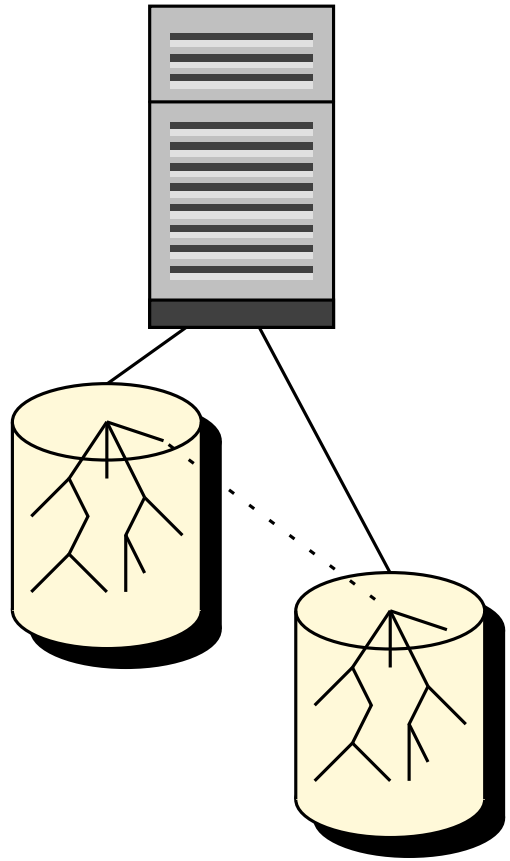
NFS

NFS (*Network File System*) :

- développé par Sun Microsystems en 1984
- indépendant du système d'exploitation
- standard *de facto*
- utilise les protocoles RPC (*Remote Procedure Call*) d'appel de procédure à distance et XDR (*eXternal Data Representation*) de présentation des données
- basé sur UDP
- protocole sans état

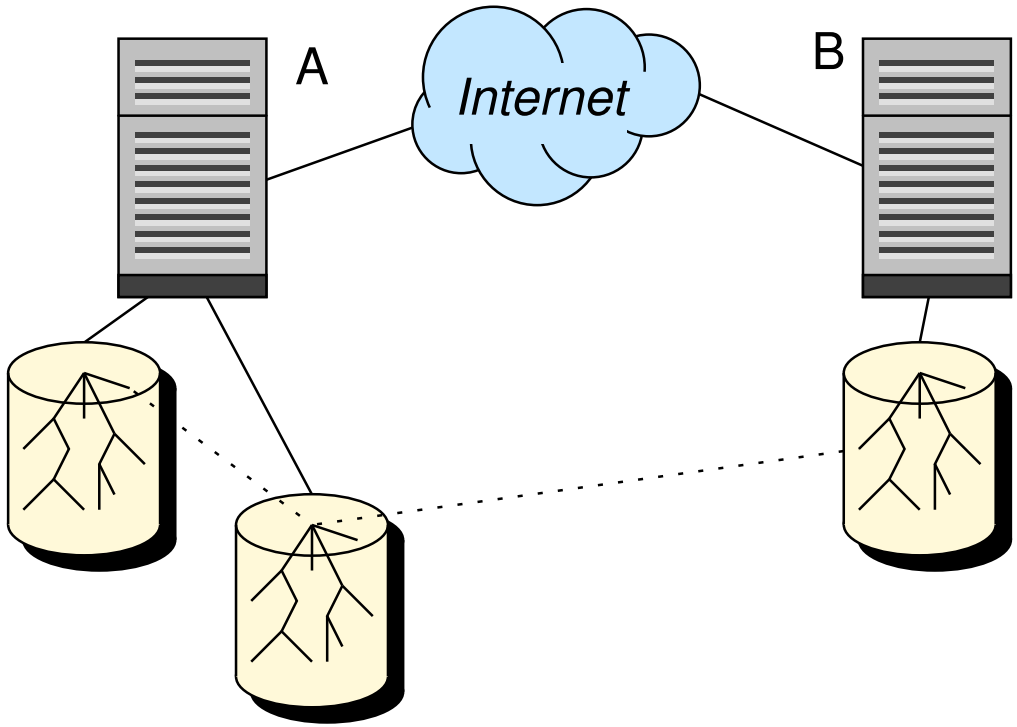
NFS

Notion de montage :



Le deuxième disque est monté $\Rightarrow$ il est « accroché » dans l'arborescence du premier disque.

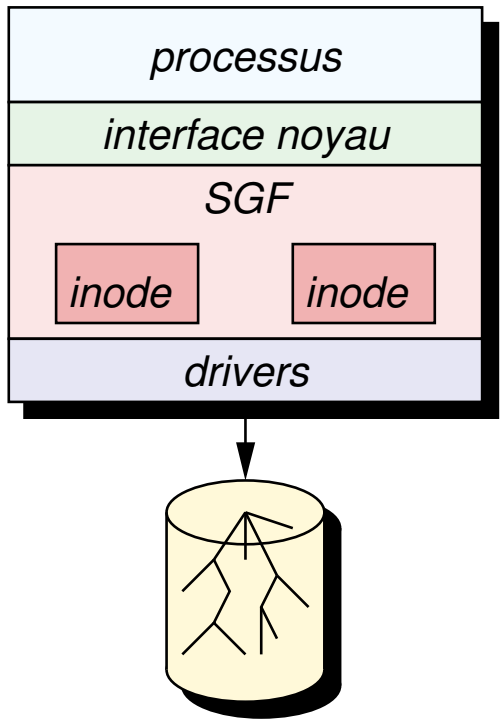
NFS



- La machine A *exporte* son deuxième disque
- La machine B *monte* le disque de la machine A

NFS

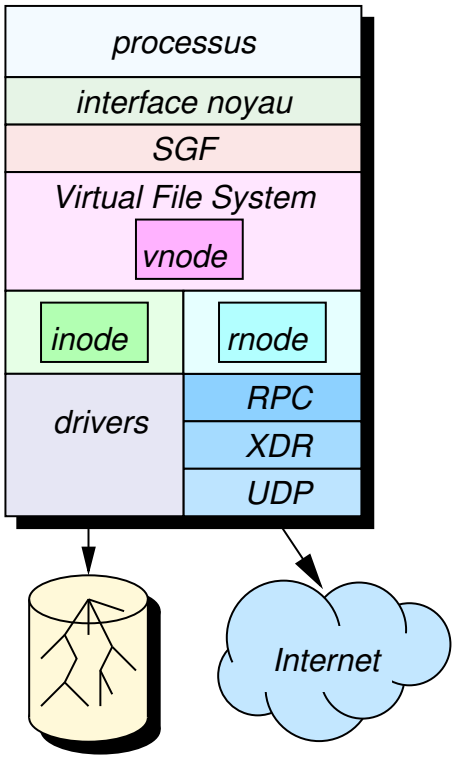
Schéma classique de l'accès au fichier dans Unix :



inode : mémorise les informations relatives à un fichier (propriétaire, droits, taille, localisation...)

NFS

Schéma modifié de l'accès au fichier dans Unix :



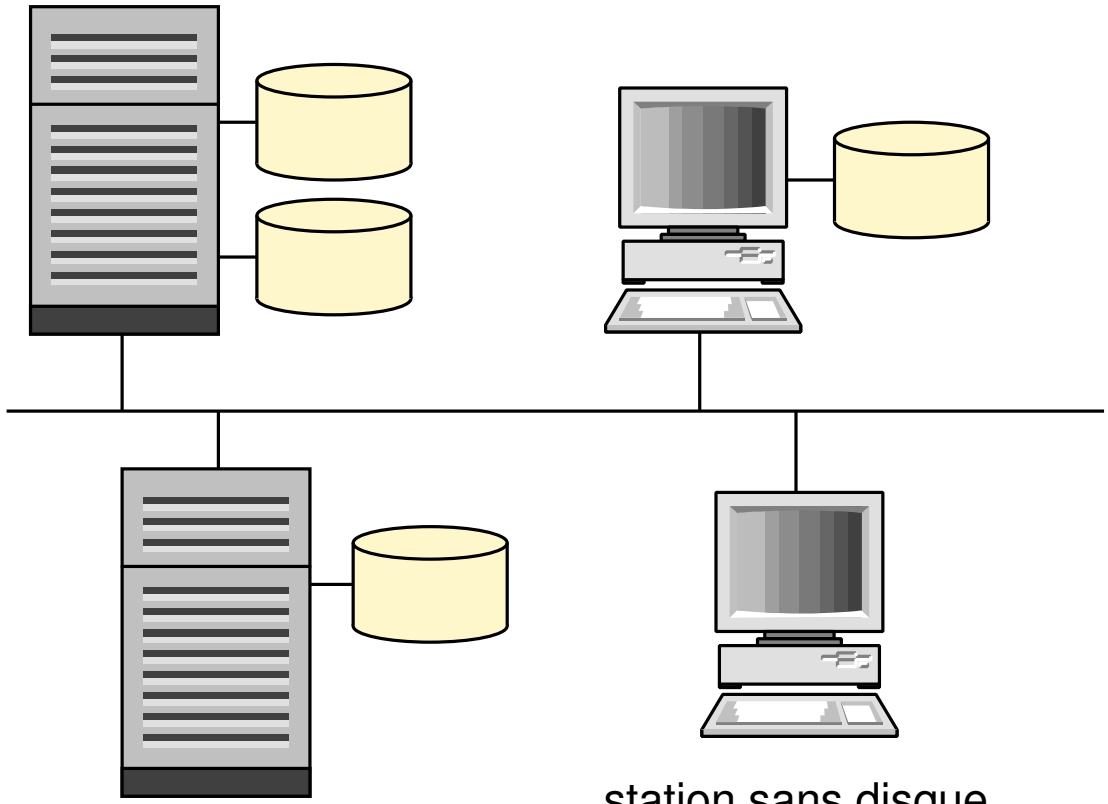
- *rnode* : mémorise les informations relatives à un fichier (attributs, localisation du serveur, référence au fichier *file handle*...)
- *vnode* : identifie l'*inode* ou *rnode* et les actions associées.
- NFS repose sur les *Remote Procedure Calls* de Sun

NFS

Actions du *Virtual File System* :

Procédures	Signification
getattr	lit les attributs du fichier
setattr	modifie les attributs du fichier
lookup	renvoie le *file handle* associé à un fichier
readlink	lit un lien symbolique
read	lit des données
write	écrit des données
create	crée un fichier
remove	détruit un fichier
rename	renomme un fichier
link	crée un lien physique
symlink	crée un lien symbolique
mkdir	crée un répertoire
rmdir	détruit un répertoire
readdir	lit un répertoire
statfs	lit les attributs d'un système de fichiers

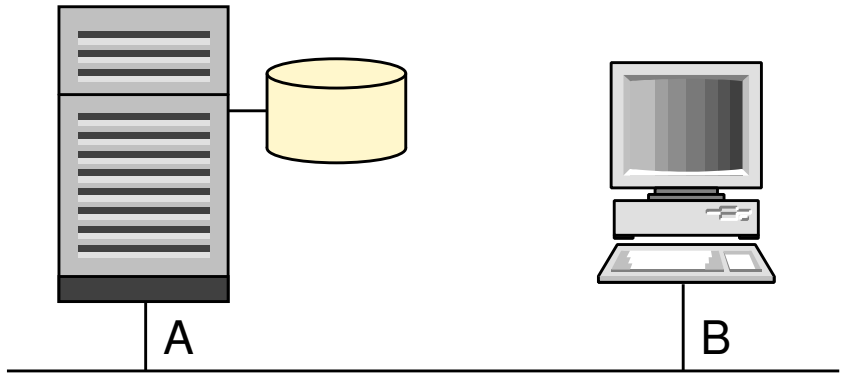
Diskless



- comment fait la station sans disque pour démarrer ?
- comment fait-elle pour connaître son adresse IP ?
- comment fait-elle pour connaître son serveur ?

Diskless

Implémentation Sun :



1. B utilise RARP
2. A répond (nécessite table Ethernet $\rightarrow$ IP)
3. B charge un *chargeur secondaire* par TFTP depuis A
4. B utilise le protocole Sun *bootparam* pour connaître son serveur de boot (*broadcast*)
5. A répond en lui donnant la localisation du serveur
6. B monte (NFS) le disque de A comme sa racine
7. B charge `/vmunix`
8. B a démarré

Plan

De Telnet/FTP à SSH

Courrier électronique

BOOTP et DHCP

RPC/XDR et NFS

X-Window

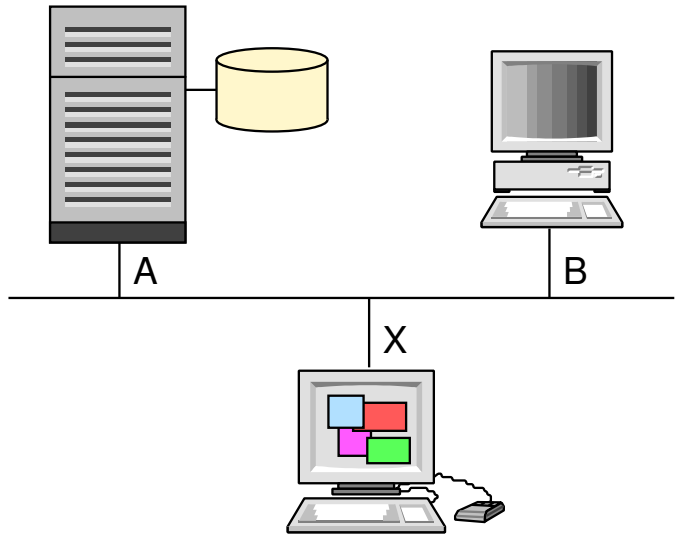
X Window

X Window System est un système de fenêtrage, mais c'est avant tout un protocole réseau.

- développé au MIT depuis 1987 (projet Athena)
- pas de politique d'interface homme-machine
- X11R*n* = version 11 du protocole
- utilise TCP, mais n'est pas lié à TCP
- création du MIT Consortium
- distribution *gratuite*

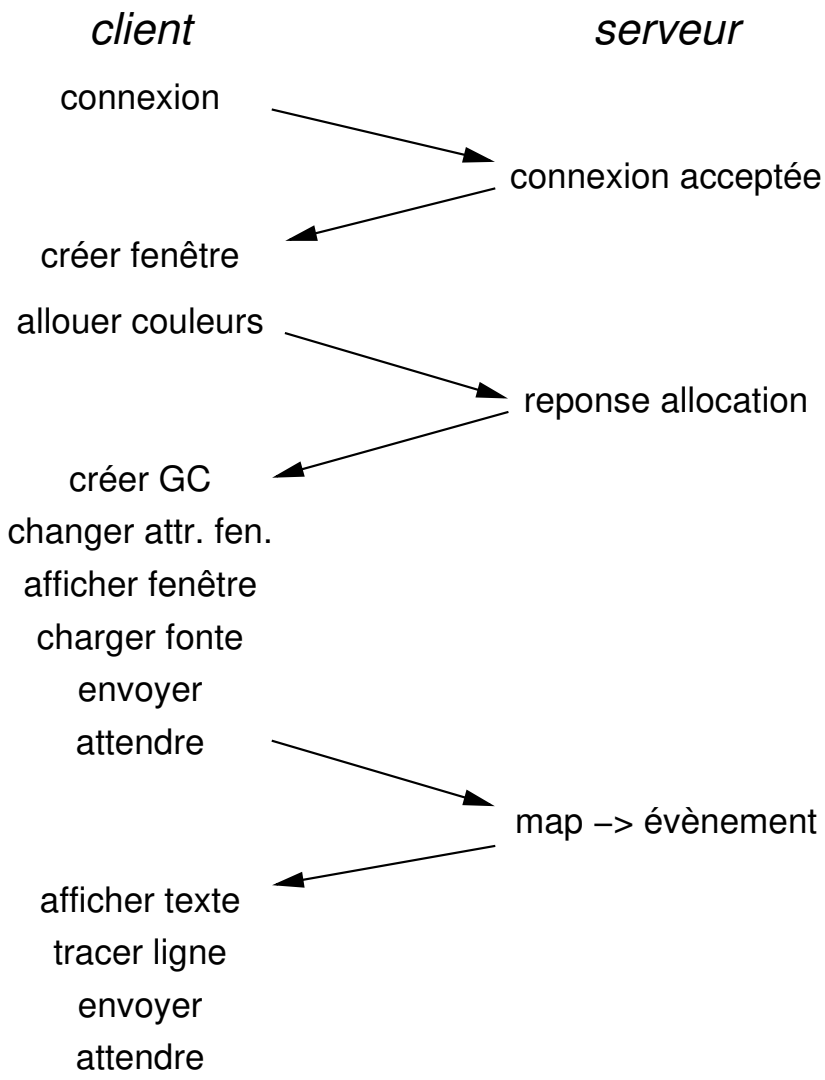
X Window

Terminologie :



- Le programme qui tourne sur *X* est le *serveur*. Il gère :
  - l'écran
  - le clavier
  - la souris
  - le réseau
- Les programmes qui tournent sur *A* et *B* et utilisent *X* sont les clients

X Window


client
serveur
connexion
connexion acceptée
créer fenêtre
allouer couleurs
reponse allocation
créer GC
changer attr. fen.
afficher fenêtre
charger fonte
envoyer
attendre
map –> évènement
afficher texte
tracer ligne
envoyer
attendre

X Window

L'écran est identifié par: *host*:*display*.*seat*

host	nom ou adresse du serveur
display	numéro du poste de travail
seat	numéro de l'écran sur ce poste

Exemple : `vagabond.u-strasbg.fr:0.0`

Connexion :

- le serveur attend les connexions en TCP sur le port 6000 + display
- négociation du format des données (ordre des octets, alignement des mots), sauf pour images
- authentification

X Window

Authentification : deux procédés

- validation par adresse IP $\Rightarrow$ peu de sécurité
- validation par clef (*authorizations*)
  La clef est connue seulement du serveur et des clients. Elle réside dans un fichier protégé (`$HOME/.Xauthority`)
  $\Rightarrow$ sécurité de X = sécurité du système d'exploitation