U.E. ARES
Architecture des Réseaux

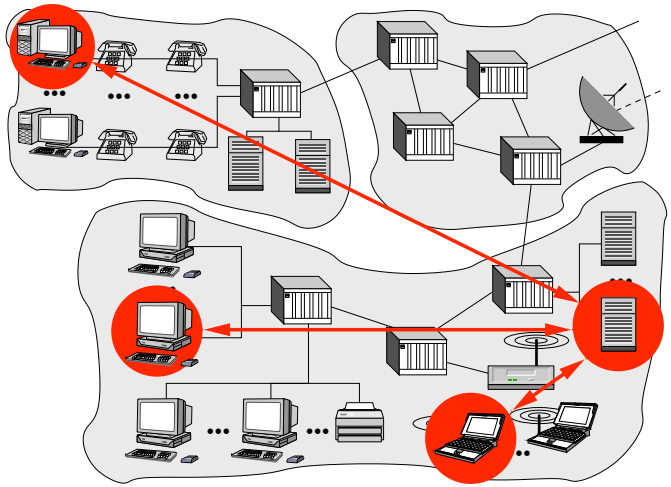
Cours 2/6 : Applications

Olivier Fourmaux
(olivier.fourmaux@upmc.fr)

Version 5.4

Plan

Applications



Couche Application

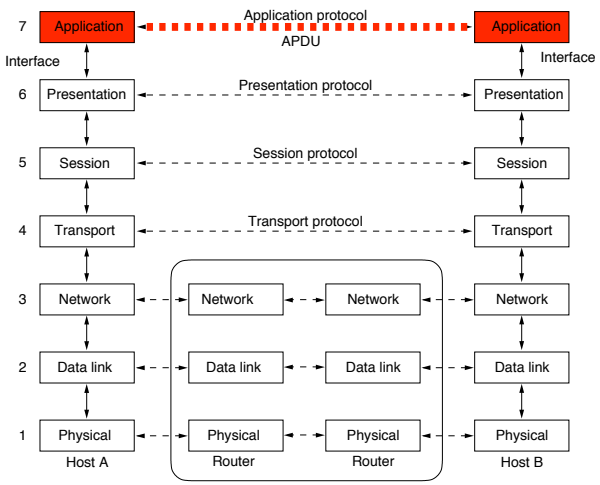
Définition :

La couche application contient les protocoles de haut niveau qui permettent aux logiciels utilisateurs de communiquer

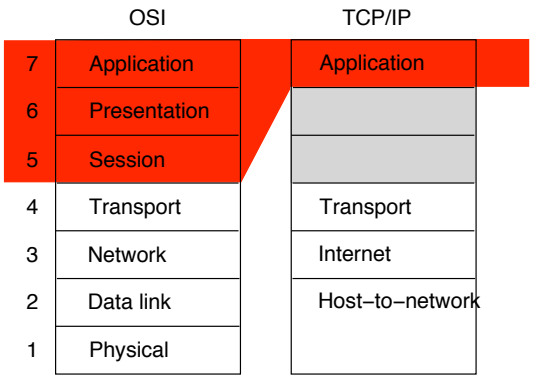
Remarques :

- standardise les échanges entre les applications les plus courantes
  - ✓ accès au web (HTTP), envoi d'*e-mail* (SMTP, POP, IMAP) ...
    - ☞ applications $\neq$ protocoles de la couche application
- définit l'interface réseau avec les utilisateurs
  - ✓ s'appuie sur les services de bout-en-bout définis dans les couches inférieures
- supporte les environnements hétérogènes

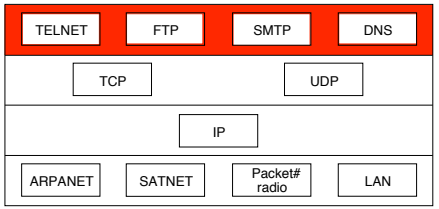
Couche Application : modèle OSI



Couche Application : modèle TCP/IP (1)



Couche Application : modèle TCP/IP (2)



- Dans l'Internet, des centaines de protocoles applicatifs existent !
  TELNET pour contrôler une machine à distance
  FTP pour transférer des données
  SMTP pour échanger du courrier électronique
  HTTP pour surfer sur la toile
  DNS pour convertir les noms de l'Internet
  SNMP pour administrer le réseau
  ...

Plan

Connexion à distance

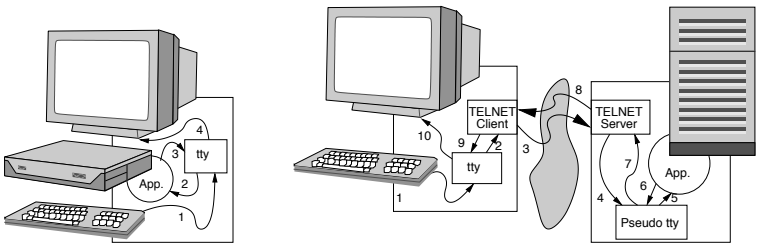
A partir d'un terminal ouvert sur une machine, connexion sur d'autres machines (environnement système hétérogène)

- plusieurs protocoles :
  - ✓ TELNET
  - ✓ RLOGIN
  - ✓ SSH
  - ✓ ...
- application de type client/serveur
  - ✓ **client** : interagit avec l'utilisateur et les protocoles réseaux
  - ✓ **serveur** : interagit avec les protocoles réseaux et l'application distante
- besoin d'interactivité
  - ✓ tout ce qui tapé au clavier est envoyé **rapidement** sur la connexion
  - ✓ tout ce qui est reçu de la connexion est affiché **rapidement** sur l'écran

TELNET

TELecommunication NETwork protocol

- application standard de l'Internet depuis 1969 ! (RFC 854)
- repose sur une connexion **TCP** (port serveur = **23**)
- mécanisme de négociation d'options
- service de terminal virtuel
- pas de confidentialité (mot de passe **en clair** pour un *login shell*)



TELNET : Options

Plusieurs échanges initiaux pour les options :

- le client émet des **requêtes** d'option (`WILL WON'T DO DON'T`)

```
Command: Do Suppress Go Ahead
Command: Will Terminal Type
Command: Will Negotiate About Window Size
Command: Will Terminal Speed
Command: Will Remote Flow Control
...
```

- le serveur renvoie des **réponses** d'options (`DO DON'T WILL WON'T`)

```
Command: Do Terminal Type
Command: Will Suppress Go Ahead
Command: Dont Negotiate About Window Size
Command: Do Terminal Speed
Command: Dont Remote Flow Control
...
```

- chaque extrémité implémente une version minimale du NVT
  - ✓ négociation d'options pour les machines plus évoluées

TELNET : NVT

Définition d'un terminal virtuel (*Network Virtual Terminal*)

- pas de format de message, mais un codage après la phase de négociation
- codage vers un système de représentation commun : NVT
  - ✓ chaque système peut transcoder

terminal local réel ⇔ terminal réseau virtuel

  - ✓ Exemple :
    - ☞ *local :* `cc maa<bs>x.c`
    - ☞ *NVT :* `c` `.` `x` `IAC EC` `a` `a` `m` ␣ `c` `c` ⇻

      IAC = *Interpret As Command (ASCII code 255)*
    - ☞ il n'est pas nécessaire de connaitre la conversion vers chaque type de machine
  - ✓ permet la communication dans les **environnement hétérogènes**
  - ✓ contrôle **in-band**

TELNET : Accès à d'autres serveurs

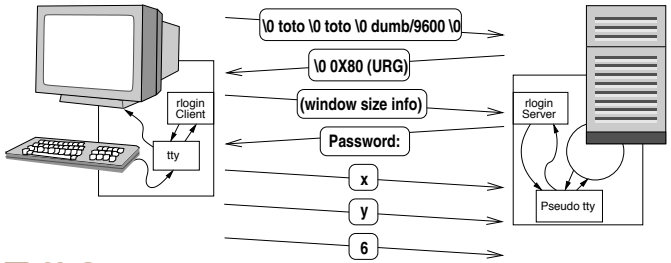
Exemple d'accès à un serveur web :

```
Unix> telnet hobbes.lip6.fr 80
          Trying 137.86.111.77...
          Connected to hobbes.lip6.fr.
          Escape character is '^]'.
GET /index.html HTTP/1.0

          HTTP/1.1 200 OK
          Date: Tue, 24 Sep 2002 15:33:07 GMT
          Server: Apache/1.3.9 (Unix) Debian/GNU
          Last-Modified: Sat, 29 Apr 2000 07:07:45 GMT
          Content-Length: 4094
          Connection: close
          Content-Type: text/html; charset=iso-8859-1

          <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
          <HTML>
          ...
          </HTML>
          Connection closed by foreign host.
Unix>
```


RLOGIN

Remote LOGIN

- application standard d'Unix BSD (RFC 1282)
- beaucoup plus simple que TELNET, pas de négociation
- repose sur une connexion **TCP** (port serveur = **513**)
- quelques commandes *in-band* en données urgentes
- pas de confidentialité (mot-de-passe **en clair**) et confiance (`.rhost`)



SSH

Secure SHell

- **communications cryptées**, assure :
  - ✓ authentification
  - ✓ confidentialité
  - ✓ intégrité
  - ✓ (compression)
- repose sur une connexion **TCP** (port serveur = **22**)
  - ✓ rajoute une couche transport intermédiaire
  - ✓ authentification cryptée
  - ✓ négotiation des algorithmes
  - ✓ multiplexage de plusieurs flux dans la connexion
- standardisation tardive (janvier 2006) : RFCs 4251 à 4254
- nombreuses implémentations dont OpenSSH (OpenBSD...)

Plan

Transfert de fichiers

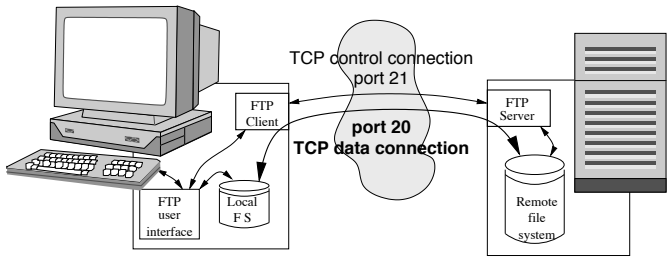
Copie d'un fichier d'un système vers un autre en environnement hétérogène

- plusieurs protocoles :
  - ✓ FTP
  - ✓ TFTP
  - ✓ RCP, SCP, SFTP
  - ✓ ...
- application de type client/serveur
  - ✓ **client** : interagit avec l'utilisateur, le système de fichier local et les protocoles réseaux
  - ✓ **serveur** : interagit avec les protocoles réseaux et le système de fichier distant
- ne pas confondre avec les systèmes de fichiers distants
  - ✓ NFS (Sun, TCP/IP)
  - ✓ SMB (Mircosoft)

FTP

File Transfer Protocol

- standard TCP/IP pour le transfert de fichiers (RFC 959)
- signalisation **out-of-band**, deux connexions **TCP**
- accès interactif
- contrôle d'accès (mais mot de passe **en clair**)



FTP : Connexions

Deux connexions **TCP** sont utilisées en parallèle :

- connexion de **contrôle**
  - ✓ **permanente** (créée à l'ouverture de la session FTP)
  - ✓ full duplex initiée par le client (port serveur = **21**)
  - ✓ utilisée uniquement pour échanger les **requêtes** et **réponses**
  - ✓ besoin d'**interactivité** (et de fiabilité)
- connexion de transfert de **données**
  - ✓ **temporaire** (créée à chaque transfert de fichier)
  - ✓ full duplex initiée par :
    - ☞ le **client** pour un envoi vers le serveur (port **20**)
    - ☞ le **serveur** pour un envoi vers le client (port **? ?**)
      - ➠ transmission du port client à utiliser
  - ✓ envoi de fichiers et de liste de fichiers/répertoires
  - ✓ besoin de **débit** (et de fiabilité)
  - ✓ libérée à la fin de chaque tranfert de fichier

FTP : Données

Nombreuses représentations des données (liée à l'hérogènéité des machines) :

- type de fichiers :
  - ✓ **non structurés**
  - ✓ enregistrements
  - ✓ pages
- encodage des données :
  - ✓ **ASCII** (*American Standard Code for Information Interchange*)
  - ✓ EBCDIC (*Extended Binary-Coded Decimal Interchange Code*)
  - ✓ **binaire**
- type de transmission :
  - ✓ **flux**
  - ✓ bloc
  - ✓ compressé

➠ **vérifier le type de données transférées**

FTP : Requêtes

Codage ASCII NVT

➠ Mode interactif possible (lisible)

```
Unix> telnet galion.ufr-info-p6.jussieu.fr 21
       Trying 197.18.176.12...
       Connected to localhost.
       Escape character is '^]'.
       220 ProFTPD 1.2.0pre10 Server (Debian) [galion.ufr-info-p6.jussieu.fr]
help
       214-The following commands are recognized (* =>'s unimplemented).
       214-USER    PASS    ACCT*   CWD     XCWD    CDUP    XCUP    SMNT*
       214-QUIT    REIN*   PORT    PASV    TYPE    STRU*   MODE*   RETR
       214-STOR    STOU*   APPE    ALLO*   REST    RNFR    RNTO    ABOR
       214-DELE    MDTM    RMD     XRMD    MKD     XMKD    PWD     XPWD
       214-SIZE    LIST    NLST    SITE    SYST    STAT    HELP    NOOP
       214 Direct comments to root@galion.ufr-info-p6.jussieu.fr.
       421 Login Timeout (300 seconds): closing control connection.
       Connection closed by foreign host.
Unix>
```


Commandes utilisateur du programme `ftp`

Ne pas confondre avec les commandes du protocole FTP

```
Unix> ftp
ftp> help
      Commands may be abbreviated.  Commands are:
      !               debug           mdir            sendport        site
      $               dir             mget            put             size
      account         disconnect      mkdir           pwd             status
      append          exit            mls             quit            struct
      ascii           form            mode            quote           system
      bell            get             modtime         recv            sunique
      binary          glob            mput            reget           tenex
      bye             hash            newer           rstatus         tick
      case            help            nmap            rhelp           trace
      cd              idle            nlist           rename          type
      cdup            image           ntrans          reset           user
      chmod           lcd             open            restart         umask
      close           ls              prompt          rmdir           verbose
      cr              macdef          passive         runique         ?
      delete          mdelete         proxy           send
ftp> quit
Unix>
```


FTP : Réponses

Codage usuel : *status* + *description* (lisible)

- 150 Opening BINARY mode data connection
- 200 Command successful
- 220 ProFTPD 1.2.0pre10 Server (Debian)
- 226 Transfer complete
- 230 User toto logged in
- 331 Username OK, Password required
- 425 Can't open data connection
- 500 Syntax error (Unknown command)
- ...

status	*description*	*status*	*description*
		x0z	syntaxe
1yz	réponse positive préliminaire	x1z	information
2yz	réponse positive complète	x2z	connexions
3yz	réponse positive intermédaire	x3z	authentification
4yz	réponse négative transitoire		
5yz	réponse négative définitive	x5z	système de fichier

FTP : Exemple

Programme `ftp` (interface utilisateur) / Protocole FTP (connexion de contrôle)

```
[toto@hobbes]$ ftp calvin.lip6.fr
   Connected to calvin.lip6.fr.
   220 FTPD 1.2pre8 Server (Debian)          220 FTPD 1.2pre8 Server (Debian)
Name (calvin.lip6.fr):toto                   USER toto
   331 Password required for toto.              331 Password required for toto.
Password:                                    PASS AB]Ga!9F
   230 User toto logged in.                     230 User toto logged in.
ftp> get toinst.txt
   local: toinst.txt remote: toinst.txt
                                             PORT 192,33,82,12,4,15
   200 PORT command successful.                 200 PORT command successful.
                                             RETR toinst.txt
   150 Opening BINARY mode data connection      150 Opening BINARY mode data connection
       for toinst.txt (1 bytes).                    for toinst.txt (1 bytes).
   226 Transfer complete.                       226 Transfer complete.
   1 bytes received in 0.377s (0.0026 KB/s)
ftp> quit                                    QUIT
   221 Goodbye.                                 221 Goodbye.
[toto@hobbes]$
```

FTP : Divers

Anonymous

- compte invité sur certains serveurs FTP :
  - ✓ `username : anonymous`
  - ✓ `password :` *adresse@electronique.org*

Mode passif

- impossibilité de créer la **connexion donnée** à partir du serveur
  - ✓ filtrage des adresses (*firewall*)
  - ✓ translation d'adresses (NAT)
- intégré dans les *browsers*
  - ➠ création en sens inverse de la connexion donnée
  - ✓ commande `PASV` (RFC 1579)
    - ☞ le client envoie `PASV a,b,c,d,x,y` au serveur
    - ☞ le serveur fait une ouverture passive sur le port 256x+y et en informe le client
    - ☞ le client fait une ouverture active par le port 256x+y

TFTP

Trivial File Transfer Protocol

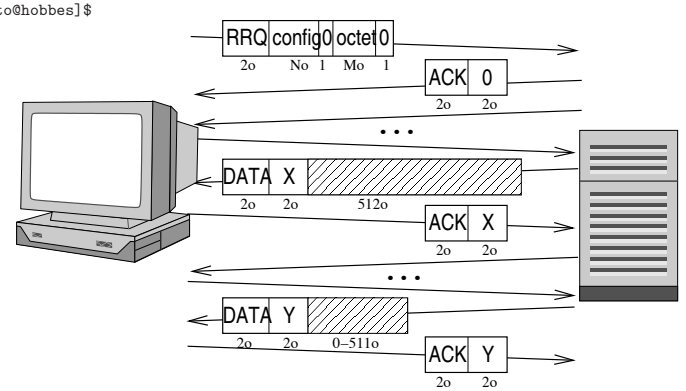
- protocole léger pour le transfert de fichiers (version 2 : RFC 1350)
- datagrammes **UDP** sur le port 69
- 5 messages :

opcode	nom	description
1	RRQ	requête de lecture
2	WRQ	requête d'écriture
3	DATA	données numérotées
4	ACK	acquittement
5	ERREUR	message d'erreur

- messages DATA avec 512 octets (sauf le dernier de taille inférieure ou éventuellement nulle)
- protocole *stop-and-wait*
  - ✓ numérotation des messages DATA
  - ✓ acquittement immédiat avec ACK
- pas de contrôle d'accès (sous Unix, souvent limité à `/tftpboot`)

TFTP : Exemple

```
[toto@hobbes]$ tftp calvin.lip6.fr
tftp> get config
   Received 5220 bytes in 0.377 secs
tftp> quit
[toto@hobbes]$
```



RCP, SCP, SFTP

Protocole R* : **RCP**

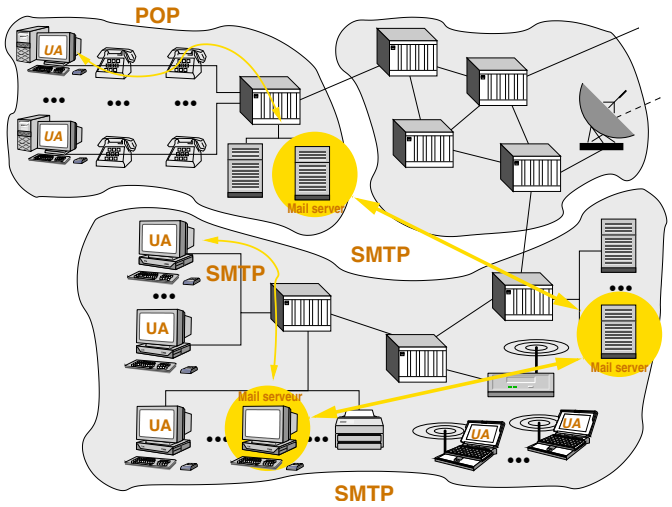
- spécifique à Unix et associé aux `r*` commandes (dont `rcp`)
  - ✓ le client `rcp` fonctionne avec un serveur `rshd`
  - ✓ idem `rlogin` : obsolète, problèmes de sécurités...

Protocoles sécurisés : **SCP, SFTP**

- `scp` : copie simple similaire à `rcp` encapsulé dans SSH
- `sftp` : idem FTP mais facilement encapsulable
  - ✓ SFTP est un nouveau protocole (groupe IPSEC de l'IETF)
  - ✓ SFTP peut être utilisé avec SSH (par défaut avec de nombreux clients `sftp`)
  - ✓ SFTP est différent de FTPS qui introduit la sécurisation au niveau des connexions avec SSL/TLS *(Secure Socket Layer/Transport Layer Security)*

Plan

Courrier électronique



SMTP : introduction

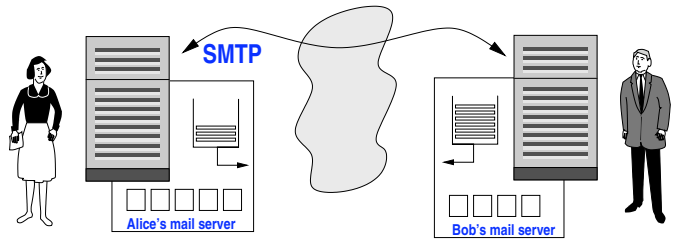
Echange de messages asynchrones à travers l'Internet

- l'ancienne "*killer app.*"
- trois éléments de base :
  - ✓ UA (*User Agent*)
    - ☞ `mail`, `elm`, `pine`, `mutt`...
    - ☞ Eudora, Outlook et MS Mail, Mail.app, Mozilla Thunderbird...
  - ✓ serveurs de mail ou MTA (*Mail Transfer Agent*)
    - ☞ `sendmail`...
    - ☞ compose l'**infrastructure** du système de distribution
    - ☞ **boites aux lettres** des utilisateurs locaux
    - ☞ file d'attente des messages au départ ou en transit
    - ☞ temporisation et **reprise** si destinataire inaccessible
  - ✓ un protocole : **SMTP**

SMTP : principes

Simple Mail Transfer Protocol (RFC821)

- application client/serveur
- repose sur le service fiable des connexions **TCP**
- ancien
  - ✓ + largement répandu
  - ✓ – messages encodées en ASCII NVT
- connexion aux serveurs mail sur le port **25**



SMTP : exemple

```
   220 hobbes.lip6.fr SMTP Sendmail 8.9.3; Wed, 22 Sep 2002 00:59:49 +0200
HELO calvin.lip6.fr
   250 hobbes.lip6.fr Hello calvin.lip6.fr, pleased to meet you
MAIL FROM: lechef@hobbes.lip6.fr
   250 lechef@hobbes.lip6.fr... Sender ok
RCPT TO: totu@hobbes.lip6.fr
   550 totu@hobbes.lip6.fr... User unknown
RCPT TO: toto@hobbes.lip6.fr
   250 toto@hobbes.lip6.fr... Recipient ok
DATA
   354 Enter mail, end with "." on a line by itself
Paris, le 22 septembre 2002.
Cher Toto,
Venez donc me voir dans mon bureau pour discuter
de votre prochaine augmentation.
                  Le chef.
.
   250 BAA01090 Message accepted for delivery
QUIT
   221 hobbes.lip6.fr closing connection
```


SMTP : commandes (1)

Serveur SMTP en mode interactif :

```
Unix> telnet galion.ufr-info-p6.jussieu.fr 25
   Trying 192.133.82.123...
   Connected to galion.ufr-info-p6.jussieu.fr
   Escape character is '^]'.
   220 galion.ufr-info-p6.jussieu.fr SMTP Sendmail 8.9.3; Wed, 25 Sep 2002 00:54:15 +0200
help
   214-This is Sendmail version 8.9.3
   214-Topics:
   214-    HELO    MAIL    RCPT    DATA
   214-    QUIT    VRFY    NOOP    RSET
   214-    HELP
   214-For more info use "HELP <topic>".
   214-To report bugs in the implementation send email to
   214-    sendmail-bugs@sendmail.org.
   214-For local information send email to Postmaster at your site.
   214 End of HELP info
quit
   221 galion.ufr-info-p6.jussieu.fr closing connection
   Connection closed by foreign host.
Unix>
```


SMTP : commandes (2)

Commandes SMTP de base (RFC 821), ensemble minimal :

`HELO`	Présentation du nom de domaine du client
`MAIL`	Identification de l'expéditeur du message
`RCPT`	Identification du destinataire du message
`DATA`	Envoi du contenu jusqu'à une ligne avec seulement un "."
`QUIT`	Termine l'échange de courrier
`VRFY`	Vérification de l'adresse du destinataire
`NOOP`	Pas d'opération, force le serveur à répondre
`RSET`	Annule la transaction

SMTP : réponses

Codage lisible usuel :

- *status + destription* :
  - ✓ `220 SMTP Sendmail 8.9.3`
  - ✓ `221 Closing connection`
  - ✓ `250 Command successful`
  - ✓ `354 Enter mail, end with "." on a line by itself`
  - ✓ `550 User Unknown`

SMTP : format des messages initiaux

Messages codés en ASCII NVT (RFC 822)

- **l'enveloppe**
  - ✓ modifiée par entités SMTP successives
    - ☞ commandes `MAIL FROM:` et `RCPT TO:`
- **le message**
  - ✓ principalement inséré par l'agent utilisateur
    - ☞ commande `DATA`
  - ✓ **entête**
    - ☞ chaque champ sur une ligne ➠ *nom*: *valeur*

```
From: Toto at Paris 13 <toto@galere.univ-paris13.fr>
Date: Mon, 22 Sep 2003 01:13:20 +0200
To: Titi at Paris 6 <titi@hypnos.lip6.fr>
Subject: rapport TER
X-Scanned-By: isis.lip6.fr
```

  - ✓ une ligne vide
  - ✓ **corps**
    - ☞ terminaison par une ligne avec seulement " ."

Evolution de l'enveloppe : ESMTP

Quelques commandes ESMTP (RFC 1425) :

`EHLO`	Utilisation de ESMTP et présentation du client
`SIZE`	Taille maximum de message acceptée par le serveur
`8BITMIME`	Possibilité d'envoyer le corps encodé sur 8 bits
`X???`	Extension SMTP locale

Négociation des extentions ESMTP :

```
EHLO hobbes.lip6.fr.
250-hobbes.lip6.fr Hello [62.62.169.227], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-EXPN
250-VERB
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-DELIVERBY
250 HELP
```


Evolution du format des entêtes

Caractères non ASCII dans les entêtes :

=?*charset*?*encode*?*encoded-text*?=

- *charset* : `us-ascii`, `iso-8859-`x, ...
- *encode* : le texte encodé doit rester en ASCII NVT
  - ✓ **Qoted-printable** (`Q`) pour les jeux de caractères latins :
    - ☞ caractères > 128 ➠ encodé sur 3 caractères (= et code hexa.)
    - ☞ caractère espace ➠ toujours =20
  - ✓ **Base64** (`B`) :
    - ☞ trois octets de texte (24 bits) ➠ encodée sur 4 car. ASCII
    - ☞ valeur sur 6 bits (0, 1, 2... 63) ➠ `ABC...YZab...yz01...9+/`
    - ☞ bourrage avec "=" si non aligné sur 4 caractères.
- *encoded-text* :
  - ✓ `=?iso-8859-2?Q?Igen,=20k=F6sz=F6n=F6m?=`
  - ✓ `=?iso-8859-1?B?QnJhdm8sIHZvdXMgYXZleiBy6XVzc2kgIQo=?=`

MIME : format des messages étendus

Multipurpose Internet Mail Extensions

Nouveaux entêtes MIME (RFC 2045 et RFC 2046)

- `Mime-Version:` **1.0**
- `Content-Type:` *type*/*sous-type*;*parametres*
  - ✓ **simples** : `text/plain; charset="ISO-8859-1"`
    - ☞ `text/html`, `image/jpeg`...
  - ✓ **structurés** : `multipart/mixed; Boundary=hjfdskjhfdshfdsk`
    - ☞ chaque partie du message débute par : `hjfdskjhfdshfdsk`
    - ☞ imbrication possible
- `Content-Disposition:` présentation du morceau (RFC 2183)
- `Content-Transfer-Encoding:` encodage indépendant d'ESMTP
  - ✓ 7 bits compatible avec les anciens MTA RFC 821
    - ☞ `7bit` (ASCII NVT)
    - ☞ `quoted-printable` (recommandé pour tout texte)
    - ☞ `base64` (recommandé pour les flux d'octets)
  - ✓ 8 bits si la commande `8BITMIME` est acceptée
    - ☞ `8bit` et `Binary` (lignes ou bloc de données sur 8 bits)

MIME : types et sous-types

Fichier `/etc/mime.types`

```
application/mac-binhex40
application/msword
application/octet-stream
application/postscript
application/vnd.hp-PCL
application/vnd.ms-excel
application/x-debian-package
application/x-doom
application/x-gnumeric
application/x-java-applet
application/x-javascript
application/x-msdos-program
application/x-tar

audio/basic
audio/midi
audio/mpeg
audio/x-wav

image/jpeg
image/png
image/tiff

message/delivery-status
message/external-body
message/http
message/partial
message/rfc822

multipart/alternative
multipart/digest
multipart/encrypted
multipart/mixed
multipart/parallel
multipart/signed

text/html
text/plain
text/richtext
text/rtf
text/xml
text/x-java
text/x-tex
text/x-vcard

video/mpeg
video/quicktime
video/x-msvideo
```


ESMTP : exemple de message MIME

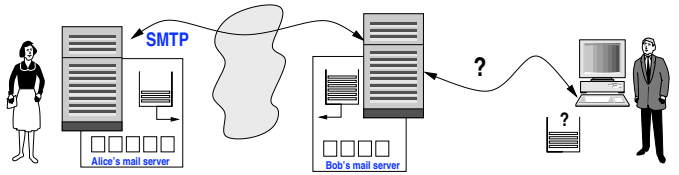
```
From: Olivier Fourmaux <olivier.fourmaux@lip6.fr>
Date: Wed, 20 Feb 2002 01:21:01 +0100
To: Toto <toto@free.fr>
Subject: Document no 3.02
Mime-Version: 1.0
Content-Type: multipart/mixed; boundary="/9DWx/yDrRhgMJTb"
Content-Disposition: inline
Content-Transfer-Encoding: 8bit
User-Agent: Mutt/1.2.5i

--/9DWx/yDrRhgMJTb
Content-Type: text/plain; charset=iso-8859-1
Content-Disposition: inline
Content-Transfer-Encoding: 8bit

Voici le document TOP SECRET que vous m'avez demandé...

--/9DWx/yDrRhgMJTb
Content-Type: application/pdf
Content-Disposition: attachment; filename="sujet-exam-RES.pdf"
Content-Transfer-Encoding: base64

JVBERi0xLjIKJcfsj6IKNSAwIG9iago8PC9MZW5ndGggNiAwIFIvRmlsdGVyIC9GbGF0ZURl
Y29kZT4+CnN0cmVhbQp4nO1dy7YdtGRd3684Mx6L07T63ZkBdghgXvYlJF1MHNsYm+sHhkCS...
```


Remise finale des messages

Machine accédant sporadiquement au réseau ?

➠ Messages stockés sur le dernier MTA (celui de l'ISP par exemple)

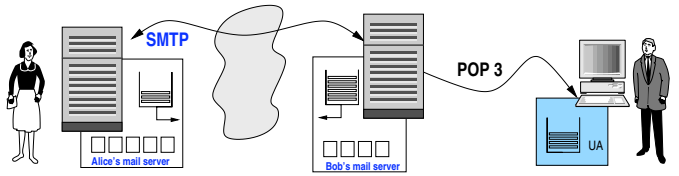
- plusieurs alternatives combinables :
  - ✓ accès direct au serveur (montage NFS ou SMB)
  - ✓ POP
  - ✓ IMAP
  - ✓ HTTP



POP 3

Post Office Protocol – Version 3 (RFC 1939)

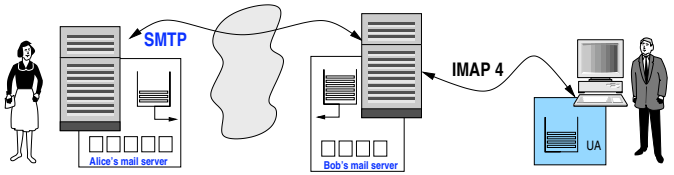
- simple
- connexion TCP sur le port 110
- trois phases :
  - ✓ autorisation (identification)
  - ✓ transaction (récupération et marquage)
  - ✓ mise à jour (suppression effective du serveur)



IMAP 4

Internet Mail Access Protocol – version 4 (RFC 2060)

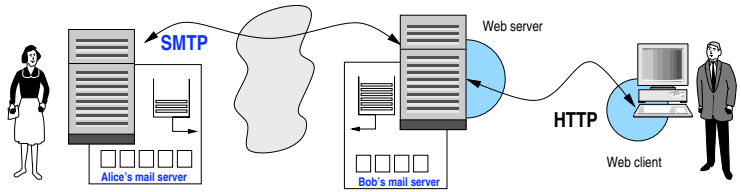
- complexe
- connexion TCP sur le port 143
- même fonctionnalité que POP avec :
  - ✓ accès par attribut (12^{eme} e-mail d'Alice)
  - ✓ récupération de partie de message (3^{eme} pièce jointe)
  - ✓ **synchronisation** de boites aux lettres



Web-mail

UA sur le serveur SMTP et interface Web

- comptes web spécifique : *Hotmail*, *Yahoo !*...
- autre moyen d'accès au serveur d'entreprise ou de l'ISP



Messagerie et sécurité

Les protocoles de base sont des passoires !

- échange textuel en clair (contrôle et données) : pas de confidentialité
- aucune authentification avec SNMP
- identifiant et mot de passe en clair avec POP et IMAP

Quelques solutions :

- PGP *(Pretty good privacy)* en environnement hostile :
  - ✓ authentification, intégrité et confidentialité (données signées et/ou cryptées)
  - ✓ **OpenPGP** (RFC 2440) : GPG *(Gnu Privacy Guard)*
- si confiance dans le site distant, sécurisation des connexions réseaux :
  - ✓ si le site distant est accessible via SSH
    - ☞ accès à distance sur le serveur via SSH (UA textuel : Unix)
    - ☞ **tunnels SSH**
  - ✓ si clients et **serveurs** avec SSL (ou TLS)
    - ☞ POP3S (RFC 2595) : port 995
    - ☞ IMAPS (RFC 2595) : port 993
    - ☞ HTTPS pour sécuriser le Web-Mail...

Plan

World wide web

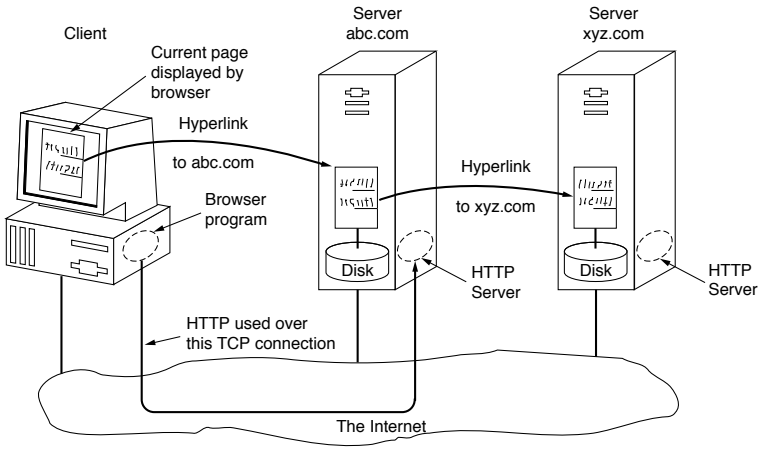
$\rightarrow 90'$: Internet = réseau académique

$90' \rightarrow$: **World Wide Web**

- système d'accès aux données convivial et intuitif (graphique)
- développé au CERN par Tim Berners-Lee à partir de 1990
- première "*killer app.*" grand public
  ✓
- client (*browser*) :
  ✓ NCSA Mosaic en 1993 (University of Illinois Urbana-Champagne)
    ☞ le WWW ne compte que 200 sites
    ☞ première intégration des dessins
    ☞ gain de popularité exponentiel !
  ✓ Netscape Navigator en 1994 (➠ **Mozilla** en 1998)
  ✓ Microsoft Internet Explorer en 1995 (début de la *browser wars*)
  ✓ et beaucoup d'autres (voir le site du W3C)
- serveur (*web server*) :
  ✓ NCSA `httpd` Web Server (➠ **Apache** en 1998)
  ✓ Microsoft IIS *(Internet Information Service)* en 1995

➠ un protocole : **HTTP**

HTTP : Principe



pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

HTTP : Terminologie

- une **page web** ou un **document** est composé d'objets
  ✓ fichiers texte au format HTML
  ✓ images GIF, JPEG...
  ✓ applets JAVA
  ✓ ...
- un document consiste généralement en un **fichier HTML de base** avec des références vers d'autres objets désignés par des URL
  ✓ **HTML** (*HyperText Markup Language*) est un langage à balises pour la description de documents contenant des hyper-liens identifiés par des URL
  ✓ une **URL** (*Uniform Resource Locator*) indique un protocole pour récupérer sur une machine un objets à travers le réseau
    ☞ `http://www.lip6.fr/info/linux.html`
    ☞ `ftp://ftp.lip6.fr/pub/linux/disrib/debian/ls-lR.txt`
    ☞ `file:/public/image/penguin.jpeg`
    ☞ `mailto:olivier.fourmaux@lip6.fr`

HTTP : Protocole

HyperText Transfer Protocol

- connexion TCP sur le port 80
- échanges définis :
  ✓ les **requêtes** de demande d'objets (client ⇒ serveur)
  ✓ les **transferts** d'objets demandés (serveur ⇒ client)
- versions HTTP :
  ✓ $\rightarrow 97$ **HTTP/1.0** (RFC1945)
    ☞ connexions **non persistantes**, une connexion créée par objet, charge et latence importantes (TCP *three-way handshake* et *slow–start*)
  ✓ $98 \rightarrow$ **HTTP/1.1** (RFC2616)
    ☞ compatibilité ascendante, connexions **persistantes**, possibilité de requêtes parallèles (**pipelining**)
- pas d'état dans le serveur (**stateless protocol**)

HTTP : Exemple

Brownser :

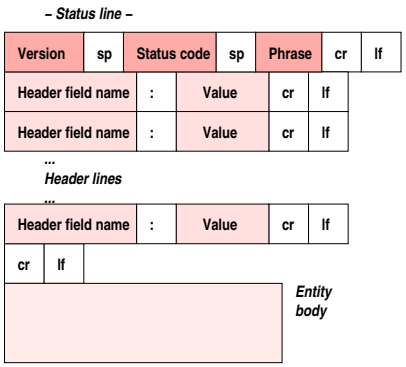
```
GET /index.html HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4 [en] (X11; I; Linux 0.99 i486)
Host: calvin.lip6.fr
Accept: image/gif, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: fr-FR, fr, en
Accept-Charset: iso-8859-1,*,utf-8
```

Web server :

```
HTTP/1.1 200 OK
Date: Tue, 24 Sep 2002 12:59:28 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU
Last-Modified: Sat, 29 Apr 2000 07:07:45 GMT
ETag: "1382c-ffe-390a8a41"
Accept-Ranges: bytes
Content-Length: 4094
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="tex...
   <META NAME="GENERATOR" CONTENT="Mozilla/4.05...
   <META NAME="Author" CONTENT="johnie@debian.o...
   <META NAME="Description" CONTENT="The initia...
   <TITLE>Welcome to Your New Home Page!</TITLE>
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0...
<BR>
<H1>Welcome to Your New Home in Cyberspace!</H1>
<BR>
<IMG SRC="icons/openlogo-25.jpg" ALT="Debian">
<IMG SRC="icons/apache_pb.gif" ALT="Apache"></P>

<P>This is a placeholder page installed by the
<A HREF="http://www.debian.org/">Debian</A>
release of the
<A HREF="http://www.apache.org/">Apache</A> Web
server package, because no home page was installed
on this host. You may want to replace this as soon
as possible with your own web pages, of course....

<BLOCKQUOTE>
This computer has installed the Debian GNU/Linux
operating system but has nothing to do with the
Debian GNU/Linux project.  If you want to report
something about this hosts behavour or domain,
please contact the ISPs involved directly,
<strong>not</strong> the Debian Project. <P>
</BLOCKQUOTE>
.............................
</HTML>
```


HTTP : Format réponse

Format général d'un message :



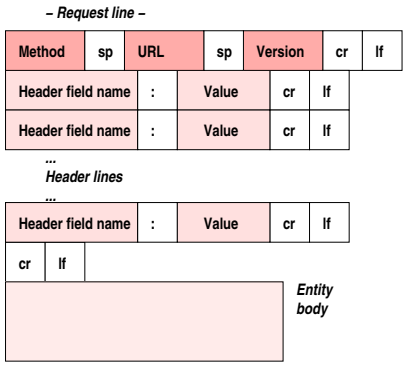
Exemple :

```
HTTP/1.1 200 OK
Date: Tue, 24 Sep 2002 12:59:28 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU
Last-Modified: Sat, 29 Apr 2000 07:07:45 GMT
Content-Length: 4094
...
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
.............................
</HTML>
```

- *status + description* :
  - ✓ 200 OK
  - ✓ 301 Move permanently
  - ✓ 400 Bad Request
  - ✓ 404 Not Found
  - ✓ 505 HTTP Version Not Supported
  - ✓ ...

HTTP : Format requête

Format général d'un message :



Exemple :

```
GET /index.html HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4 [en] (X11; I; Linux 0.99 i486)
Host: calvin.lip6.fr
Accept: image/gif, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: fr-FR, fr, en
Accept-Charset: iso-8859-1,*,utf-8
```

- **Method**
  - ✓ GET
  - ✓ POST (formulaires)
  - ✓ HEAD (test de pages)
- **Connection**
  - ✓ Close
  - ✓ Keep-Alive

HTTP : Identifier les utilisateurs (1)

Authentification

- requête du client sur une page avec procédure d'authentification
  - ✓ réponse du serveur page vide avec entête :
    - ☞ `401 Authorisation Required`
    - ☞ `WWW-Authenticate:` *détails_méthode_d'autorisation*
  - ✓ requête du client sur la même page avec entête :
    - ☞ `Authorization:` *nom_utilisateur mot_de_passe*
  - ✓ réponse du serveur :
    - ☞ si Ok ➠ la page demandée
    - ☞ sinon `401 Authorisation Required`...

HTTP : Identifier les utilisateurs (2)

Cookies (RFC 2109)

- identifiant associé à un utilisateur sur sa machine :
- le serveur indique un cookie avec l'entête :
  ✓ `Set-cookie:` *nombre_identifiant*
- le cookie est stocké chez le client qui, lorsqu'il demandera la même page sur le même serveur, l'intégrera grâce à l'entête :
  ✓ `Cookie:` *nombre_identifiant*

HTTP : GET conditionnel

1^{ere} requête HTTP :

```
GET /carte/france.jpg HTTP/1.1
Host: www.atlas.org
```

1^{ere} réponse HTTP :

```
HTTP/1.1 200 OK
Date: Mon, 2 Oct 2005 23:56:18
Server: Apache/1.3.9 (Unix)
Last-Modified: Sat, 29 Apr 2005 07:07:45
Content-Type: image/jpeg

Données.....................................
.....................................
.....................................
.....................................
.....................................
```

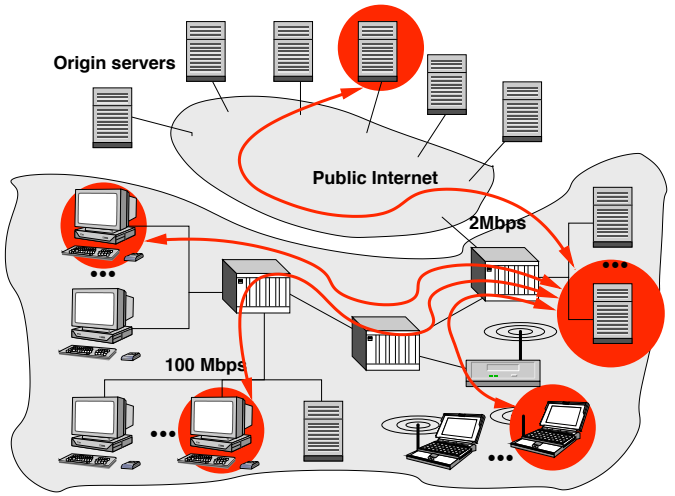
2^{eme} requête HTTP :

```
GET /carte/france.jpg HTTP/1.1
Host: www.atlas.org
If-modified-since: Sat, 29 Apr 2005 07:07:45
```

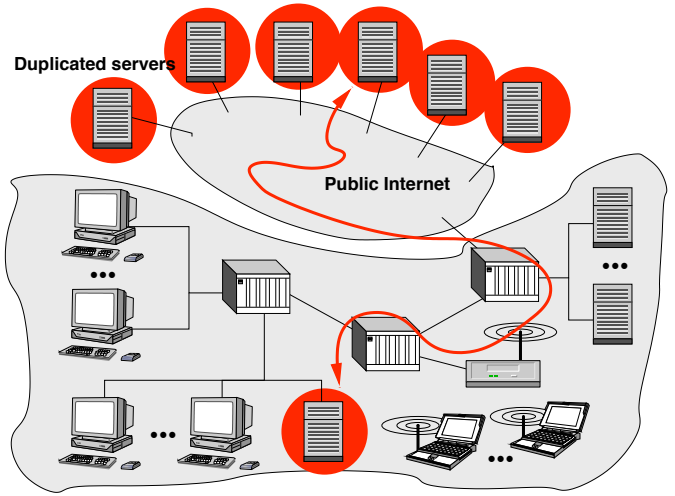
2^{eme} réponse HTTP :

```
HTTP/1.1 304 Not Modified
Date: Mon, 3 Oct 2005 00:06:43
Server: Apache/1.3.9 (Unix) Debian/GNU
```


HTTP : Cache et proxy



HTTP : CDN



Autour d'HTTP

Beaucoup de choses à étudier (bi-normalisation IETF et W3C)

Optimisation de l'accès aux ressources
- hiérarchie de caches
- répartition de charge
  - ✓ domaine des systèmes répartis ➠ U.E. **SRCS**

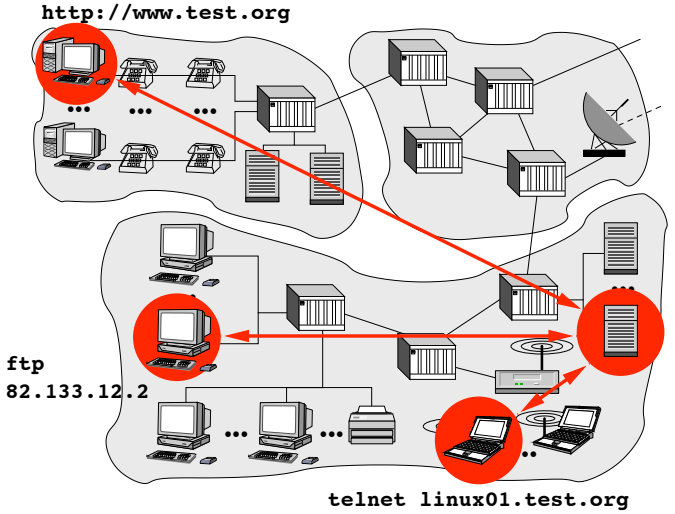
Contenu transféré
- génération automatique : PHP, ASP, Servlet...
  - ✓ programmation événementielle
- couplage aux bases d'information
  - ✓ domaine des bases de données et de la structuration de l'information type XML ➠ U.E. **BDWEB**

Sécurité
- HTTPS (RFC 2818) : utilise SSL sur le port 443 (ou TLS)
- Applets...

Protocole de transport générique
- XML, WML...
- encapsulation (firewall...)

Plan

Correspondance noms – adresses



Annuaire

Conversion des noms littéraux des hôtes de l'Internet en adresses numériques

- initialement
  - ✓ **un** fichier
  - ✓ espace de "nommage" à plat
  - ✓ gestion de manière centralisée par le *NIC*
- actuellement : **DNS**
  - ✓ base de données **distribuée**
  - ✓ espace de "nommage" **hiérarchique**
    - ☞ décorrélé de la topologie physique
  - ✓ système contrôlé par l'*InterNIC* et ses nombreux délégués
    - ☞ délégation hiérarchique (proche de celle du "nommage")
    - ☞ taille des délégations raisonables
  - ✓ protocole d'échange...

DNS : principe

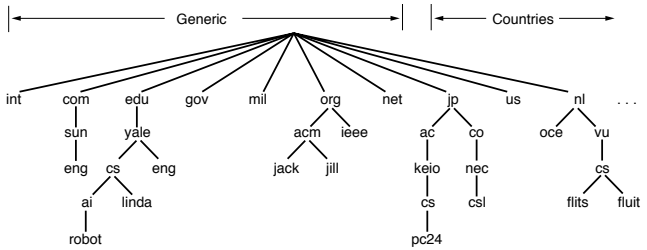
Domain Name System
Annuaire standard de l'Internet (RFC 1034 et RFC 1035)

- espace de "nommage" hiérarchique et système de délégation
- **serveurs de noms** (serveurs DNS)
  - ✓ composants physiques de la **hiérarchie** supportant la base distribuée
  - ✓ gèrent les requêtes DNS
  - ✓ transport sur **UDP** ou TCP, port **53**
  - ✓ les applications y accèdent à travers le **resolver** (UNIX) :
    - ☞ `gethostbyname` (3), `gethostbyaddr` (3)
- services :
  - ✓ *name resolving*
  - ✓ *host aliasing*
  - ✓ *mail server aliasing*
  - ✓ *load disribution...*
- exemple :
  - ✓ BIND (*Berkeley Internet Name Domain*)
    - ☞ `named` (UNIX)

DNS : Espace de "nommage"

Système de "nommage" hiérarchique

- structure arborescente (∼ système de fichier Unix)
- label d'un nœud : 63 caractères max (`A..Za..z-` insensible à la casse)
- **domain name** = liste des labels en parcourant l'arbre vers la racine (255 caractères max au total et "**.**" séparateur de label) :
  - ✓ absolu (**FQDN**) : `pc24.CS.keio.ac.jp.`
  - ✓ les noms relatifs sont gérés localement (hôte)



DNS : TLD

Top Level Domain

- ICANN (*Internet Corporation for Assigned Names and Numbers*)
  - ✓ partage du premier niveau et délégation à des *registrars*

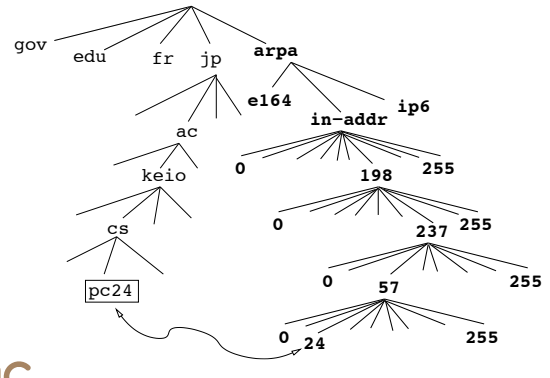
gTLD	intro.	description	operator
.aero	2001	Air-transport ind. *	SITA
.asia	2006	Asia-Pacific region *	Afilias
.biz	2001	Unrestricted	NeuLevel
.cat	2005	Catalan lingu. & cult.*	Asso. puntCAT
.com	1985	Unrestricted	VeriSign
.coop	2001	Cooperative *	DotCooperation
.edu	1985	(US) educ. inst. *	VeriSign
.gov	1985	US government *	US Admin.
.info	2001	Unrestricted	Afilias
.int	1988	Int. organisations	ICANN
.job	2005	Human resr. mngmnt*	Employ Media
.mil	1985	US military *	US DoD NIC
.mobi	2005	Mobile device use *	Mobi JV
.museum	2001	Museums *	MuseDoma
.name	2001	Individuals	VeriSign
.net	1985	Unrestricted	VeriSign
.org	1985	Unrestricted	Afilias
.pro	2001	Professionals	RegistryPro
.tel	2005	Internet Tel. serv.*	Telnic Limited
.travel	2005	Travel industry*	Tralliance Corp.

ccTLD ISO 3166	240 countries and external territories
.ac	Ascension Island
.af	Afghanistan
.aq	Antarctica (-60°S)
.eu	European Union
.fr	France
.gf	French Guiana
.gp	Guadeloupe
.mq	Martinique
.pf	French Polynesia + Clipperton
.pm	Saint-Pierre and Miquelon
.re	Réunion
.tf	TAAF
.ru	Russia (+.su)
.tv	Tuvalu
.uk	United Kingdom (+.gb)
.us	United States
.yu	Serbia + Montenegro
.za	South Africa
.zw	Zimbabwe

DNS : Domaine `.arpa`

Résolution : `pc24.cs.keio.ac.jp.` ➠ ?

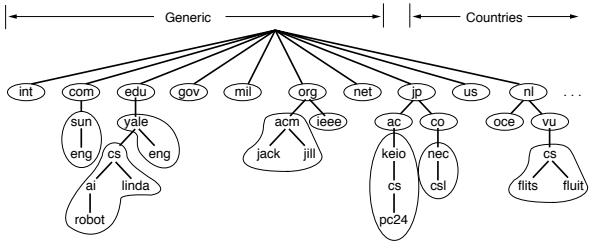
Résolution **inverse** : `24.57.237.198.in-addr.arpa.` ➠ ?



DNS : Zones

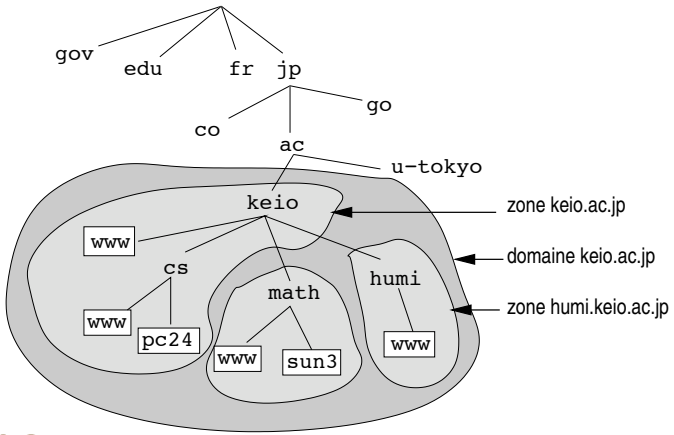
Sous-arbre de l'arbre DNS administré séparément

- (∼ partitions physiques d'un système de fichier Unix)
- délégation des noms de sous-domaines correspondants
  - ✓ exemple : `keio.ac.jp.`
- des **serveurs de noms** y sont associés



DNS : Zones

Ne pas confondre zone et domaine !

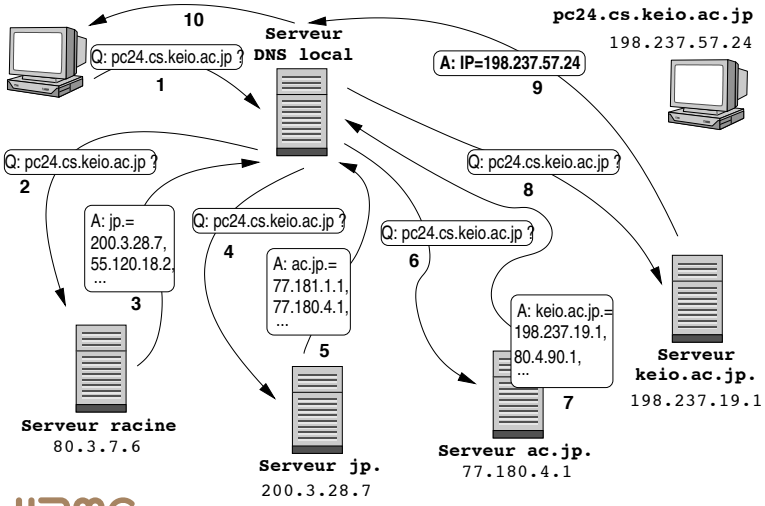


DNS : Serveurs de noms

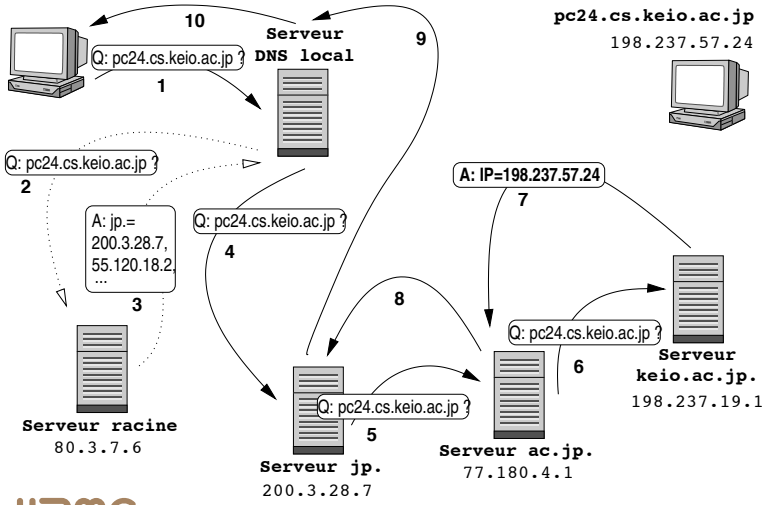
Différents types de serveurs de noms

- **serveurs de référence** d'une zone :
  - ✓ un **primaire** (*primary name server*)
    - ☞ informations de référence (*authoritative reccords*)
    - ☞ connaissance de ses descendants (délégations)
    - ☞ initialisation locale (disque)
  - ✓ un ou plusieurs **secondaire** (*secondary name server*)
    - ☞ redondance : complètement séparé du primaire
    - ☞ initialisation et m-à-j. à partir du primaire (**transfert de zone**)
  - ✓ physiquement indépendant de la zone
- **serveurs locaux** (accès au service)
  - ✓ résolution *top-down* (des TLD vers les sous-domaines)
  - ✓ connaissance des serveurs racines (*root name server*)
    - ☞ 1 primaire et 12 secondaires, haute disponibilité (anycast)
    - ☞ config. en dur (`ftp.rs.internic.net/domain/named.root`)
  - ✓ requêtes **récursives** ou **itératives**

DNS : Requête itérative



DNS : Requête récursive



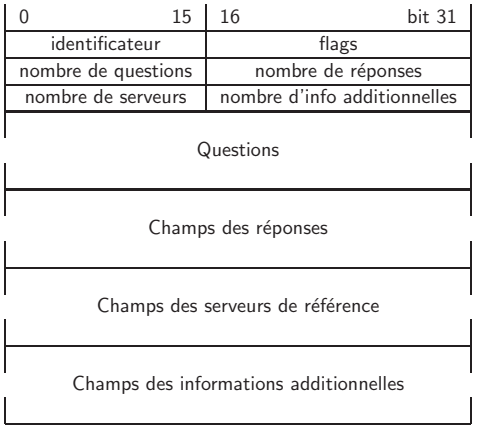
DNS : Performances

Capacité du système DNS à supporter la charge ?

- problèmes liés à la consultation systématique de la racine
  - ✓ ne tient pas compte de la localité des requêtes
    - ☞ serveur local généralement distinct du serveur de référence
  - ✓ charge sur les serveurs racines
    - ☞ combien de requêtes pour tout l'Internet ?
  - ✓ disponibilité des serveurs racines
    - ☞ passage obligé pour toute requête
- utilisation de **cache**
  - ✓ informations de seconde main (*non-authoritative reccords*)
  - ✓ réponses d'un serveur de référence inclue un délai de validité (TTL)
    - ☞ réponses pour les TLD sur les serveurs racines valide 48h
      - ➠ 100.000 requêtes par secondes (2005)

DNS : Format du message (1)

Pour les questions et les réponses :

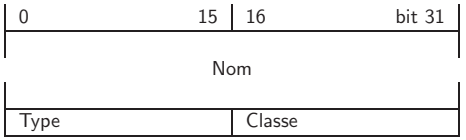


flags :

- **QR** (1 bit) : 0 = question, 1 = réponse
- **opcode** (4 bit) 0 = standard ...
- **AA** (1 bit) : 1 = réponse autoritaire
- **TC** (1 bit) : 1 = tronqué (datagramme UDP < 512o)
- **RD** (1 bit) : 1 = demande récursion (indiqué par le client)
- **RA** (1 bit) : 1 = récursion disponible (indiqué par le serveur)
- **réservé** (3 bits) : 000
- **rcode** (4 bits) : 0 = pas d'erreur... 3 = erreur de nom...

DNS : Format du message (2)

Une question :



- **Nom** : N octets, chaque nom de label est précédé par un octet indicant le nombre de caractères ASCII le suivant[1]. Terminé par 0x00[2].

  4, 'p', 'c', '2', '4', 2, 'c', 's', 4, 'k', 'e', 'i', 'o', 2, 'a', 'c', 2, 'j', 'p', 0

- **Type** (16 bits) :

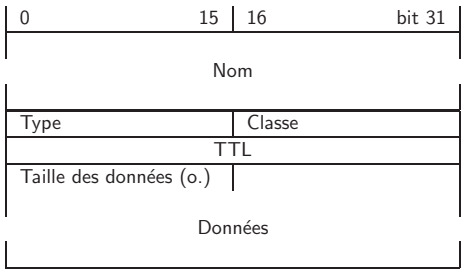
val	nom	description	val	nom	description
1	**A**	adresse IPv4	13	**HINFO**	info sur l'équipement
2	**NS**	nom de serveur	15	**MX**	serveur de messagerie
5	**CNAME**	alias	28	**AAAA**	adresse IPv6
6	**SOA**	zone DNS gérée		...	
12	**PTR**	pointeur de nom	255	*	tous les types (quest.)

- **Classe** (16 bits) : 1 = Internet

[1] Si > 0x3F alors échappement : 0xC0ZZ = renvoi au label situé à ZZ octets du début du message DNS
[2] Pas de bourrage si les N octets ne sont pas aligné sur 32 bits.

DNS : Format du message (3)

Un champ réponse :



- **Nom**, **Type**, **Classe** : idem
- **TTL** (32 bits) : validité en secondes
- **Taille des données** (16 bits) : en octets
- **Données** (N octets sans bourrage) :
  - ✓ Nom (chaine codée comme pour une question) NS, CNAME...
  - ✓ Adresses (valeur numérique) A sur 4 octets, AAAA sur 16 octets...

DNS : Annuaire inversé

Conversion des adresses numériques en noms littéraux

- requêtes de type **pointeur de nom** (PTR)
  - ✓ adresse IPv4
    - ☞ `198.237.57.24`
  - ✓ conversion dans le domaine `in-addr.arpa`
    - ☞ `24.57.237.198.in-addr.arpa`
      - ➠ souvent utilisé pour vérifier les droits d'accès

DNS : Obtention d'une délégation

Pour être référence pour un sous domaine officiel

- serveur conforme à la norme DNS
- information de référence de la zone
  - ✓ réplication dans au moins un serveur secondaire
- si sous délégations :
  - ✓ connaissance des serveurs descendants
- si gestion des adresses IP correspondantes :
  - ✓ information de référence des pointeurs de nom

DNS : Modification dynamique

Dynamique DNS (RFC 2136)

- pour fonctionner avec l'auto-configuration des hôtes (DNS local) :
  - ✓ *update*
  - ✓ *notification*
- problèmes de sécurité...

Service DNS dynamique (prestataire externe)

- pour fonctionner avec une adresse dynamique (accès résidentiels) :
  - ✓ serveur : `dyndns.org`, `no-ip.org`...
  - ✓ client spécifique indiquant le changement d'adresse *(host/setupbox)*
  - ✓ délégation virtuelle (sous domaine de 3ème niveau)
    - ☞ `toto123.myftp.biz`
    - ☞ `toto123.blogsite.org`
    - ☞ `toto123.homelinux.org`
    - ☞ `toto123.dyn-o-saur.com`
    - ☞ `toto123.endofinternet.net...`

DNS : Sécurité

Pas de sécurité dans le protocole de base (RFC 3833)
- interception / modification de message DNS
- faux messages *(DNS cache poisoning)*
- déni de service...

DNSSEC (RFC 4033 à 4035 + RFC 4310 + RFC 4641)
- extension du système DNS permettant :
  - ✓ authentification de l'origine des données
  - ✓ authentification du déni d'existence
  - ✓ intégrité des données
- obligatoire pour sécuriser les *DNS update*
  - ✓ attention aux extensions propriétaires...

DNS : Exemple

```
Unix> dig www.math.keio.ac.jp

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11895
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 4

;; QUESTION SECTION:
;www.math.keio.ac.jp.           IN      A

;; ANSWER SECTION:
www.math.keio.ac.jp.    3600    IN      CNAME   sun3.math.keio.ac.jp.
sun3.math.keio.ac.jp.   3600    IN      A       131.113.70.3

;; AUTHORITY SECTION:
math.keio.ac.jp.        3600    IN      NS      relay.math.keio.ac.jp.
math.keio.ac.jp.        3600    IN      NS      ns.st.keio.ac.jp.
math.keio.ac.jp.        3600    IN      NS      ns0.sfc.keio.ac.jp.

;; ADDITIONAL SECTION:
relay.math.keio.ac.jp.  3600    IN      A       131.113.70.1
ns.st.keio.ac.jp.       127     IN      A       131.113.1.8
ns0.sfc.keio.ac.jp.     1199    IN      AAAA    3ffe:501:1085:8001::121
ns0.sfc.keio.ac.jp.     2358    IN      A       133.27.4.121

;; Query time: 577 msec MSG SIZE  rcvd: 206
```


Plan

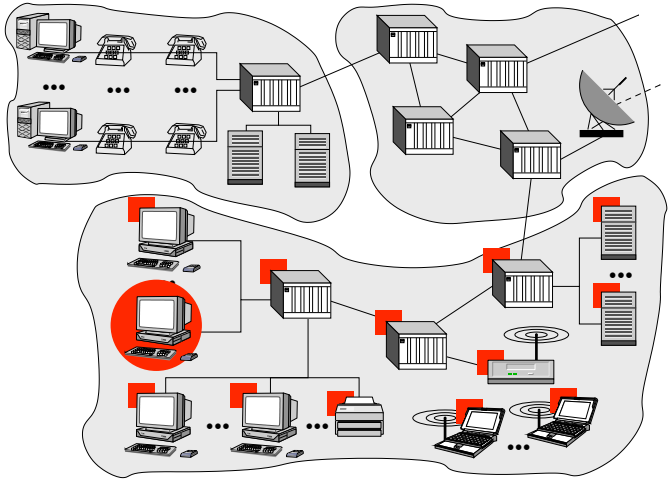
Administration de réseau

Développement du réseau (nombreux équipements et machines à gérer)

Besoins :
- surveillance du réseau
  - ✓ détection de pannes
  - ✓ mesure de performance
- intervention sur le matériel
  - ✓ activation (interface...)
  - ✓ configuration (table de routage...)
- poste de contrôle centralisé

Contraintes :
- matériels hétérogènes
  - ✓ routeurs, hubs, switchs...
  - ✓ ordinateurs, imprimantes, sondes...
- constructeurs multiples
- localisation géographique distante

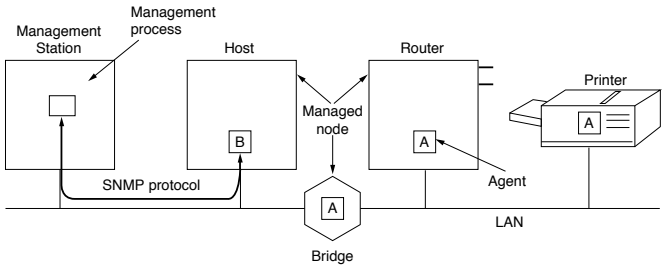
Equipements administrables



Administration TCP/IP

Comment gérer les machines en environnement TCP/IP ?

- instrumentation des équipements (**agents**)
- logiciels de supervision (HP Openview, Cisco Works, Nagios...)
- protocole de gestion ➠ SNMP



pictures from Tanenbaum A. S. *Computer Networks 3rd edition*

SNMP : principe

Informations réseau stockées dans deux types de bases :

- **bases agents** (dans les équipements) : Les valeurs sont directement couplées avec les registres internes
- **base centralisée** (plateforme de supervision) : dernières valeurs transmises et historique (statistiques)

Standardisation (pour échange en milieu hétérogène)

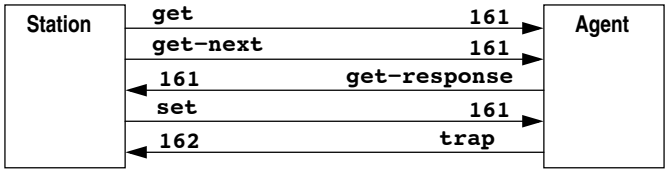
- désignation et type d'information définis par des **MIB**
- structures communes et nomenclature définies dans la **SMI**
- représentation des données en **ASN.1**
- protocole **SNMP** entre la station et les agents permettant :
  - ✓ **lecture/écriture** de variables sur des éléments gérés
  - ✓ **alarmes** non sollicitées
  - ✓ **parcours** de listes de variables dans les éléments gérés

➠ **vision agrégée globale**

SNMP : Commandes

La richesse est dans la MIB !

- seulement 5 commandes **simples**
- utilisation sur **UDP** port **161** et **162**



SNMP : Format des messages

version	communauté	type PDU	ident req.	erreur status	erreur index	nom	valeur	nom	valeur	...

- `version` : version SNMP - 1 (0 ∼ SNMPv1)
- `communauté` : chaine de caractères autorisant l'accès
  ✓ généralement "`public`"
- `type PDU` : 0 (`get`), 1 (`get-next`), 2 (`set`), 3 (`get-response`)
  ✓ le message de type 4 (`trap`) sera présenté dans la suite...
- `ident. req.` : permet de faire correspondre requêtes et réponses
- `erreur status` et `erreur index` : indique le type d'erreur concernant la variable référencée par l'indexage (0 ∼ pas d'erreur)
- `nom` et `valeur` : variables transportées

Les tailles des champs ne sont pas précisées car la structure du message est décrite en ASN.1 avec encodage BER.

SNMP : SMI

Structure for Management Information

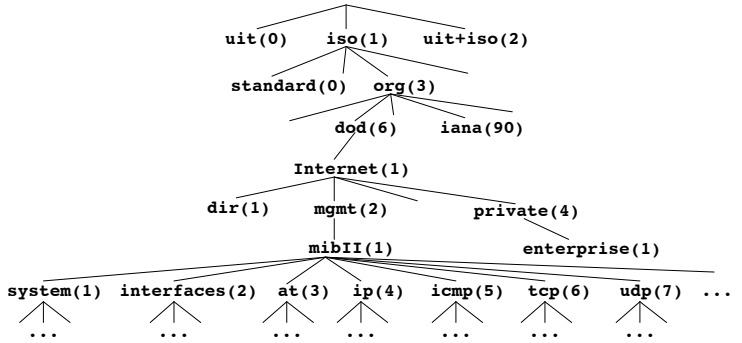
- les informations respectent les types de la SMIv1 (RFC 1155 et 1212)

`NULL`	pas de valeur
`INTEGER`	entier signé non limité
`Counter`	entier positif (0 à $2^{32}-1$) croissant et bouclant
`Gauge`	entier positif (0 à $2^{32}-1$) borné
`TimeTicks`	durée en centième de secondes
`OCTET STRING`	chaine d'octets non limitée
`DisplayString`	chaine codée en NVT de 255 caractères max.
`IpAddress`	chaine de 4 octets
`PhyAddress`	chaine de 6 octeys
`OBJECT IDENTIFIER`	identifiant numérique...
`SEQUENCE`	structure de différents éléments nommés
`SEQUENCE OF`	vecteur d'éléments identiques

OID

Object IDentifier

- arbre de "nommage" (référencement **unique** d'un objet)
  ✓ les objets de l'Internet commencent par 1.3.6.1.



SNMP : MIB

Management Information Base

- les goupes d'objets définis dans la MIB II (RFC 1213) :

```
1.3.6.1.2.1.1    system
1.3.6.1.2.1.2    interfaces
1.3.6.1.2.1.3    at
1.3.6.1.2.1.4    ip
1.3.6.1.2.1.5    icmp
1.3.6.1.2.1.6    tcp
1.3.6.1.2.1.7    udp
1.3.6.1.2.1.8    egp
1.3.6.1.2.1.10   transmission
1.3.6.1.2.1.11   snmp
```

- d'autres groupes, ou sous-groupes sont définis (autres RFC) :

```
1.3.6.1.2.1.17      bridge
1.3.6.1.2.1.43      printer
1.3.6.1.2.1.10.15   fddi
```

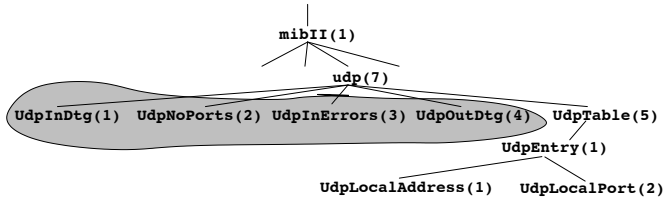
Ces groupes contiennent des variables **simples** ou **tables**

MIB : Variable simple

Dans le groupe UDP, 4 variables simples :

- la MIB II fait correspondre des types SMI

udpInDatagrams	Counter	ro	nb datagrammes délivrés aux applications
udpNoPorts	Counter	ro	nb datagrammes sans application en attente
udpInErrors	Counter	ro	nb datagrammes non délivrables
udpOutDatagrams	Counter	ro	nb datagrammes émis

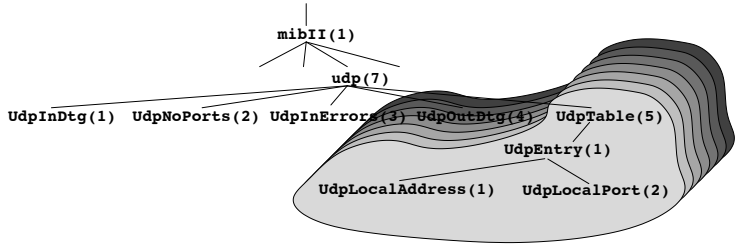


MIB : Variable table

Dans le groupe UDP, 1 variable table :

- udpTable indique les ports scrutés sur l'équipement
- udpTable est un **vecteur** de structures udpEntry

udpLocalAddress	IpAddress	ro	adresse IP locale
udpLocalPorts	[0..65535]	ro	port correspondant



- l'**index** dans la table est ici udpLocalAddress.udpLocalPorts
  - ✓ l'index est précisé à la conception de la MIB

SNMP : Référencement des variables

Référencement des variables :

- simples : ajout de ".0" à la fin
- tables : ajout des valeurs des champs index
  - ✓ parcours des OID de la table dans l'ordre **lexicographique**

nom abrégé	OID	valeur
udpInDatagrams.0	1.3.6.1.2.1.7.1.0	17625
udpLocalAddress.0.0.0.0.53	1.3.6.1.2.1.7.5.1.1.0.0.0.0.53	0.0.0.0
udpLocalAddress.0.0.0.0.161	1.3.6.1.2.1.7.5.1.1.0.0.0.0.161	0.0.0.0
udpLocalPort.0.0.0.0.53	1.3.6.1.2.1.7.5.1.2.0.0.0.0.53	53
udpLocalPort.0.0.0.0.161	1.3.6.1.2.1.7.5.1.2.0.0.0.0.161	161

- le référencement permet de spécifier les objets dans les messages UDP
  - ✓ seuls les OID et les valeurs sont transportées

SNMP : Commande `get-next`

Opérateur de parcours basé sur l'ordre **lexicographique** des OIDS :

- renvoie la prochaine référence terminale
  - ✓ `get-next udp ➠ udpInDatagrams.0 = 17625`
- permet le parcours des variables...
  - ✓ `get-next udpInDatagrams.0 ➠ udpNoPorts.0 = 0`
- ... et des tables
  - ✓
    ```
    get-next udpTable
    ➠ udpLocalAddress.0.0.0.0.53 = 0.0.0.0
    get-next udpLocalAddress.0.0.0.0.53
    ➠ udpLocalAddress.0.0.0.0.161 = 0.0.0.0
    get-next udpLocalAddress.0.0.0.0.161
    ➠ udpLocalPort.0.0.0.0.53 = 53 ...
    ```
- fin du tableau lors du changement de nom :
  - ✓
    ```
    get-next udpLocalPort.0.0.0.0.161
    ➠ snmpInPkts.0 = 12
    ```

SNMP : Trap

Envoi d'un message SNMP de l'agent vers l'administrateur sur le **port 162**

version	communauté	type = 4	entreprise	adr. agent	type trap	code entr.	estamp. temp.	nom	valeur	...

- `entreprise` : identificateur du créateur de l'agent
  - ✓ OID débutant par `1.3.6.1.4.1.`
- `adr. agent` : adresse IP de l'agent
- `type trap` :

0	`coldStart`	agent initialisé
1	`warmStart`	agent réinitialisé
2	`linkDown`	interface désactivée
3	`linkUp`	interface activée
	...	
6	`entr. specific`	voir le champ code entr.

- `code entr.` : sous-code du trap spécifique à l'entreprise
- `estamp. temp.` : valeur indiquant le nombre de centièmes de secondes depuis le démarrage de l'agent

Syntaxe abstraite ASN.1

Abstract Syntax Notation One

- couche 6 de l'OSI (définie par l'UIT, recommandation X.680)
- propriétés :
  - ✓ représentation universelle d'informations
  - ✓ type associé aux données
  - ✓ désignation par un identificateur unique (OID)
  - ✓ notation de type BNF
- description des informations échangées par SNMP :

```
RFC1157-SNMP DEFINITIONS ::= BEGIN
     Message ::= SEQUENCE {
                     version      INTEGER {version-1(0)},
                     community    OCTET STRING,
                     data         ANY
                 }
     PDUs ::= CHOICE {
                 get-request      GetRequest-PDU,
                 get-next-request GetNextRequest-PDU,
                 get-response     GetResponse-PDU,
                 set-request      SetRequest-PDU,
                 trap             Trap-PDU
             }...
```


ASN.1 : PDU

Message get écrit en ASN.1 :

```
getRequest-PDU ::=  [0]
                    IMPLICIT SEQUENCE {
                         request-id   INTEGER,
                         error-status INTEGER {
                                              noError(0), tooBig(1),
                                              noSuchName(2), badValue(3),
                                              readOnly(4) genErr(5),    -- always 0
                                      }
                         error-index  INTEGER,                          -- always 0
                         variable-bindings
                                      SEQUENCE OF
                                            SEQUENCE {
                                                name   ObjectName,
                                                value  ObjectSyntax
                                            }
                    }
```


SNMP : Encodage BER

Encodage **TLV** (Type, Longueur, Valeur)

- types (1 octet) : les 2 bits de poids fort déterminent la catégorie
  - ✓ `UNIVERSAL (00)`

0x02	`INTERGER`
0x04	`OCTET STRING`
0x05	`NULL`
0x06	`OBJECT IDENTIFIER`
0x30	`SEQUENCE`

  - ✓ `APPLICATION (01)`

0x40	`IpAddress`
0x41	`Counter`
0x42	`Gauge`
0x43	`TimeTicks`

  - ✓ `CONTEXT (10)`
  - ✓ `PRIVATE (11)`
- longueur des données (1 octet si $< 0x80$, sinon voir la norme X.208)
  - ✓ longueur 49 ➠ 0x31, longueur 242 ➠ 0x8200F2...
- données (valeur)
  - ✓ les OID (avec les valeurs entières successives A.B.C.D...) sont codés sur des octets avec les 2 premiers agrégés : A*40+B, C, D...

SNMP : Exemple

```
0020                                30 82 00 f2 02 01   J...D... ..0.....

0030  00 04 06 70 75 62 6c 69  63 a2 82 00 e3 02 01 01   ...publi c.......

0040  02 01 00 02 01 00 30 82  00 d6 30 82 00 0d 06 08   ......0. ..0.....

0050  2b 06 01 02 01 02 01 00  02 01 03 30 82 00 0f 06   +....... ...0....

0060  0a 2b 06 01 02 01 02 02  01 08 01 02 01 01 30 82   .+...... ......0.

0070  00 0f 06 0a 2b 06 01 02  01 02 02 01 08 02 02 01   ....+... ........

0080  02 .. ..

0100                                .. .. 30 82 00 10   ........ C..,0...

0110  06 0a 2b 06 01 02 01 02  02 01 09 01 43 02 01 2c   ..+..... ....C...
```


MIB RMON

Remote MONitoring (RFC 2819 - STD 59)

Sonde pour obtenir des **statistiques** sur un réseau administré

- 9 groupes :
  - ✓ statistiques sur Ethernet (table de 21 attributs)
  - ✓ équipements du réseau (adresses observées...)
  - ✓ matrice de statistiques (entre deux stations)
  - ✓ capture de trames
  - ✓ ...
- nombreuses extensions
  - ✓ identification de protocoles pour RMON (RFC 2895, 2896)
  - ✓ RMON pour réseaux commutés (SMON : RFC 2613)
  - ✓ gestion des interface pour RMON (IFTOPN : RFC 3144)
  - ✓ RMON pour les services différenciés (DSMON : RFC 3287)
  - ✓ ...

MIB Imprimante

Printer MIB (**RFC 1759** - RFC 3805)

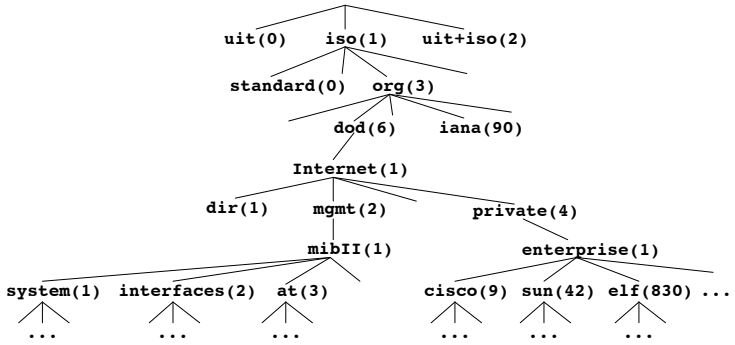
- 274 Objets (228 OID dont 16 tables)
  - ✓ 20 groupes :
    - ☞ groupe général
    - ☞ groupe des entrées
    - ☞ groupe des sorties
    - ☞ groupe des dimensions de sortie
    - ☞ groupe de la couverture
    - ☞ groupe des fournitures
    - ☞ groupe des colorants
    - ☞ ...

Autres MIB IETF (+100)

RFC1230 : IEEE 802.4 Token Bus MIB
RFC1381 : MIB Extension for X.25 LAPB
RFC1559 : DECnet Phase IV MIB Extensions
RFC1593 : SNA APPN Node MIB
RFC1611 : DNS Server MIB Extensions
RFC1612 : DNS Resolver MIB Extensions
RFC1696 : Modem MIB
RFC1697 : Relational DB Mngmnt System MIB
RFC1724 : RIP Version 2 MIB
RFC1748 : IEEE 802.5 MIB
RFC2020 : IEEE 802.12 Interface MIB
RFC2320 : Classical IP and ARP Over ATM MIB
RFC2564 : Application Management MIB
RFC1792 : TCP/IPX Connection MIB
RFC2605 : Directory Server Monitoring MIB
RFC2707 : Job Monitoring MIB
RFC2720 : Traffic Flow Measurement : Meter MIB
RFC2788 : Network Services Monitoring MIB
RFC2789 : Mail Monitoring MIB
RFC2790 : Host Resources MIB
RFC2863 : The Interfaces Group MIB
RFC2922 : Physical Topology MIB
RFC2932 : IPv4 Multicast Routing MIB
RFC2933 : IGMP MIB
...

...
RFC4087 : IP Tunnel MIB
RFC4150 : Transport Performance Metrics MIB
RFC4268 : Entity State MIB
RFC4292 : IP Forwarding Table MIB
RFC4323 : DOCSIS-QoS MIB
RFC4438 : Fibre-Channel Name Server MIB
RFC4439 : Fabric Address Manager MIB
RFC4624 : Multicast Source Discovery Protocol MIB
RFC4625 : Fibre Channel Routing Information MIB
RFC4626 : MIB Fabric Shortest Path First Protocol
RFC4631 : Link Management Protocol MIB
RFC4668 : RADIUS Authent. Client MIB for IPv6
RFC4669 : RADIUS Authent. Server MIB for IPv6
RFC4670 : RADIUS Accounting Client MIB for IPv6
RFC4671 : RADIUS Accounting Server MIB for IPv6
RFC4672 : RADIUS Dynamic Authoriz. Client MIB
RFC4673 : RADIUS Dynamic Authoriz. Server MIB
RFC4711 : Real-time Application QoS Monit. MIB
RFC4747 : The Virtual Fabrics MIB
RFC4807 : IPsec Security Policy DB Conf. MIB
RFC4898 : TCP Extended Statistics MIB
RFC4935 : Fibre Channel Fabric Conf. Server MIB
RFC4936 : Fibre Channel Zone Server MIB
RFC4983 : Fibre Channel RSCN MIB

MIB constructeur



SNMP : Versions

Plusieurs versions ont été standardisées :

- **SNMPv1** définie dans le RFC 1157 (1990) simple et non sécurisée
  ➠ encore très utilisée
- **SNMPv2** définie dans les RFC 1901 à 1908 avec extensions (requêtes `get-bulk` et `inform`, MIB SNMPv2 et SNMPv2-M2M) et sécurisation mais pas de consensus des industriels
  - ✓ **SNMPv2c** réduite aux nouvelles fonctionnalités mais sans la sécurité *(Community-Based)*
  - ✓ **SNMPv2u** nouveau mécanisme de sécurité simplifié *(User-Based)*
- **SNMPv3** définie dans les RFC 3410 à 3418, réintègre la sécurité
  - ✓ seule la v3 est un standard IETF (STD-62)
  - ✓ Utilisation de multi-version : RFC 3584

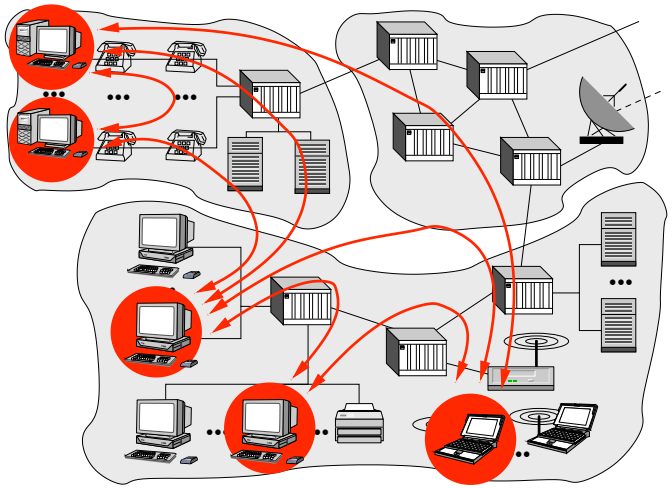
SNMP : limitations

- la mesure ne doit pas perturber le réseau
- latence
- MIB propriétaires
- sécurité
  - ✓ écoute sur le réseau *(packet sniffing)* pour connaitre la communauté
  - ✓ usurpation d'identité *(IP spoofing)* facilité par UDP

➠ améliorations avec **SNMPv3**

Plan

Peer-to-peer



Peer-to-peer

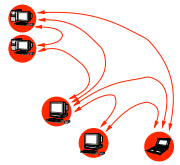
Applications *peer-to-peer*

- **partage de fichiers**
  - ✓ Napster, eDonkey, BitTorrent...
  - ✓ FastTrack (KaZaA, Grokster et Imesh), Gnutella2...
  - ✓ Gnutella...
- **stockage anonyme**
  - ✓ Freenet, Entropy...
- **distribution de flux audio/vidéo**
  - ✓ VoD : Peercast, Joost...
  - ✓ P2PTV : Coolstreaming, TVants, PPLive...
- autres services réseau "remontés" au niveau applicatif
  - ✓ protocoles de routage IP
  - ✓ système DNS
  - ✓ réseaux ad-hoc
  - ✓ communications multipoints
  - ✓ ...

P2P : Questions

Principes fondamentaux

- éléments de base générique (ni client, ni serveur)
- agrégation des ressources réseaux/processeur/stockage
- protocoles de niveau applicatif



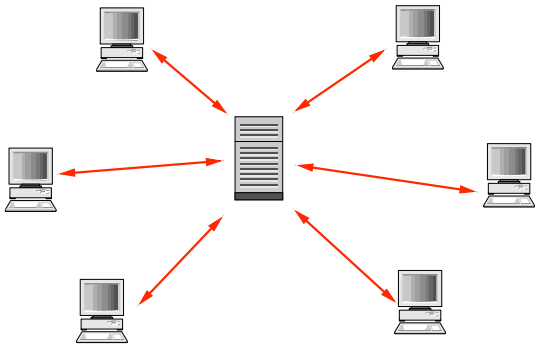
Standards ?

Ne peut-on pas tout faire en client/serveur ?

- est-ce juste du "réseau au niveau applicatif" ?
- quels nouveaux types de services ? d'applications ?
- quels sont les nouveaux challenges techniques ?

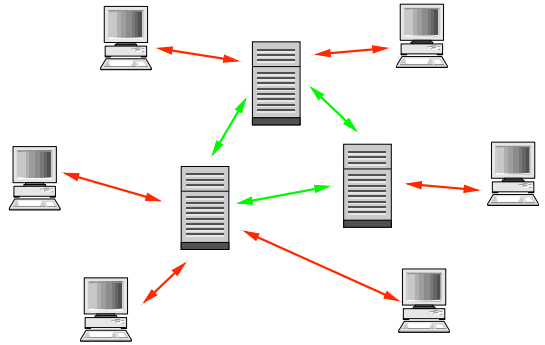
P2P : Architectures (1)

Client/serveur centalisé classique



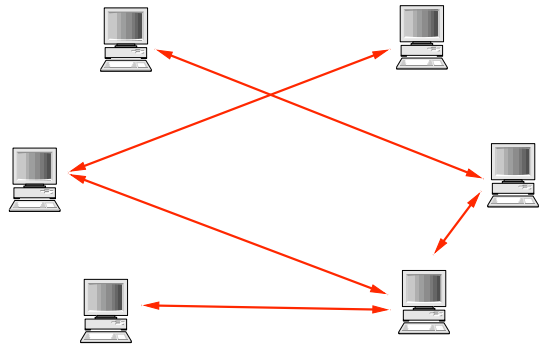
P2P : Architectures (2)

Client/serveur avec réplication des serveurs



P2P : Architectures (3)

Peer-to-peer classique



P2P : Comparaison Client/serveur

RPC/RMI
- synchrones
- asymétriques
- orientés langage
- identification
- authentification

```
Client_call(args)

Server_main_loop()
  while (true)
    await(call)
    switch(call.procid)
      case 0: call.ret=proc0(call.arg)
      case 1: call.ret=proc1(call.arg)
      ...
      default: exception
```

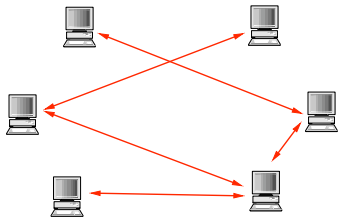
Messages P2P
- asynchrones
- symétriques
- orientés service
- anonymat
- haute disponibilité

```
Peer_main_loop()
  while (true)
    await(event)
    switch(event.type)
      case timer_expire:
            do_some_P2P_work()
            randomize_timers()
      case inbound_mesg:
            handle_mesg()
```


P2P : Fonctionnalités

Caractéristiques des systèmes *peer-to-peer*

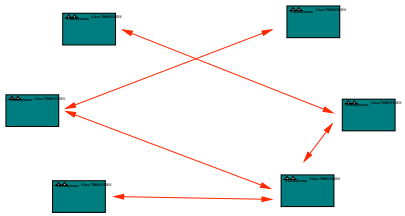
- pas de rôle distinct
  - ✓ évite les points de congestion ou les nœuds défaillants
  - ✓ besoin d'algorithmes distribués
    - ☞ découverte de service (nommage, adressage, calcul de métriques)
    - ☞ maintien d'un état du voisinage
    - ☞ routage au niveau applicatif (lié au contenu)
    - ☞ robustesse, gestion de perte de liens ou de nœuds
    - ☞ ...



P2P : Applications existantes

Le *peer-to-peer* n'est pas nouveau :

- Routeurs IP
  - ✓ découverte de la topologie
  - ✓ maintien de l'état du voisinage
  - ✓ autonome et tolérance aux fautes
  - ✓ algorithme distribué de routage
  - ✓ ...



Plan

Napster

Programme de partage de fichiers MP3

- pas le premier, mais le plus connu.
  - ✓ très informatif sur l'intérêt du *peer-to-peer*...
  - ✓ ... sur les problèmes techniques, légaux, politiques, économiques
- une technologie de rupture ?
  - ✓ historique
    - ☞ fin 98 : Shawn Fanning (19 ans) débute le developpement
    - ☞ 05/99 : création de *Napster Online Music Service*
    - ☞ 06/99 : premiers tests du logiciel
    - ☞ 12/99 : premières poursuites juridiques (Metallica, RIAA...)
    - ☞ mi 00 : plus de **60M** d'utilisateurs
      importante part du trafic universitaire (30% à 50%)
    - ☞ 02/01 : jugement par la Cour d'Appel des US
    - ☞ mi 01 : 160K utilisateurs...

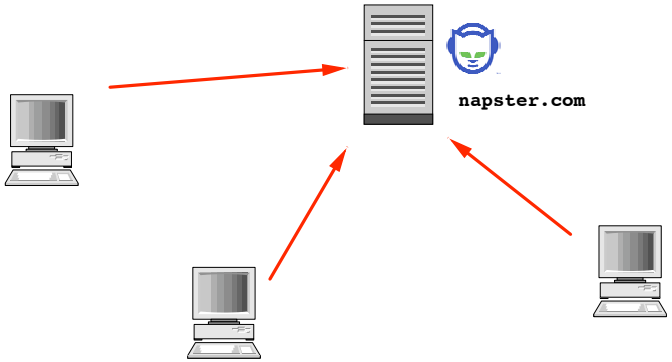
Napster : Principe

Approche mixte :

- recherche client/serveur avec liste centralisée
- échange direct des données recherchées entre pairs
- connexions point à point TCP (port 7777 ou 8888)
- 4 étapes :
  - ✓ Connexion au serveur Napster
  - ✓ Envoi de sa liste de fichiers au serveur (*push*)
  - ✓ Envoi des mots recherchés et récupération d'une liste de pairs
  - ✓ Selection du meilleur pair (*pings*)

Napster : *upload*

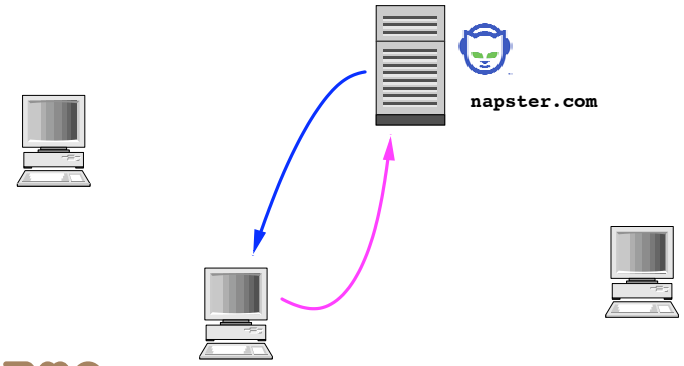
Les utilisateurs chargent la liste des fichiers à partager



Napster : *search*

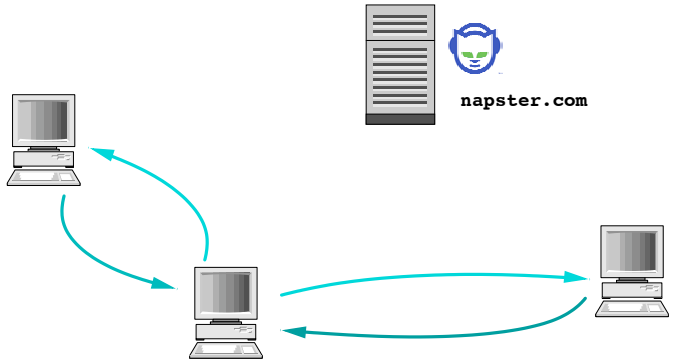
Un utilisateur émet une requête de recherche

Le serveur indique les localisations potentielles



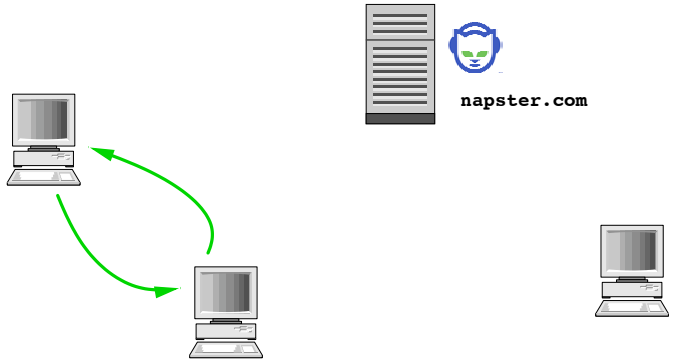
Napster : *pings*

Ping des pairs potentiels (recherche du meilleur débit de transfert)



Napster : *upload*

L'utilisateur récupère directement le fichier chez le pair choisi



Napster : Remarques

- serveur **centralisé**
  - ✓ un point de défaillance
  - ✓ risque de congestion
  - ✓ partage de charge en utilisant la rotation DNS
  - ✓ controlé par une entreprise
- absence de **sécurité**
  - ✓ mot de passe en clair
  - ✓ pas d'authentification
  - ✓ pas d'anonymat
  - ✓ code propriétaire
  - ✓ téléchargement de mises-à-jour
- évolution :
  - ✓ **OpenNap** :
    - ☞ *open source*
    - ☞ communications entre serveurs
    - ☞ tous types de données

Plan

Gnutella

Partage de fichiers complètement distribué

- corrige les défauts majeurs de Napster :
  - ✓ *Open source*
  - ✓ pas de serveurs - pas d'index global
  - ✓ connaissance locale seulement
- mais toujours les mêmes problèmes juridiques, économiques...
  - ✓ pas de responsable direct du service
  - ✓ absence d'anonymat
    - ☞ le RIAA attaque directement des utilisateurs !

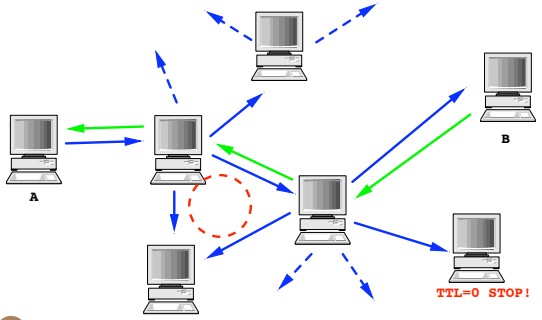
Gnutella

Peer-to-peer Networking
- connexion exclusive entre les applications des pairs
- problème :
  - ✓ recherche de fichiers **décentralisée**
- chaque application :
  - ✓ stocke une sélection de fichiers
  - ✓ oriente les requêtes de recherche (*route query*) de et vers ses pairs
  - ✓ répond aux demandes de transfert de fichiers
- historique
  - ✓ 03/00 abandon du projet *freelance* au bout de qqs jours après son lancement par AOL (Nullsoft)
  - ✓ trop tard : déjà plus de 20K utilisateurs...

Gnutella : Principe

Recherche par **inondation** (*flooding*)

- si je n'ai pas le fichier recherché :
  - ✓ je demande à 7 pairs s'ils ont ce fichier
- s'ils ne l'ont pas, ils contactent à leur tour 7 pairs voisins
  - ✓ recherche récursive limitée à N sauts
- détection de boucle par mémorisation temporaire des requêtes
  - ✓ les messages peuvent être reçus deux fois



Gnutella : Messages

Structure du message de contrôle Gnutella :

Gnode ID (16 octets)	**Type** (1 octet)	**TTL** (1 octet)	**Sauts** (1 octet)	**Longueur** (4 octets)	**Données...**

Gnode ID : identification du nœud dans le réseau gnutella

Type : action à réaliser

- `Ping` (recherche de pair)
- `Pong` (réponse à un `Ping`, adresse IP)
- `Query` (recherche de fichier selon certains critères)
- `Query-Hit` (réponse à un `Query`, avec liste des fichiers et adresse IP)
- `Push` (mécanisme de passage de *firewall*)

TTL : nombre de sauts encore réalisables

Sauts : nombre de sauts réalisés

- $TTL_n + Sauts_n = TTL_{initial}$

Longueur : taille du bloc **données** en octets

Données : peuvent être vides

Gnutella : Identification des pairs (1)

Détection active des pairs

- requête `Ping`
  - ✓ pas de données
  - ✓ limitations des envois pour ne pas saturer le réseau
  - ✓ crée un état dans la table de routage (retour des `Pong`)
  - ✓ répondre et relayer aux pairs connectés (limite du TTL)
- réponse `Pong`
  - ✓ données :

Port (2 octets)	Adresse IP (4 octets)	Nb de fichiers (4 octets)	Nb de Ko partagés (4 octets)

Port : port sur lequel le pair est en attente

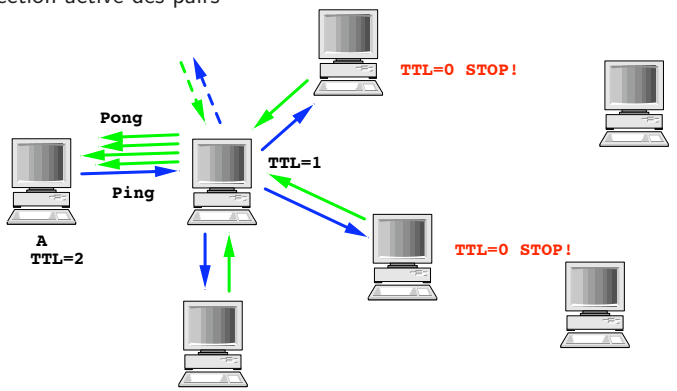
Adresse IP : adresse à laquelle le pair est atteignable

Nb de fichiers : nombre de fichiers partagés par le pair

Nb de Ko partagés : quantité de données partagées par le pair

Gnutella : Identification des pairs (2)

Détection active des pairs



Gnutella : Recherche de fichier (1)

Requête pour trouver une information

- requête `Query`
  - ✓ données :

Vitesse minimum (2 octets)	Critères de recherche (N octets)

Vitesse : débit minimum pour qu'un pair réponde (Ko/s)

Critères : chaine de caractères terminée par 0x00

  - ✓ crée un état dans la table de routage (retour des `Query-Hit`)
  - ✓ relayer aux pairs connectés (limite du TTL)
- réponse `Query-Hit`...

Gnutella : Recherche de fichier (2)

- requête `Query`
- réponse `Query-Hit`
  - ✓ données :

Nb Hit (1 octet)	Port (2 octets)	Adresse IP (4 octets)	Vitesse (4 octets)	Résultats (N octets)	GID Pair (16 octets)

Nb Hit : indique le nombre de champs des **Résultats**

Port : port sur lequel le pair est en attente

Adresse IP : adresse à laquelle le pair est atteignable

Vitesse : débit minimum demandé (Ko/s)

Résultats : ensemble de **Nb Hit** champs :

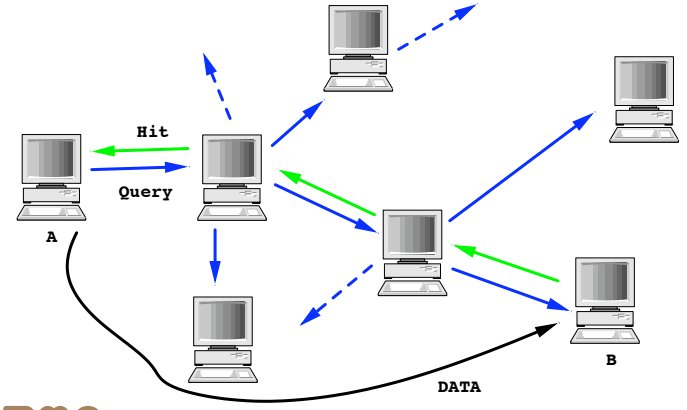
Index du fichier (4 octets)	Taille du fichier (4 octets)	Nom du fichier chaine terminant par 0x0000

GID Pair : identification pour un `Push`

  - ✓ sont routées selon le chemin inverse suivi par reqêtes `Query`

Gnutella : Recherche de fichiers (3)

Requête pour trouver une information



Gnutella : *Firewall* (1)

Retourner la connexion des données

- requête `Push`
  - ✓ données :

GID Pair (16 octets)	Index du fichier (4 octets)	Adresse IP (4 octets)	Port (2 octets)

GID Pair : identification du pair

Index : identifiant unique du fichier sur le pair

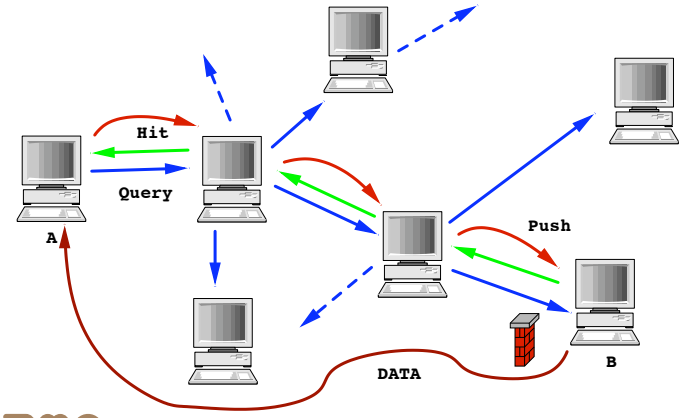
Adresse IP : adresse à laquelle le fichier doit être envoyé

Port : port sur lequel le destinataire est en attente

  - ✓ sont routées selon le chemin inverse suivi par réponses `Query-Hit`
  - ✓ permet la création de la connexion donnée à partir du pair

Gnutella : *Firewall* (2)

Retourner la connexion des données



Gnutella : Gestion des connexions

Connexion de contrôle sur TCP

- demande de connexion :
  ```
  GNUTELLA CONNECT/0.6
  Node: 201.33.182.178:6346
  User-Agent: gtk-gnutella/0.80 beta2 - 22/01/2002
  ```
- réponse du pair :
  ```
  GNUTELLA/0.6 200 OK
  User-Agent: Morpheus 2.0.1.7
  Remote-IP: 181.185.36.178
  ```
- confirmation :
  ```
  GNUTELLA/0.6 200 OK
  ```

Récupération des données par **HTTP**

- séparée du réseau Gnutella :
  ✓ connexion directe entre pairs et envoi d'un `GET`

Gnutella : Remarques

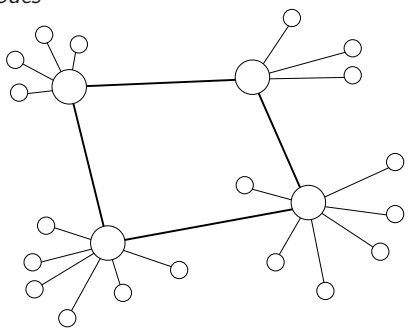
Leçons retenues :

- saturation des petits pairs (modems)
  ✓ possibilité d'indiquer que l'on a un fichier mais que l'on est saturé
- taille du réseau atteignable limitée (rupture de connectivité liée aux modems)
  ✓ création d'une hiérachie de pairs
- **anonymat ?**
  ✓ le pair où l'on récupère le fichier nous connait
  ☞ ∃ protocoles permettant de ne pas connaitre le destinataire

Evolutions P2P

Gnutella2, **KaZaA** (réseau FastTrack)

- Hôtes hétérogènes
- Topologie hiérarchique
  ✓ *Super-Nodes*



Plan

BitTorrent (1)

Partage d'un fichier :
- découpage en bloc de 64Ko à 1Mo *(Chunk)*
- création d'un `.torrent`
  - ✓ méta-données
  - ✓ signature pour chacun des *chunks*
- mise en place d'un *tracker*
  - ✓ machine qui supervise la distribution
- échange de données entre tous les demandeurs *(leechers)*
  - ✓ la source *(seed)* n'est sollicitée que pour amorcer

Spécificité :
- pas de système de recherche
- pas de téléchargement direct (type HTTP)
- avantages :
  - ✓ économique
  - ✓ redondant
  - ✓ résistance aux *flash-crowd*

BitTorrent (2)

Stratégies :
- sélection de pair
  - ✓ *tit-for-tat* + *choking*
    - ☞ encourage la coopération et diminue les *free-riders*
    - ☞ sélectionne les meilleurs contributeurs, étouffe les autres
    - ☞ mécanisme périodique (10 s)
  - ✓ *optimistic unchoke*
    - ☞ découverte de nouveaux pairs
    - ☞ alimente un nouveau pair aléatoirement
    - ☞ mécanisme périodique (30 s)
- sélection de *chunk* :
  - ✓ *rarest first*
    - ☞ donne le *chunk* le plus rare en premier
    - ☞ maximise l'entropie de chaque *chunk*
  - ✓ *random first*
    - ☞ pour accélérer le démarrage des nouveaux

BitTorrent (3)

Evolutions :
- Indexage/recherche
  - ✓ actuellement moteurs de recherche spécialisés (web) :
    - ☞ `http://thepiratebay.org/`
    - ☞ `http://www.mininova.com/`
    - ☞ `http://www.demonoid.com/` (abonnement)
    - ☞ ...
  - ✓ *tracker* distribué (table de hachage distibuée)
    - ☞ basée sur Kademlia
- *multitracker*
  - ✓ redondance
  - ✓ surcout en signalisation
- cryptage des échanges
  - ✓ *Protocol header encrypt (PHE)*
  - ✓ *Message stream encryption/Protocol encryption (MSE/PE)*
- distribution de contenus *(streaming A/V)*
  - ✓ nombreux projets...

P2P : Autres

Partage de fichier **complètement anonyme**

- **Freenet**
  - ✓ *peer-to-peer* décentralisé (comme Gnutella)
    - ☞ connaissance locale seulement
    - ☞ accès aux ressources de proche en proche (routage)
    - ☞ producteur anonyme
    - ☞ consommateur anonyme
    - ☞ résistance aux tentatives de limitation d'accès

Systèmes *peer-to-peer* structurés de recherche par le contenu :

- **Chord**
  - ✓ identificaton par clé (SHA-1 sur une chaine)
  - ✓ localisation par clé (SHA-1 sur l'adresse du noeud)
    - ☞ positionnement sur le nœud successeur le plus proche
- **Tapestry**
  - ✓ routage des identificateurs (hash) selon le suffixe des nœuds
- **CAN** (*Content Addressable Network*)
  - ✓ système de coordonnées cartésiennes virtuelles
- ...

Fin

Document réalisé avec LaTeX.
Classe de document `foils`.
Dessins réalisés avec `xfig`.

Olivier Fourmaux, `olivier.fourmaux@upmc.fr`
`http://www-rp.lip6.fr/~fourmaux`

Ce document est disponible en format PDF sur le site :
`http://www-master.ufr-info-p6.jussieu.fr/`

U.E. ARES
Architecture des Réseaux

Cours 3/6 : Couche transport

Olivier Fourmaux
(olivier.fourmaux@upmc.fr)

Version 5.4

Couche transport

Compréhension des principes de base de la couche transport[1]

- multiplexage
- transfert fiable
- contrôle de flux
- contrôle de congestion

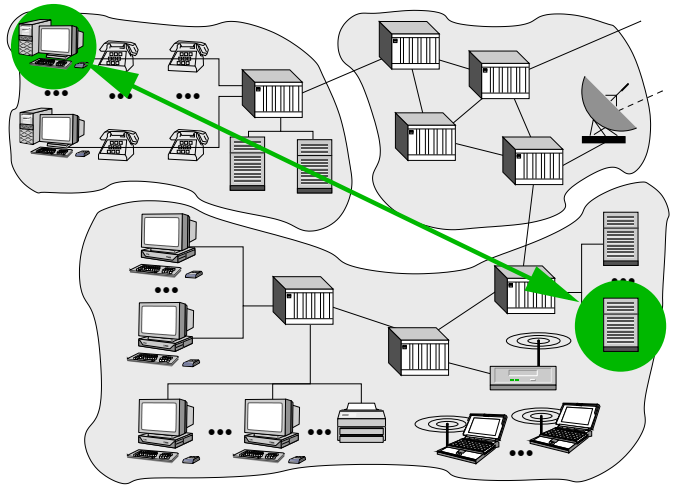
Etude des protocoles de transport dans l'Internet

- UDP : transport sans connexion
- TCP : transport orienté-connexion
- contrôle de congestion de TCP

[1] Nombreuses adaptations des slides, des schémas et du livre de J. F. Kurose et K. W. Ross, *Computer Networking : A Top Down Approach Featuring the Internet*, 3e edition chez Addisson-Wesley, juillet 2004.

Plan

Couche transport



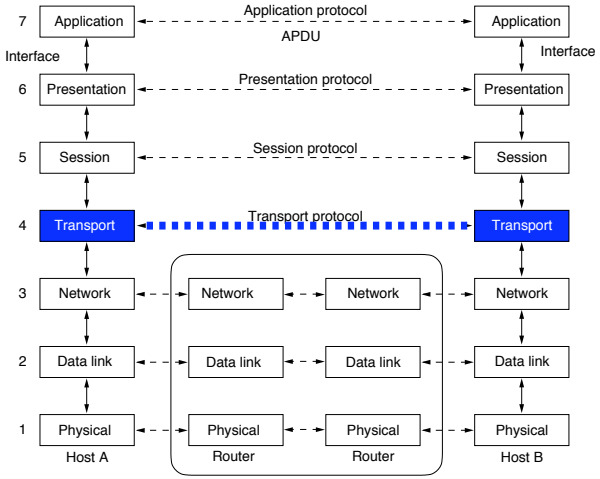
Couche transport

La **Couche transport** permet de faire **communiquer directement** deux ou plusieurs entités sans avoir à se préoccuper des différents éléments de réseaux traversés :

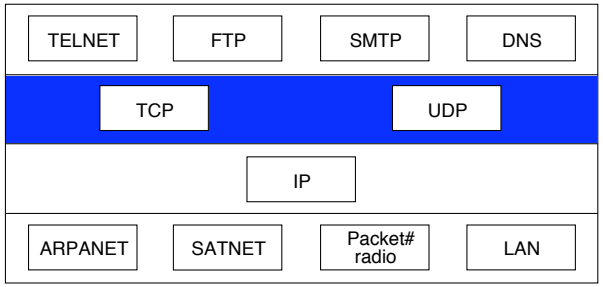
- associations virtuelles entre **processus**
- communication de bout-en-bout (*end-to-end*)
  - ✓ abstraction de la **topologie** et des **technologies** sous-jacentes
  - ✓ fonctionne dans les machines d'extrémité
    - ☞ **émetteur** : découpe les messages de la couche applicative en segments et les "descend" à la couche réseau
    - ☞ **récepteur** : réassemble les segments en messages et les "remonte" à la couche application

➠ 2 **modèles** définissent les fonctionnalités associés à chaque couche...

Couche transport : OSI



Couche transport : TCP/IP



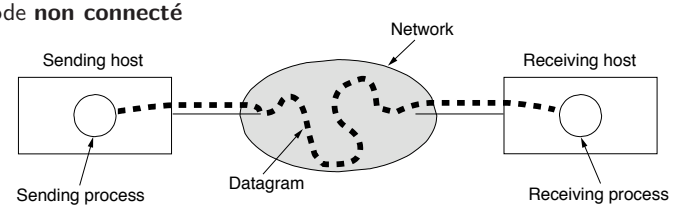
Couche transport : Internet

2 protocoles de transport standard : TCP et UDP

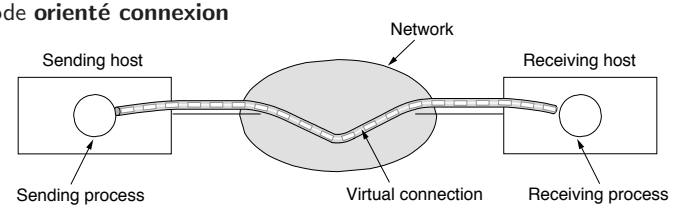
- transfert fiable et ordonné : **TCP**
  - ✓ gestion de la connexion
  - ✓ contrôle de flux
  - ✓ contrôle de congestion
- transfert non fiable non ordonné : **UDP**
  - ✓ service *best effort* ("au mieux") d'IP
  - ✓ très léger
- non disponible :
  - ✓ garanties de débit
  - ✓ garanties temporelles
    - ☞ délais non bornés
    - ☞ jigue imprévisible

Couche transport : 2 approches

Mode **non connecté**



Mode **orienté connexion**

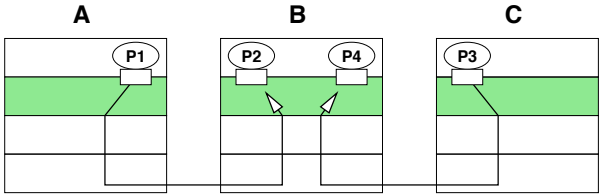


Plan

Multiplexage/Démultiplexage

Les **processus** applicatifs transmettent leurs données au système à travers des **sockets** : Le **multiplexage** consiste à regrouper ces données.

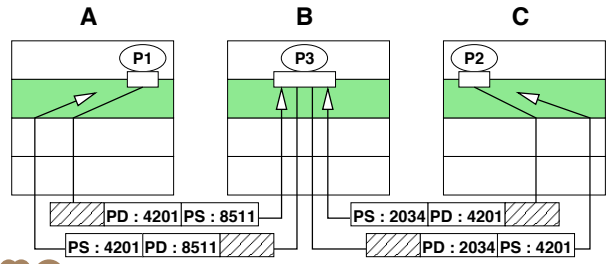
- **mux** (à l'émetteur) :
  - ✓ ajout d'un entête à chaque bloc de données d'un socket
  - ✓ collecte les données de plusieurs socket
- **demux** (au récepteur) :
  - ✓ fourniture des données au socket correspondant



Démultiplexage en mode non connecté

Association d'un socket avec un numéro de port

- identification du `DatagramSocket` : (`@IPdest, numPortDest`)
- réception d'un **datagramme** à un hôte :
  - ✓ vérification du `numPortDest` contenu
  - ✓ envoi au socket correspondant à `numPortDest`
    - ☞ ∀ `@IPsource`, ∀ `numPortSource`



Multiplexage en mode orienté connection

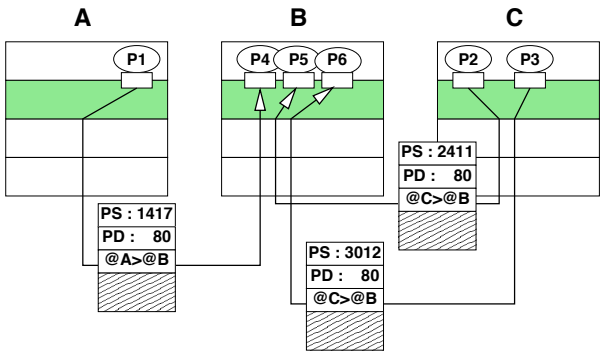
Association relative à une **connexion** entre deux processus

- identification du `StreamSocket` par le quadruplet :
  - ✓ adresse source : `@IPsource`
  - ✓ port source : `numPortSource`
  - ✓ adresse destination : `@IPdest`
  - ✓ port destination : `numPortDest`
- réception d'un **segment** à un hôte :
  - ✓ vérification du quadruplet contenu
  - ✓ envoi au socket correspondant au quadruplet
    - ☞ un serveur web peut avoir plusieurs connexions simultanée

Démultiplexage en mode orienté connection (1)

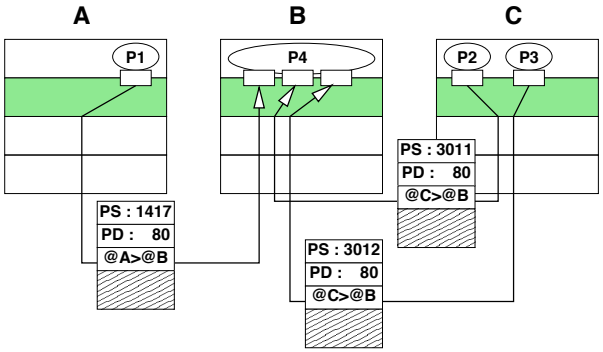
Serveur web classique (apache 1.x)

- un socket par connexion
  - ✓ HTTP en mode non persistant : un socket par requête !

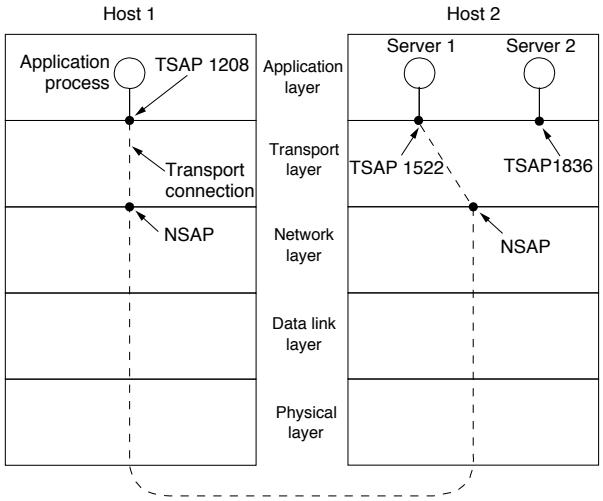


Démultiplexage en mode orienté connection (2)

Serveur web multi-threadé (apache 2.x)



Multiplexage : dénominations OSI



Plan

Rappels sur la couche transport

Multiplexage et démultiplexage

UDP : un protocole en mode non connecté
- format du datagramme UDP
- utilisation d'UDP

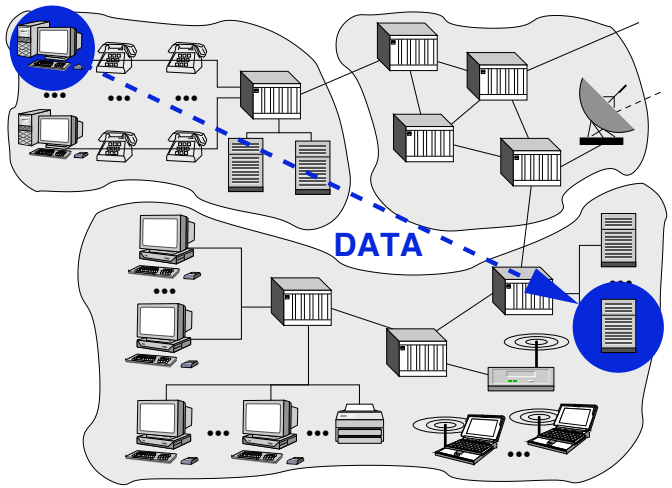
Principes de transfert de données fiable

TCP : un protocole en mode orienté connexion

Principes de contrôle de congestion

Contrôle de congestion de TCP

UDP



UDP

User Datagram Protocol [RFC 768]

- protocole de transport Internet basique (sans fioriture)
- service *best effort* :
  - ✓ les datagrammes transférés peuvent être...
    - ☞ perdus
    - ☞ dupliqués
    - ☞ désordonnés
- service sans connexion :
  - ✓ pas d'échange préalable
  - ✓ pas d'information d'état aux extrémités
    - ☞ chaque datagramme géré indépendamment

Plan

Rappels sur la couche transport

Multiplexage et démultiplexage

UDP : un protocole en mode non connecté
- **format du datagramme UDP**
- utilisation d'UDP

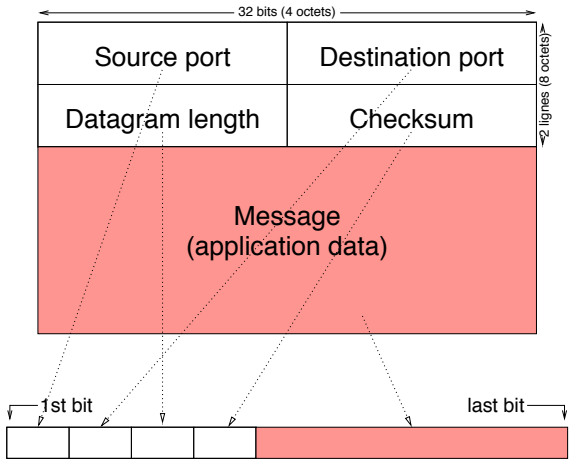
Principes de transfert de données fiable

TCP : un protocole en mode orienté connexion

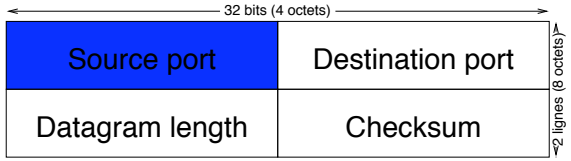
Principes de contrôle de congestion

Contrôle de congestion de TCP

Datagramme UDP

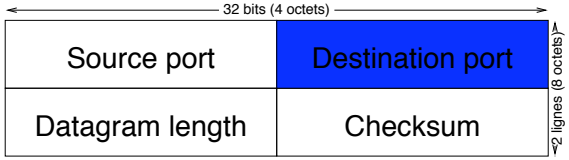


UDP : port source



- 16 bits (65535 ports)
- **multiplexage** à la source
- identification du socket pour un retour potentiel
- allocation fixe ou dynamique (généralement dans le cas d'un client)
- répartition de l'espace des ports :
  - ✓ $0 \leq \texttt{numPort} \leq 1023$: accessible à l'administrateur
    - ☞ socket serveurs (généralement)
  - ✓ $1024 \leq \texttt{numPort}$: accessible aux utilisateurs
    - ☞ socket clients (généralement)

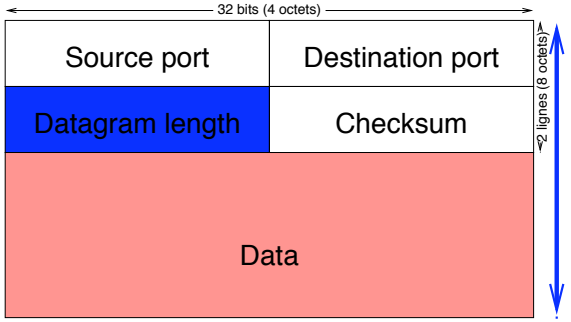
UDP : port destination



- 16 bits (65535 ports)
- **démultiplexage** à la destination
- le destinataire doit être à l'écoute sur ce port
- négociation du port ou *well-known ports* (numéros de port réservés) :

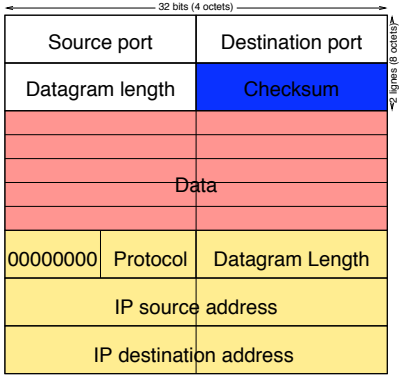
```
Unix> cat \etc\services |grep udp          ..
echo            7/udp                      domain          53/udp
discard         9/udp                      tftp            69/udp
daytime         13/udp                     gopher          70/udp
chargen         19/udp                     www             80/udp
ssh             22/udp                     kerberos        88/udp
time            37/udp                     snmp            161/udp
..                                         snmp-trap       162/udp
                                           ....
```


UDP : longueur du datagramme



- 16 bits (64 Koctets maximum)
- longueur totale avec les données exprimée en **octets**

UDP : contrôle d'érreur



- 16 bits
- contrôle d'erreur **facultatif**
- émetteur :
  - ✓ ajout *pseudo-header*
  - ✓ datagram+ = suite $mot_{16\text{bits}}$
  - ✓ *checksum*[2] = $\overline{\sum mot_{16\text{bits}}}$
- récepteur :
  - ✓ ajout *pseudo-header*
  - ✓ recalcul de $\sum mot_{16\text{bits}}$
    - ☞ $= 0$: pas d'erreur détectée toujours possible...
    - ☞ $\neq 0$: erreur (destruction silencieuse)

[2] Somme binaire sur 16 bits avec report de la retenue débordante ajoutée au bit de poid faible

Plan

UDP : arguments pour un transport sans connexion

Le choix d'un service transport non connecté peut être motivé par :
- ressources limitées aux extrémités
  - ✓ pile TCP/IP limitée
  - ✓ absence d'**état** dans les hôtes
  - ✓ capacité de traitement limitée
- besoin d'échange rapide
  - ✓ pas d'**établissement** de connexion
- besoin d'éfficacité
  - ✓ **entête réduit**
- contraintes temporelles
  - ✓ **retransmission** inadapté
  - ✓ pas de **contrôle** du débit d'émission
- besoin de nouvelles fonctionnalités
  - ✓ remontés dans la couche application...

UDP : exemples d'applications

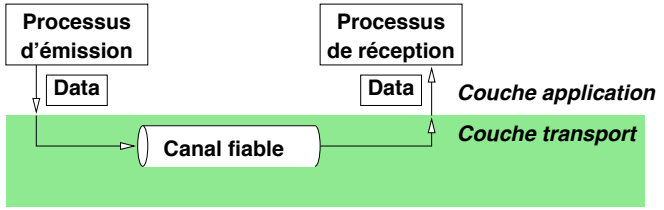
- les applications suivantes reposent typiquement sur UDP :
  - ✓ résolution de noms (DNS)
  - ✓ administration du réseau (SNMP)
  - ✓ protocole de routage (RIP)
  - ✓ protocole de synchronisation d'horloge (NTP)
  - ✓ serveur de fichiers distants (NFS)
    - ☞ fiabilisation implicite par redondance temporelle
    - ☞ fiabilisation explicte par des mécanismes ajoutés dans la couche application
- toutes les applications *multicast* ➠ U.E. **ING**
- et les applications multimédia ➠ U.E. **MMQOS**
  - ✓ diffusion multimédia, *streaming* audio ou vidéo
  - ✓ téléphonie sur Internet
  - ✓ visioconférence
    - ☞ contraintes temporelles
    - ☞ tolérance aux pertes

UDP : Interface socket

```
#include <sys/types.h>
#include <sys/socket.h>

# Create a descriptor
int socket(int domain, int type, int protocol);
#   domain : PF_INET for IPv4 Internet Protocols
#     type : SOCK_DGRAM Supports datagrams (connectionless, unreliable msg of a fixed max length)
# protocol : UDP (/etc/protocols)

# Bind local IP and port
int bind(int s, struct sockaddr *my_addr, socklen_t addrlen);

# Send an outgoing datagram to a destination address
int sendto(int s, const void *msg, size_t len, int flags,
                                    const struct sockaddr *to, socklen_t tolen);

# Receive the next incoming datagram and record is source address
int recvfrom(int s, void *buf, size_t len, int flags,
                                    struct sockaddr *from, socklen_t *fromlen);

# End : dealocate
int close(int s);
```


Plan

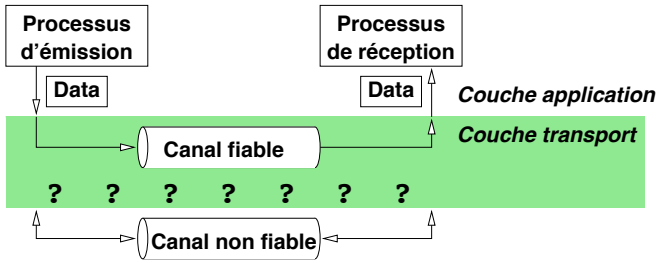
Couche transport et fiablité (1)



Problématique multi-couche :

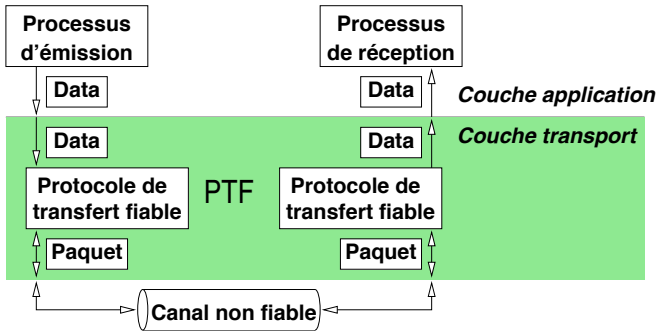
- couche application
- couche transport
- couche liaison

Couche transport et fiablité (2)

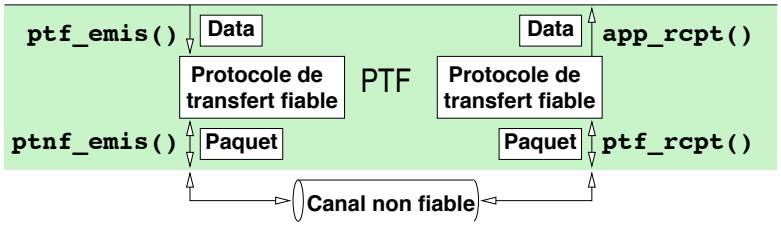


Les cartactéristiques du **canal non fiable** determine la complexité du **protocol de transfert fiable** (PTF).

Couche transport et fiablité (3)



Protocole de Transfert Fiable (PTF)



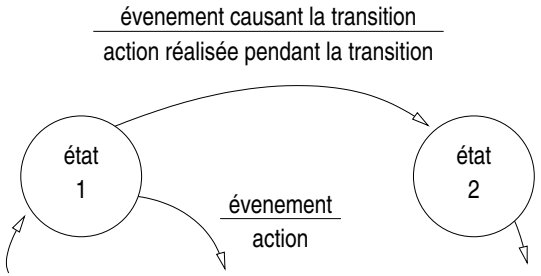
Primitives de base du PTF :

- `ptf_emis()` : appelée par la couche supérieure (application) pour envoyer des données à la couche correspondante du récepteur
- `ptfn_emis()` : appelée par le PTF transférer un paquet sur le canal non fiable vers le récepteur
- `ptf_rcpt()` : appelée lorqu'un paquet arrive au récepteur
- `app_rcpt()` : appelée par le PTF pour livrer les données

PTF et AEF

Nous allons construire progressivement le **PTF**

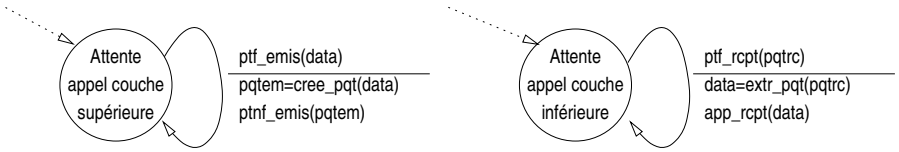
- transfert de données dans un seul sens
  - ✓ information de contrôle dans les 2 directions
- spécification de l'émetteur et du récepteur par des Automates à Etats Finis (AEF) :



PTF v1.0

Transfert fiable sur un canal sans erreur

- canal sous-jacent complètement fiable
  - ✓ pas de bits en erreur
  - ✓ pas de perte de paquets
- automates séparés pour l'émetteur et le récepteur :



émetteur *récepteur*

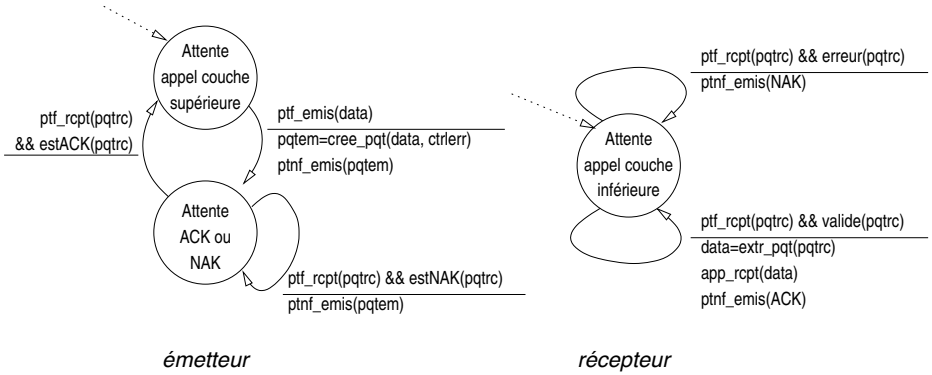
PTF v2.0

Transfert fiable sur un **canal avec des erreurs**

- canal sous-jacent pouvant changer la valeur des bits d'un paquet
  - ✓ introduction de contrôle d'erreur :
    - ☞ `ctrlerr` : redondance rajoutée au paquet
- *Comment récupérer les erreurs ?*
  - ✓ **acquittement** (ACK) : le récepteur indique explicitement la réception correcte d'un paquet
  - ✓ **acquittement négatif** (NAK) : le récepteur indique explicitement la réception incorrecte d'un paquet
    - ☞ l'émetteur ré-émet le paquet sur réception d'un NAK
- nouveau mécanisme dans PTV v2.0 :
  - ✓ détection d'erreur
    - ☞ `valide(pqt)` : vrai si le contrôle d'erreur de `pqt` est correct
    - ☞ `erreur(pqt)` : vrai si le contrôle d'erreur de `pqt` est incorrect
  - ✓ retour d'information *(feedback)* du récepteur (ACK et NAK)

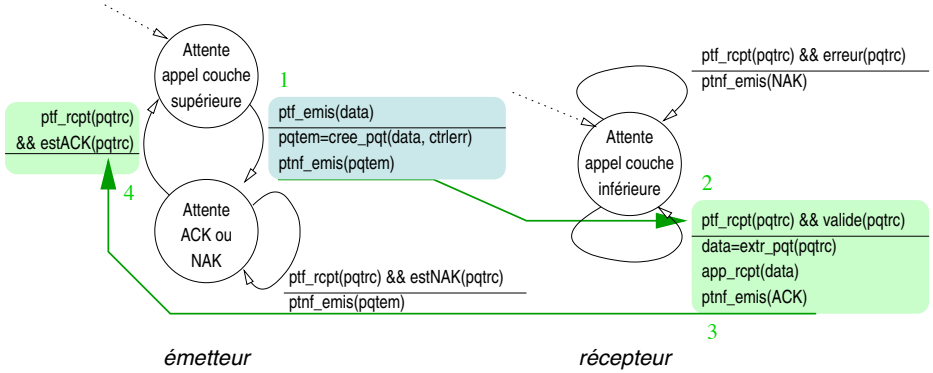
PTF v2.0

Transfert fiable sur un **canal avec des erreurs** :



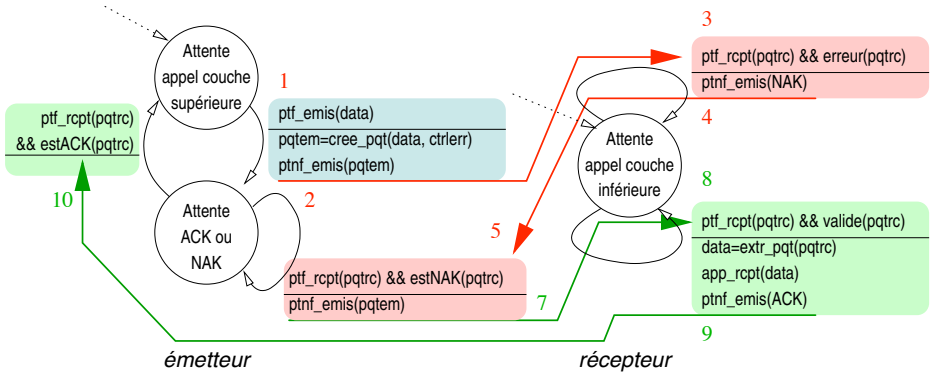
PTF v2.0 : ACK

Transfert fiable lorsqu'il n'y a pas d'erreur :



PTF v2.0 : NAK

Transfert fiable lorqu'il y a une erreur :



PTF v2.0 : discussion

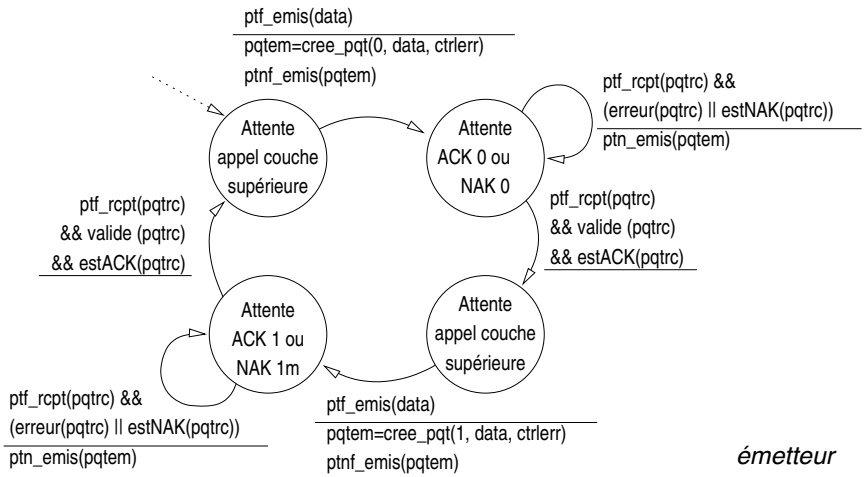
PTF v2.0 est un protocole *stop and wait* :

- émetteur envoi un paquet et attend la réponse du récepteur
- peu performant...

PTF v2.0 à un défaut majeur !

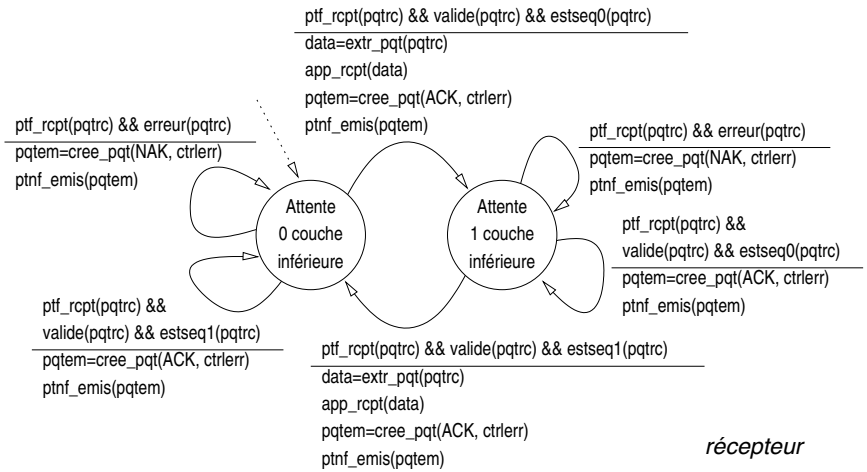
- *Que se passe-t-il si les ACK ou NAK sont incorrect ?*
  - ✓ pas d'information sur l'état du récepteur
  - ✓ une retransmission simple risque de dupliquer les données
- gestion des duplicats :
  - ✓ émetteur **retransmet** le paquet courant si ACK/NAK incorrect
  - ✓ émetteur insert un **numéro** de séquence à chaque paquet
  - ✓ récepteur **supprime** les paquets dupliqués
    - ☞ inclu dans **PTF v2.1**

PTF v2.1 : émetteur



émetteur

PTF v2.1 : récepteur



récepteur

PTF v2.1 : discussion

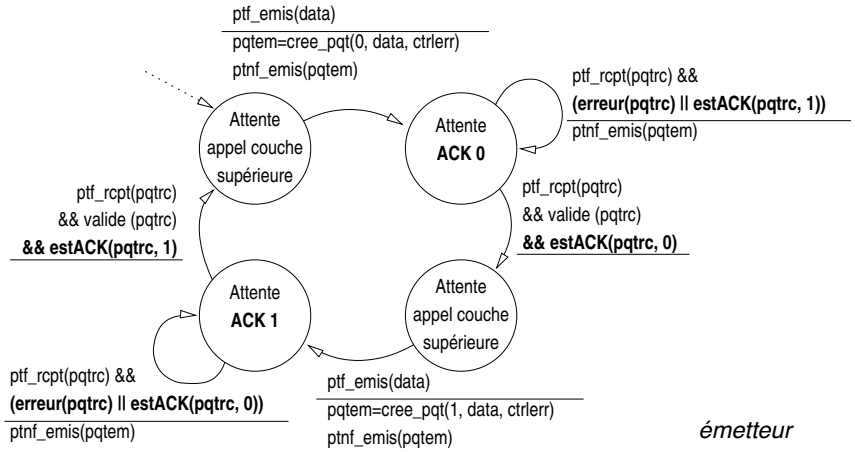
Comportement des extrémités avec PFT v2.1

- **émetteur**
  - ✓ ajout de numéro de séquence à chaque paquet
    - ☞ 2 suffisent (0 et 1)
  - ✓ contrôle d'erreur sur les ACK et NAK
  - ✓ 2 fois plus d'états
- **récepteur**
  - ✓ vérification que le paquet n'est pas dupliqué
    - ☞ l'état où l'on se trouve indique le numéro de séquence attendu

Peut-on éliminer les NAK ?

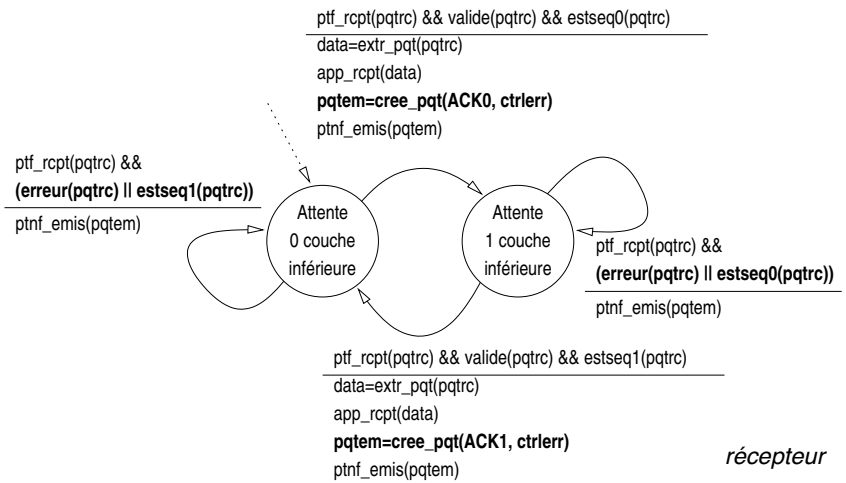
- remplacement des NAK par **ACK du dernier paquet** valide recu
  - ✓ récepteur inclu le numéro de séquence correspondant dans le ACK
  - ✓ ACK dupliqué au récepteur = NAK recu au récepteur
    - ☞ intégré dans **PFT v2.2**

PTF v2.2 : émetteur



émetteur

PTF v2.2 : récepteur



récepteur

PTF v3.0

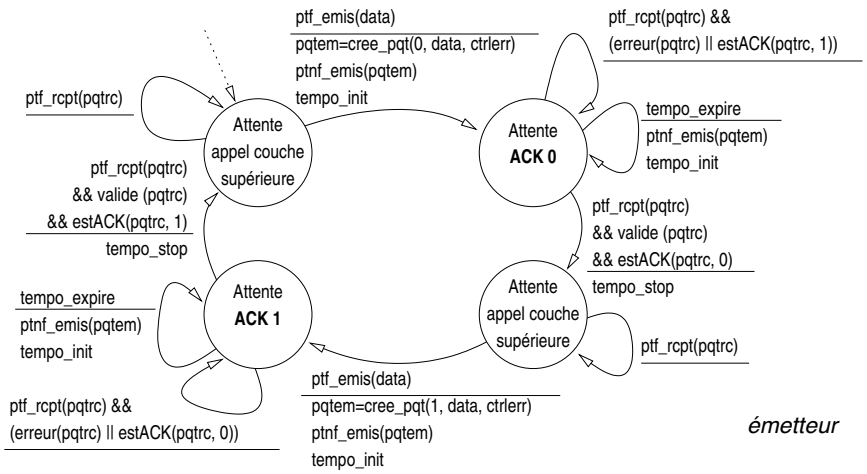
Transfert fiable sur un **canal avec erreurs et pertes**

- canal sous-jacent peut aussi perdre des paquets (data ou ACK)
  - ✓ `ctrlerr` + `numSeq` + ACK + retransmission
    - ☞ insuffisant : l'absence d'un paquet bloque l'automate !

Temporisation des retransmission

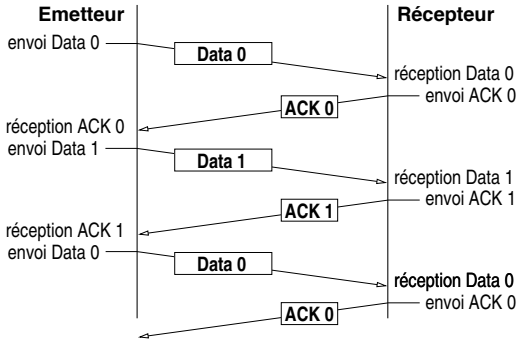
- estimation d'un temps de retour de ACK raisonable
  - ✓ déclenchement d'une temporisation à l'emission d'un paquet
    - ☞ `tempo_init`
  - ✓ ACK avant l'expiration de la temporisation ➠ rien
    - ☞ `tempo_stop`
  - ✓ pas de ACK à l'expiration de la temporisation ➠ retransmission
    - ☞ `tempo_expire`
- si le ACK est seulement en retard...
  - ✓ retransmission = duplication
    - ☞ détectée grâce au numéro de séquence

PTF v3.0 : émetteur

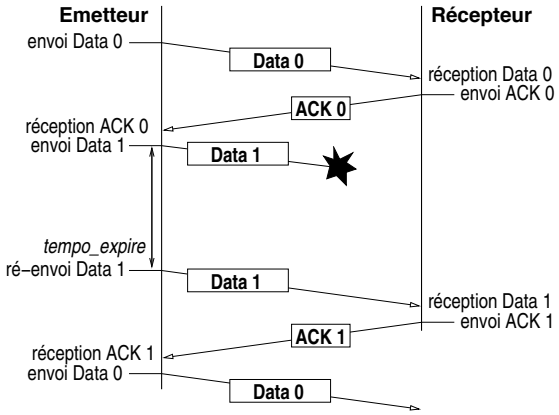


émetteur

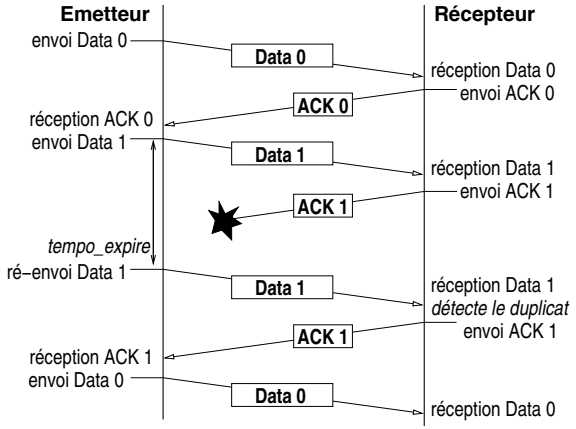
PTF v3.0 : sans perte



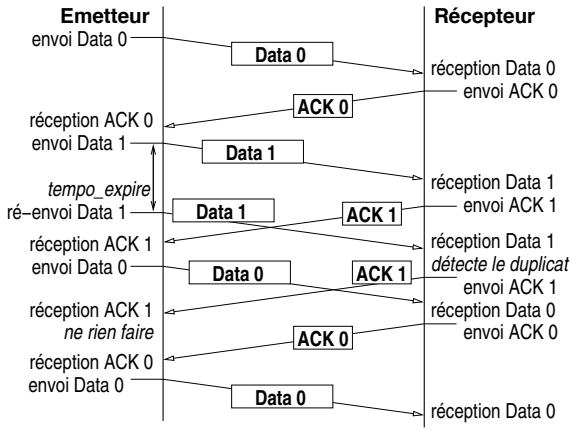
PTF v3.0 : perte d'un paquet de données



PTF v3.0 : perte d'un ACK



PTF v3.0 : fin de temporisation prématurée

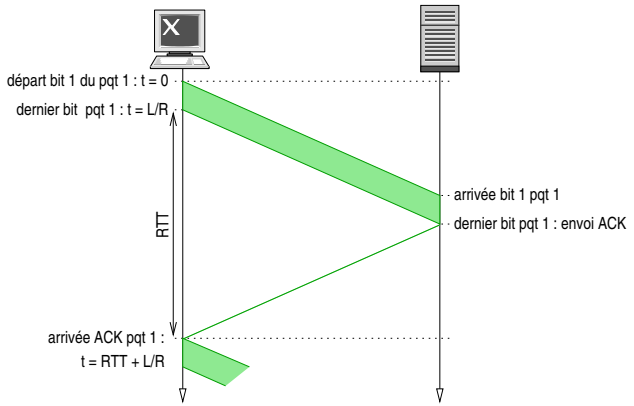


PTF v3.0 : performance

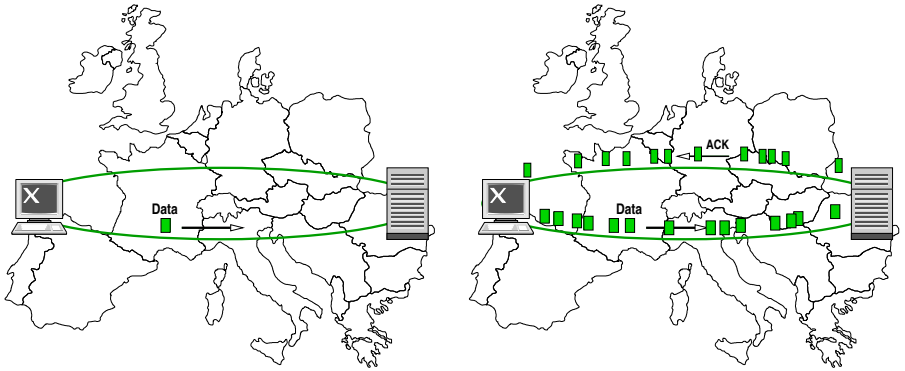
PFT v3.0 fonctionne mais quelles sont ses performances ?

- exemple de communication :
  - ✓ débit du lien : $D_{reseau} = 1\ Gbps$,
  - ✓ délais de bout-en-bout : $d = 40\ ms$ ($d_{AR} = 80\ ms$)
  - ✓ paquets de longueur 1000 octets ($L_{paquet} = 8000\ b$)
- $T_{transmission} = L_{paquet}/D_{reseau} = 8.10^3/10^9 = 8\ \mu s$
- efficatité émetteur (E_{emis}) : fraction de temps en émission
  - ✓ $E_{emis} = \frac{L_{paquet}/D_{reseau}}{L_{paquet}/D_{reseau}+d_{AR}} = \frac{8.10^{-6}}{8.10^{-6}+8.10^{-2}} = \frac{1}{10000}$
  - ✓ $D_{transport} = L_{paquet}/d_{AR} = 8.10^3/8.10^{-2} = 100\ Kbps$
    - ☞ **le protocole limite l'utilisation des ressources disponibles**

PTF v3.0 : stop and wait

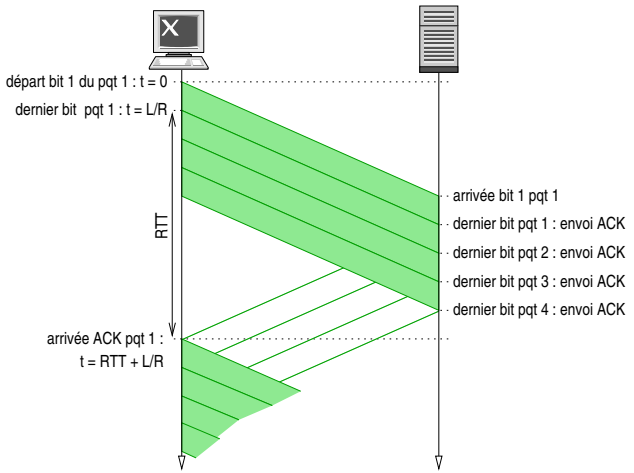


Protocole pipeline



- l'émetteur autorise plusieurs paquets en attente d'acquittement
  - ✓ numéro de sequences étendus
  - ✓ tampons d'émission et de réception
    - ☞ 2 types de protocole pipeliné : **Go-Back-N** et **Retransmissions sélectives**

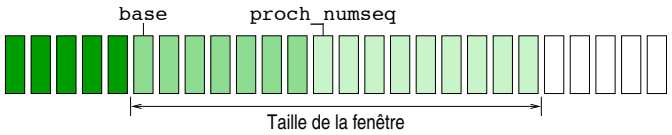
Performance pipeline



Go-Back-N : émetteur

Emetteur avec gestion *Go-Back-N* (retour arrière).

- entête des paquets avec k bits de numéro de séquence
- acquittements **cumulatifs**
  - ✓ `ACK(n)` acquitte tous les paquets jusqu'au numéro de séquence `n`
- fenêtre d'au maximun *N* paquets non acquités :



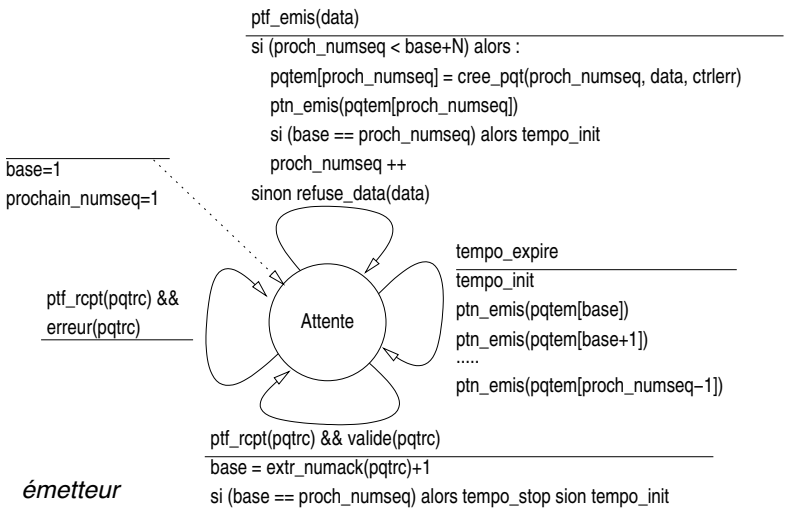
- temporisation pour chaque paquet en attente *(in-flight)*
  - ✓ `tempo_expire(n)` : retransmission du paquet `n` et des suivants avec numéro de séquence supérieur

Go-Back-N : Récepteur

Récepteur avec gestion *Go-Back-N* (retour arrière).

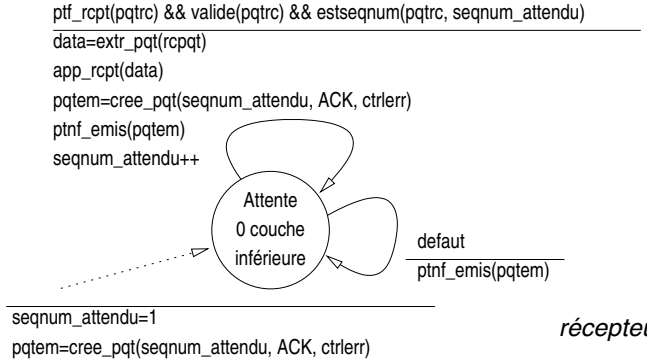
- **seulement des ACK** :
  - ✓ envoie toujours des ACK avec le plus élevé des `seqnum` de paquets valides **ordonnés**
    - ☞ peut générer des ACK dupliqués
    - ☞ seul `seqnum_attendu` est mémorisé
- **déséquencement** :
  - ✓ élimine les paquets déséquencés
    - ☞ pas de tampon au niveau du récepteur
  - ✓ ré-émet le ACK avec le plus élevé des `seqnum` de paquets valides **ordonnés**

PTF v4.0 : émetteur



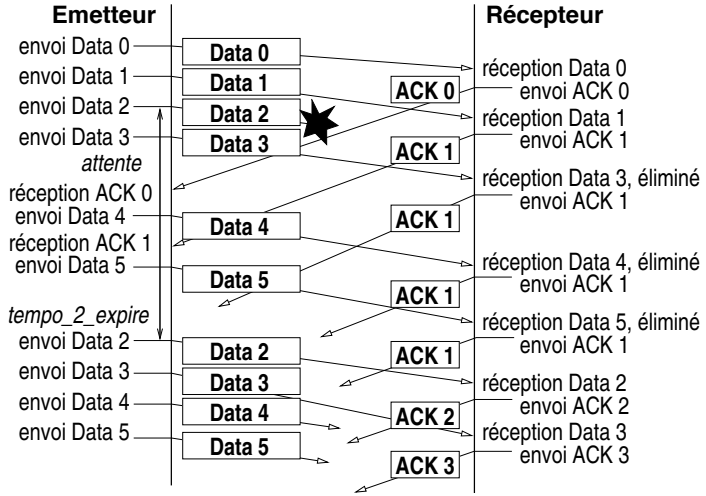
émetteur

PTF v4.0 : récepteur



récepteur

PTF v4.0 : exemple



Retransmissions sélectives (1)

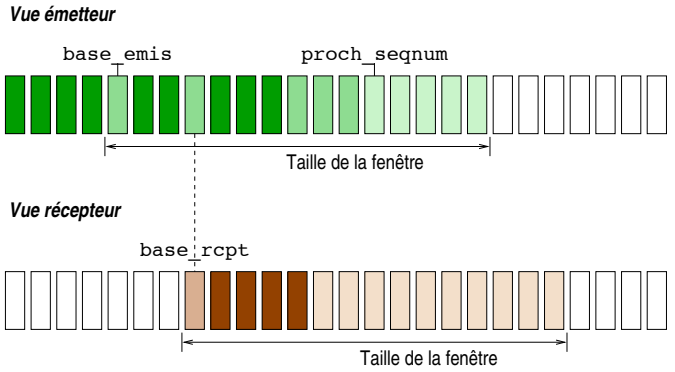
Emetteur

- retransmet **seulement** les paquets non acquittés
- fenêtre d'émission limité à *N* paquets consécutifs
- algo :
  - ✓ `pft_emis(data)`
    - ☞ envoi un paquet si `seqnum` dans la fenêtre
  - ✓ `tempo_expire(n)`
    - ☞ retransmet paquet `n`
    - ☞ `tempo_init(n)`
  - ✓ `ACK(n)`
    - ☞ marque le paquet `n` reçu
    - ☞ si `n` est le plus petit paquet non acquitté, décale la fenêtre

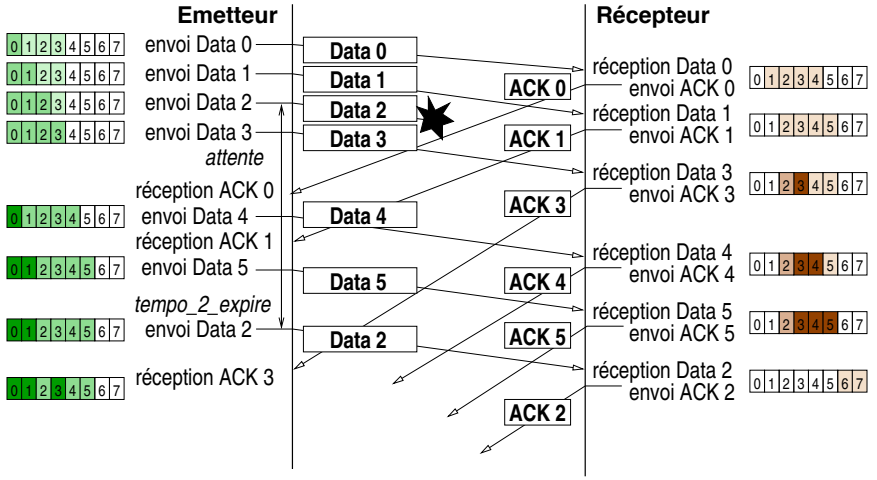
Récepteur

- acquitte **explicitement** chaque paquet valide reçu
- tampon de réception pour reséquencement
- algo :
  - ✓ `ptf_rcpt(n)`
    (`base_rcpt` $\leq$ `n` $\leq$ `base_rcpt`+N-1)
    - ☞ `ACK(n)`
    - ☞ si déséquensé : tampon
    - ☞ si séquense : `app_emis(data)`, est le plus petit paquet non acquitté, décale la fenêtre
  - ✓ `ptf_rcpt(n)`
    (`base_rcpt-N` $\leq$ `n` $\leq$ `base_rcpt-1`)
    - ☞ `ACK(n)`
  - ✓ autre
    - ☞ ignore

Retransmissions sélectives (2)



Retransmissions sélectives (3)



Plan

Rappels sur la couche transport

Multiplexage et démultiplexage

UDP : un protocole en mode non connecté

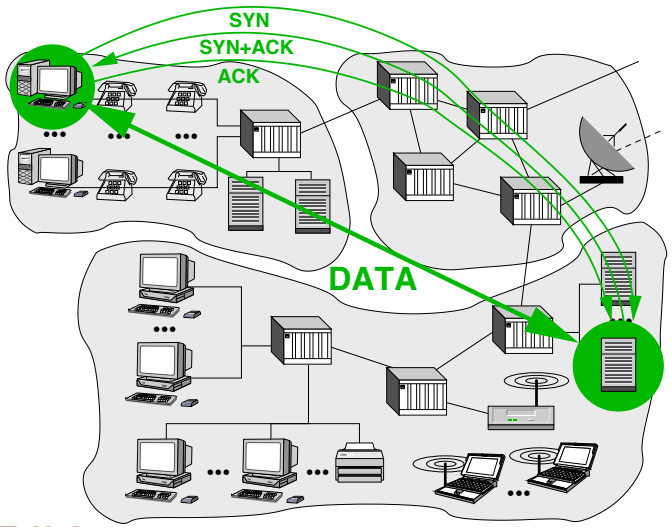
Principes de transfert de données fiable

TCP : un protocole en mode orienté connexion
- format du segment TCP
- gestion de la connexion
- calcul des temporisations
- mise en œuvre de la fiabilité
- contrôles de flux
- utilisation de TCP

Principes de contrôle de congestion

Contrôle de congestion de TCP

TCP (1)



TCP (2)

Transmission Control Protocol [RFCs : 793, 1122, 1323, 2018, 2474, 2581, 3168 et 4379]

- service **fiable**
  - ✓ mécanismes ARQ
- **point-à-point**
  - ✓ deux processus (généralement un client et un serveur)
- flot d'**octet** continu
  - ✓ pas de frontières de messages
- **orienté connexion**
  - ✓ ouverture en trois échanges (*three-way handshake*)
    - ☞ initiation des états aux extrémité avant l'échanges de données
  - ✓ fermetures courtoise ou brutale
- connexion **bidirectionnelle** (*full duplex*)
  - ✓ flux de données dans chaques sens
  - ✓ taille maximum du segment : MSS (*Maximun Segment Size*)
- **pipeline**
  - ✓ tampons d'émission et de réception
  - ✓ double fenêtre asservie aux contrôles de flux et de congestion

Plan

Rappels sur la couche transport

Multiplexage et démultiplexage

UDP : un protocole en mode non connecté

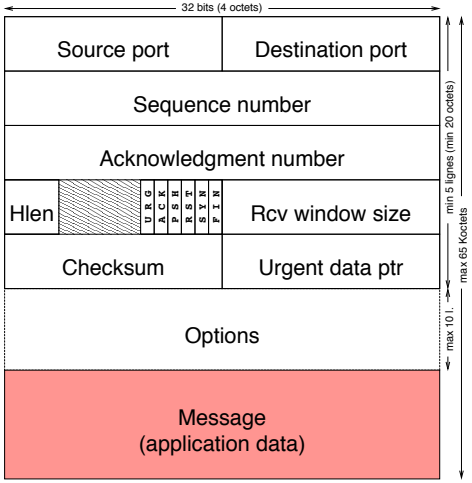
Principes de transfert de données fiable

TCP : un protocole en mode orienté connexion
- **format du segment TCP**
- gestion de la connexion
- calcul des temporisations
- mise en œuvre de la fiabilité
- contrôles de flux
- utilisation de TCP

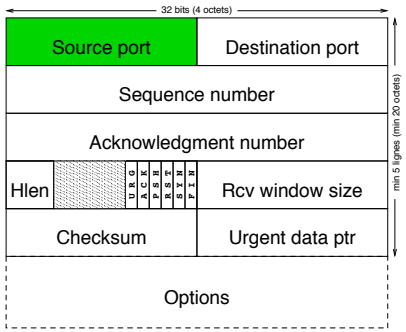
Principes de contrôle de congestion

Contrôle de congestion de TCP

Segment TCP

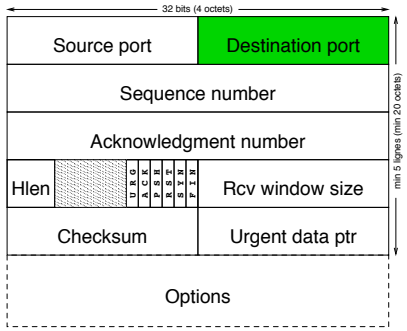


TCP : Port source



- 16 bits (65535 ports)
- **multiplexage** à la source
- identification partielle du socket (demi-association locale)
- allocation fixe ou dynamique (généralement dans le cas d'un client)
- répartition espace des ports :
  - ✓ $0 \leq \texttt{numPort} \leq 1023$: accessible à l'administrateur
    - ☞ socket serveurs (généralement)
  - ✓ $1024 \leq \texttt{numPort}$: accessible aux utilisateurs
    - ☞ socket clients (généralement)

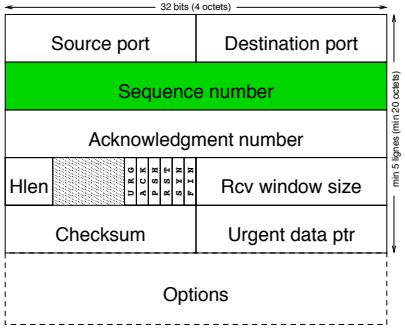
TCP : Port destination



- 16 bits (65535 ports)
- **démultiplexage** au niveau de la destination
- identification partielle du socket (demi-association distante)
- lors de la cration de l'association, le destinataire doit être à l'écoute sur ce port
- négociation du port ou *well-known ports* (numéros de port réservés) :

```
Unix> cat \etc\services|grep tcp
tcpmux          1/tcp
discard         9/tcp
systat          11/tcp
chargen         19/tcp
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp ..
telnet          23/tcp
smtp            25/tcp
whois           43/tcp
domain          53/tcp
gopher          70/tcp
finger          79/tcp
www             80/tcp
kerberos        88/tcp ....
```

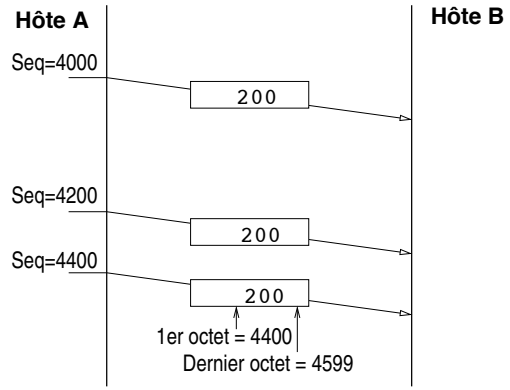

TCP : Numéro de séquence (1)



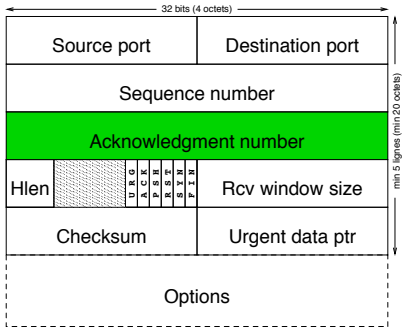
- 32 bits
- associé à chaque **octet** (et non pas à un segment)
  - ✓ numérote le **premier** octet des *data*
  - ✓ numérotation implicite des octets suivants
  - ✓ boucle au bout de 4 Goctets
- détection des **pertes**
- **ordonnancement**

TCP : Numéro de séquence (2)

Numérotation de chaque **octet** du flot continu de données



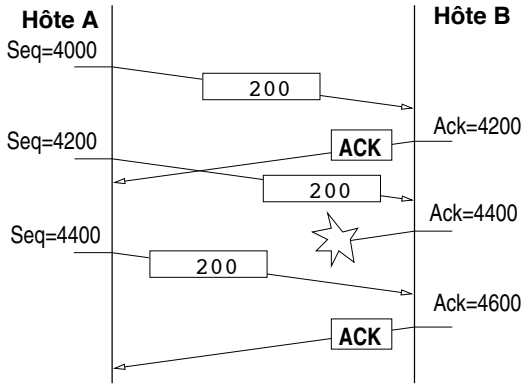
TCP : Numéro d'acquittement (1)



- 32 bits
- *piggybacking*
- indique le numéro du **prochain** octet attendu
- **cumulatif**, indique le premier octet non reçu (d'autres peuvent avoir été reçus avec des numéros de séquence supérieurs)

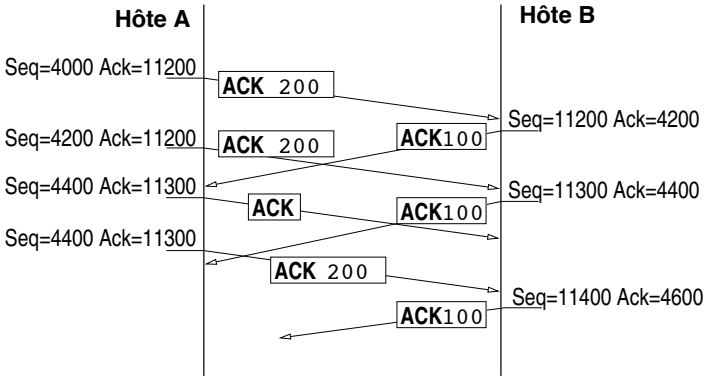
TCP : Numéro d'acquittement (2)

Acquittement de chaque **octet** du flot continu de données



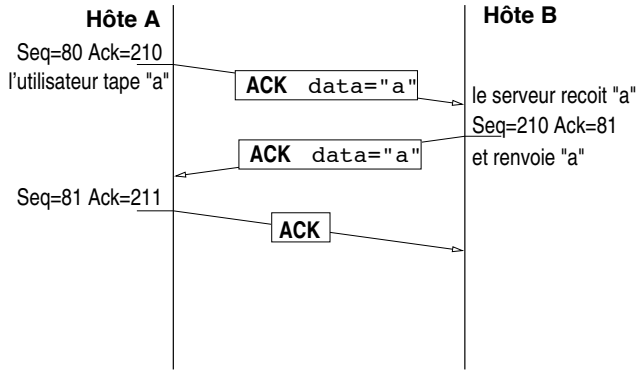
TCP : Numéro d'acquitement (3)

Piggybacking



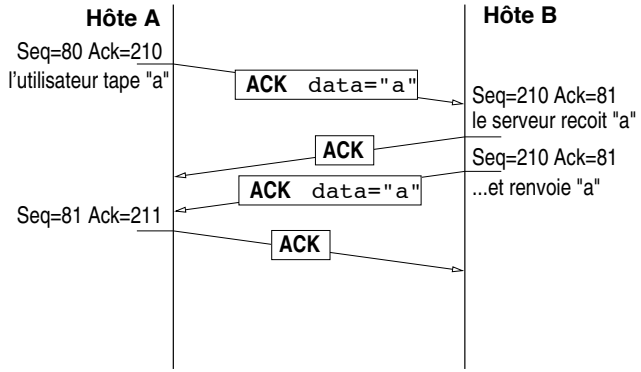
TCP : Exemple TELNET (1)

Emission d'un caractère frappé et renvoi par le serveur pour l'affichage



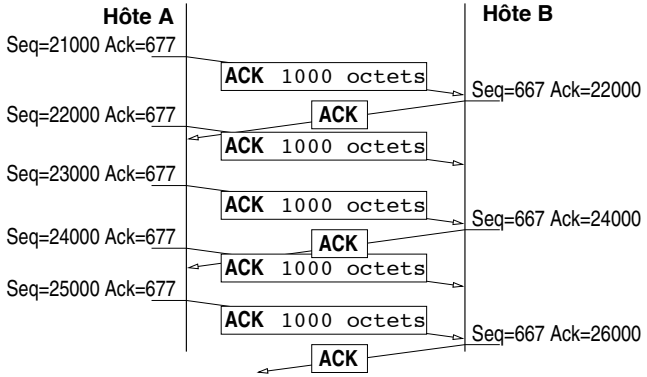
TCP : Exemple TELNET (2)

Les acquittements peuvent être plus rapide que l'application

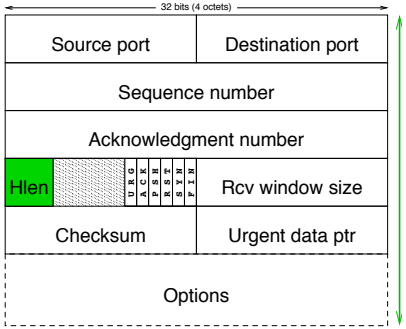


TCP : Acquittements temporisés

Delayed ACK (attente de deux segments ou 500 ms max.)

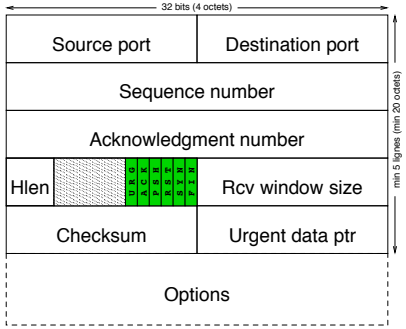


TCP : Longueur de l'entête



- 4 bits (valeur 15 max)
- nombre de lignes de 32 bits dans l'entête TCP
- nécessaire car le champ option est de longueur variable
  - ✓ valeur 5...
    - ☞ pas d'options
    - ☞ entête TCP de 20 octets minimum
  - ✓ ... à 15
    - ☞ 10 lignes d'options
    - ☞ 40 octets d'options max
    - ☞ entête TCP de 60 octets maximum

TCP : Indicateurs (*flags*)

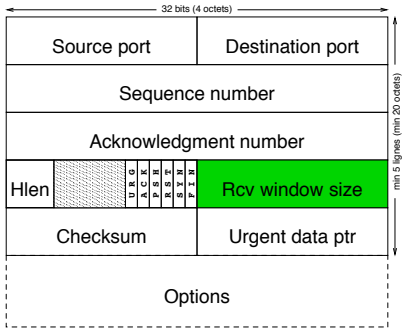


Chacun sur 1 bit indique :

- `URG` : présence de données **urgentes**
- `ACK` : le champ **acquittement** est valide
- `PSH` : envoi **immédiat** avec vidage des tampons
- `RST` : **terminaison** brutale de la connexion
- `SYN` : synchronisation lors de l'**ouverture**
- `FIN` : echanges terminaux lors d'une **fermeture** courtoise

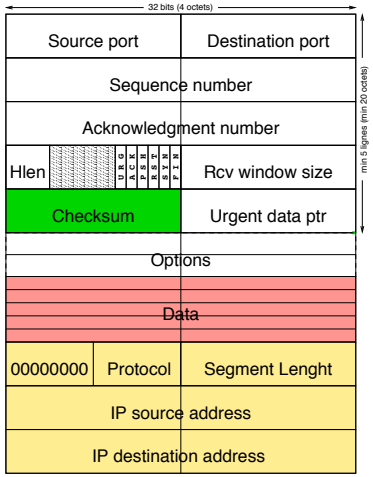
✓ il y en a d'autres récents
➠ U.E. **ING**

TCP : Taille de la fenêtre de réception



- 16 bits
  - ✓ le récepteur peut annoncer jusqu'à 64 Koctets
- *piggybacking*
- **contrôle de flux**
  - ✓ indique le nombre d'octets disponibles du coté du récepteur
  - ✓ dimentionne la taille de la fenêtre d'anticipation de l'émetteur

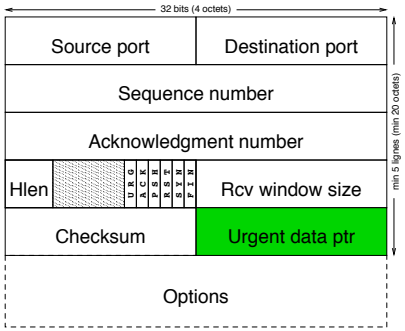
TCP : Somme de contrôle du segment



- 16 bits
- contrôle d'erreur (idem UDP)
- émetteur :
  - ✓ ajout *pseudo-header*
  - ✓ datagram$+=$ suite $mot_{16\text{bits}}$
  - ✓ *checksum*[3] $= \overline{\sum mot_{16\text{bits}}}$
- récepteur :
  - ✓ ajout *pseudo-header*
  - ✓ recalcul de $\sum mot_{16\text{bits}}$
    - ☞ $= 0$: pas d'erreur détectée toujours possible...
    - ☞ $\neq 0$: erreur (destruction silencieuse)

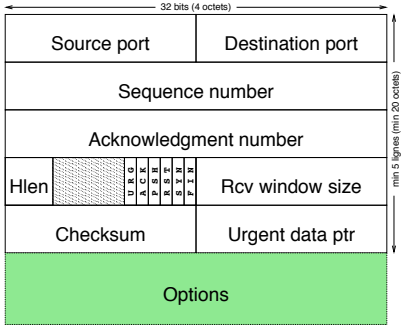
[3] Somme binaire sur 16 bits avec report de la retenue débordante ajoutée au bit de poid faible

TCP : Pointeur sur les données urgentes



- 16 bits
- permet l'envoi de données spéciales (et non **hors bande**)
- délimite des données traitées en priorité
- indique la fin des données urgentes
  - ✓ interprétation de la quantité de données et de leur rôle par l'application

TCP : Options



Les options sont de la forme TLV ou *Type, Length* (octets), *Value* :

- `END` : fin de liste d'options (T=0, non obligatoire)
- `NOOP` : pas d'opération (T=1, bourrage)
- `MSS` : négociation du MSS (T=2, L=4, V=MSS)
- `WSIZE` : mise à l'échelle de la fenêtre par le facteur 2^N (T=3, L=3, V=N)
- `SACK` : demande d'acquittement sélectif (T=4,L=2, à l'ouverture)
- `SACK` : acquittement sélectif de n blocs (T=5, L=$2+8n$, $2n$ numéros de séquences)
- ...

Plan

TCP : Gestion de la connexion

Ourverture (création) de la **connexion** préalable à l'échange des données :

- initialisation des variables TCP
  - ✓ synchronisation des numéros de séquence
  - ✓ création des tampons
  - ✓ initiation du controle de flot
- **client** : initiateur de la connexion
- **serveur** : en attente de la demande de connexion

Fermeture (terminaison) de la connexion après l'échange des données :

- attente ou non de l'émission des données restantes
- libération des tampons

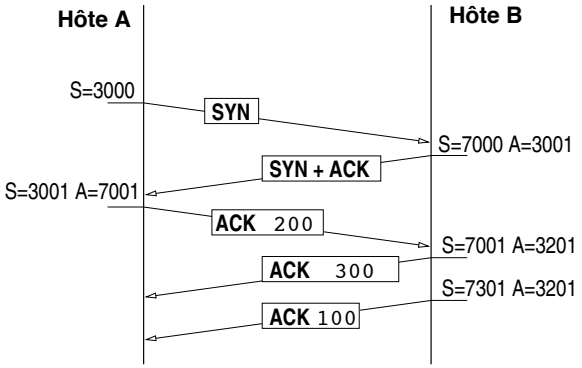
TCP : *Three-Way Handshake* (1)

Echange initial en 3 segments *(Three-Way Handshake)*

- **1** client ➠ serveur : segment TCP avec le bit `SYN`
  - ✓ indique le numéro de séquence initial (ISN) choisi par le client
  - ✓ l'emission du `SYN` incrémentera le futur numéro de séquence
  - ✓ pas de données
- **2** serveur ➠ client : segment TCP avec les bits `SYN` + `ACK`
  - ✓ la réception du `SYN` à incrémenté le numéro de d'aquittement
  - ✓ indique le numéro de séquence initial (ISN) choisi par le serveur
  - ✓ l'emission du `SYN` incrémentera le futur numéro de séquence
  - ✓ allocation des tampons du serveur
- **3** client ➠ serveur : segment TCP avec le bit `ACK`
  - ✓ la réception du `SYN` à incrémenté le numéro de d'aquittement
  - ✓ peut contenir des données

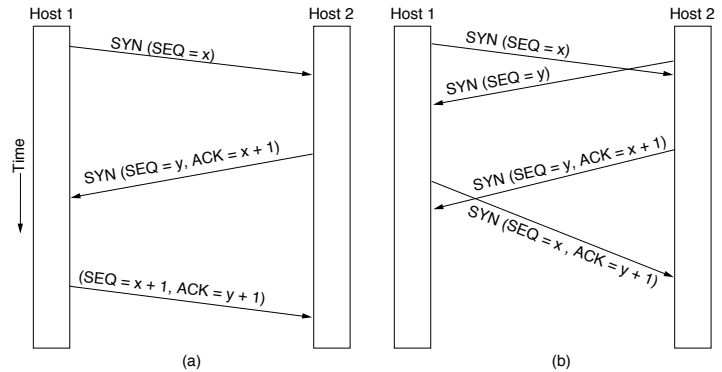
TCP : *Three-Way Handshake* (2)

Echange initial en 3 segments



TCP : *Three-Way Handshake* (3)

Gestion des ouvertures simultanées



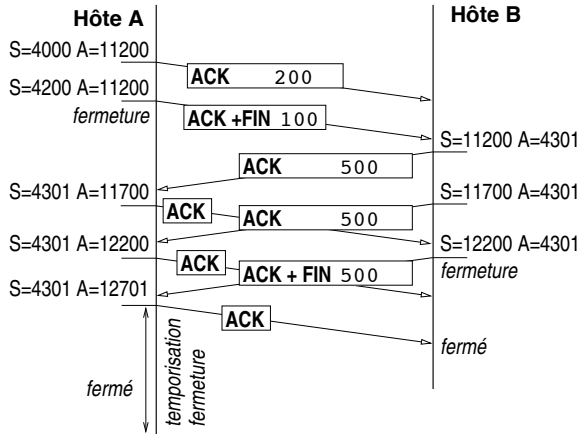
pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

TCP : *Gracefull Release* (1)

- 1 le **client** émet un segment TCP avec FIN
  - ✓ l'émission du FIN incrémentera le futur numéro de séquence
  - ✓ peut contenir des données
- 2 le **serveur** recoit le segment avec FIN
  - ✓ la réception du FIN incrémente le numéro d'aquittement
  - ✓ émet un segment TCP avec ACK
  - ✓ termine la connexion **(envoie les données restantes)**
  - ✓ émet un segment TCP avec FIN
  - ✓ l'émission du FIN incrémentera le futur numéro de séquence
- 3 le **client** recoit le segment avec FIN
  - ✓ la réception du FIN incrémente le numéro d'aquittement
  - ✓ emet un segment TCP avec ACK
  - ✓ termine la connexion
    - ☞ déclanche une temporisation d'attente (FIN dupliquées)
- 4 le **serveur** recoit le segment avec FIN

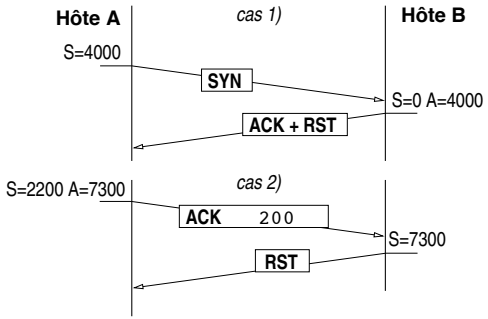
TCP : *Gracefull Release* (2)

Déconnexion : terminaison courtoise

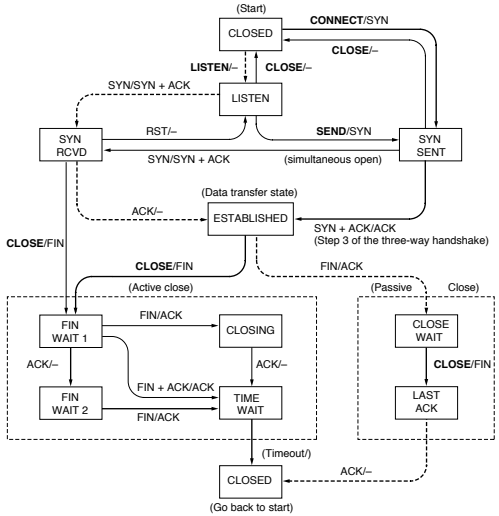


TCP : *Shutdown*

Déconnexion : terminaison unilatérale
(pour tout comportement anormal ou indésiré)



TCP : Automate d'états finis



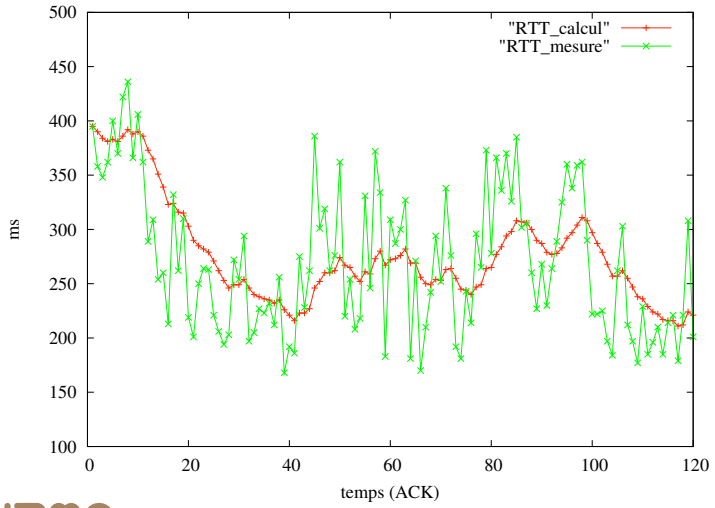
Plan

TCP : Calcul du RTT

Round Trip Time

- Estimation de la temporisation de retransmission :
  - ✓ supérieure au RTT... mais le RTT varie !
    - ☞ trop petit : retransmissions inutiles
    - ☞ trop grand : réaction lente aux pertes
- Estimation du RTT :
  - ✓ $RTT_{mesure} = \Delta$ (envoi segment, reception ACK correspondant)
  - ✓ RTT_{mesure} peut varier rapidement ➠ lissage
    - ☞ $RTT = \alpha RTT_{mesure} + (1 - \alpha) RTT_{ancien}$
      avec α usuel $= 1/8$
  - ✓ moyenne glissante à décroissance exponentielle

TCP : Exemple de calcul de RTT



TCP : Temporisations

Gestion de multiples temporisations (*timers*) :

- *retransmission timer* (détecte les pertes)
  - ✓ $RTO = RTT + \delta D$
    - ☞ avec $\delta = 4$ et une valeur initiale du RTT élevée (3 secondes)
  - ✓ $D = \beta(|RTT_{mesure} - RTT_{ancien}|) + (1 - \beta)D_{ancien}$
    - ☞ calcul de l'écart moyen avec β usuel $= 1/4$
  - ✓ **algorithme de Karn**
    - ☞ ne pas tenir compte des paquets retransmis et doubler le RTO à chaque échec (*exponential backoff*)
- *persistence timer* (évite les blocages)
  - ✓ envoi d'un acquittement avec une fenêtre à 0
- *keep alive timer* (vérifie s'il y a toujours un destinataire)
- *closing timer* (terminaison)

Plan

Transmission fiable de TCP

TCP est un protocole fiable de transfert sur le service IP non fiable

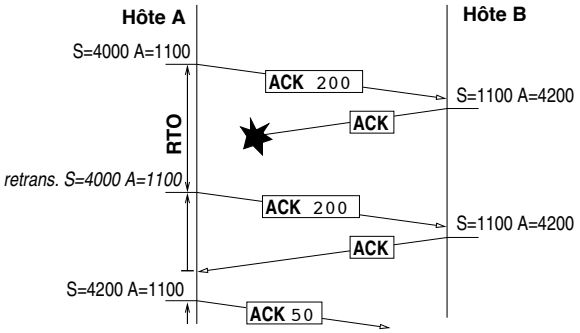
- mécanismes de base :
  - ✓ **pipeline**
  - ✓ ACK **cumulatifs**
  - ✓ temporisateur de retransmission **unique**
  - ✓ retransmissions déclanchées par :
    - ☞ expiration de temporisation *(timeout)*
    - ☞ duplication d'ACK
- dans la suite...
  - ✓ émetteur TCP simplifié :
    - ☞ pas d'ACK dupliqué
    - ☞ pas de contrôle de flux
    - ☞ pas de contrôle de congestion

Évènements émetteur TCP

- **réception des données de la couche supérieure**
  - ✓ **création** d'un segment avec `numSeq`
    - ☞ `numSeq` est le numéro dans le flux d'octet du premier octet de donnée du segment
  - ✓ démarrer la **temporisation** si elle n'est pas déjà en cours
    - ☞ la temporisation correspond au segment non acquitté le plus ancien
- **expiration de temporisation** *(timeout)*
  - ✓ **retransmission** du segment associé à la temporisation
  - ✓ redémarrer la **temporisation**
- **réception d'aquittement** (ACK)
  - ✓ si acquitte des segments non acquités :
    - ☞ actualiser la base de la fenêtre de transmission (`base_emis`)
    - ☞ redémarrer la **temporisation** si d'autres ACK sont attendus

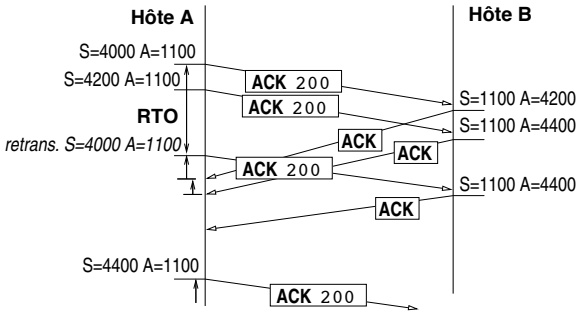
Retransmission TCP (1)

Scénario avec ACK perdu



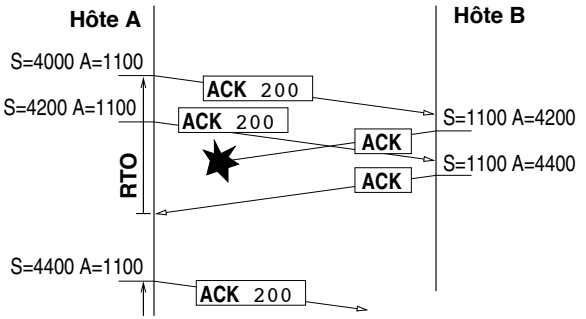
Retransmission TCP (2)

Scénario avec temporisation sous-estimée



Retransmission TCP (3)

Scénario avec ACK cumulatifs



Évènement récepteur TCP

Génération d'**ACKs** (actions du récepteur)

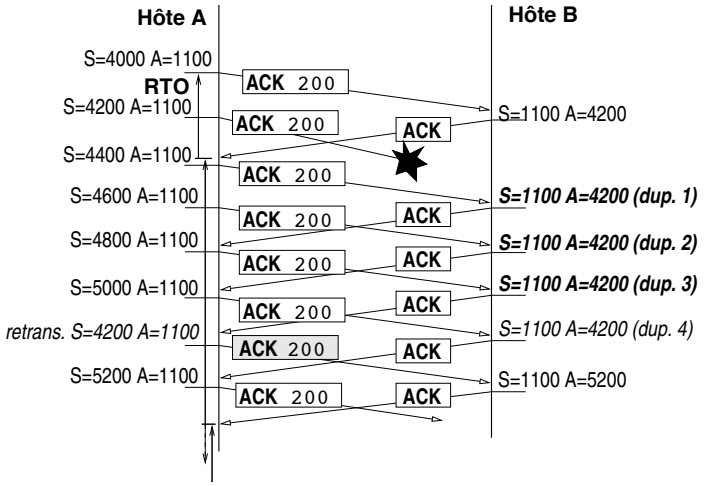
- arrivée d'un segment **dans l'ordre** avec le `numSeq` attendu :
  - ✓ les segments précédents sont déjà acquittés
    - ☞ ACK **retardé** *(delayed ACK)*, attente jusqu'à 500 ms
    - ☞ si pas d'autre segments, envoi d'un ACK
  - ✓ un autre segment est en attente d'acquittement
    - ☞ envoi immédiat d'un ACK **cumulatif** pour ces deux segments dans l'ordre
- arrivée d'un segment **dans le désordre** :
  - ✓ `numSeq` supérieur à celui attendu (intervalle détecté)
    - ☞ envoi immédiat d'un ACK **dupliqué**
    - ☞ rappel du prochain `numSeq` attendu
  - ✓ rempli partiellement ou totalement un intervalle
    - ☞ envoi immédiat d'un ACK
    - ☞ nouveau `numSeq` attendu suite au remplissage de l'intervalle

TCP : *Fast Retransmit* (1)

Optimisation du mécanisme de retransmission

- temporisation souvent relativement élevée
  - ✓ délai important avant une retransmission
- détection des segments perdus grâce aux ACKs **dupliqués**
  - ✓ ensemble de segments souvents envoyés cote-à-cote
  - ✓ si un segment est perdu ➠ nombreux ACKs dupliqués
- si l'émetteur reçoit 3 ACK dupliqués (4 ACKs identiques)
  - ✓ TCP suppose que le segment suivant celui acquité est perdu
    - ☞ **fast retransmit** : retransmission du segment avant l'expiration de la temporisation

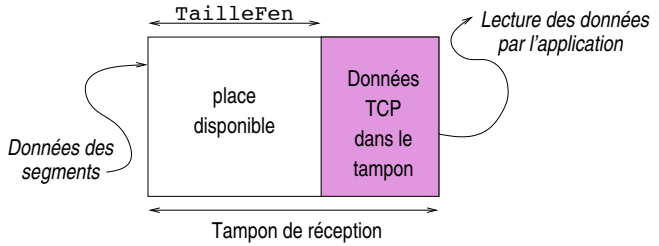
TCP : *Fast Retransmit* (2)



Plan

TCP : Asservissement au récepteur

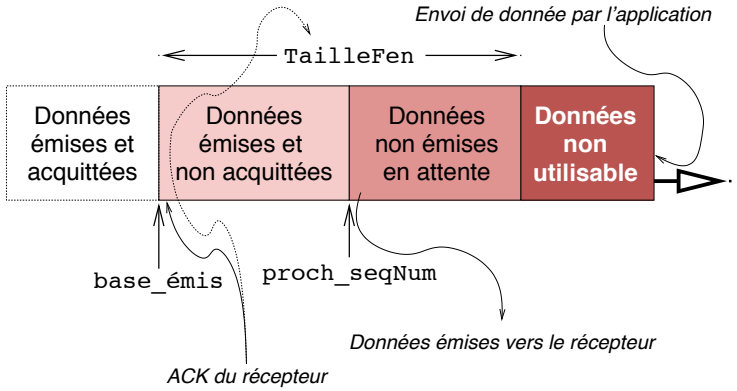
- **contrôle de flux**
  - ✓ l'émetteur ne doit pas dépasser les capacités du récepteur
  - ✓ récupération de la taille de la place disponible du tampon de réception du récepteur :



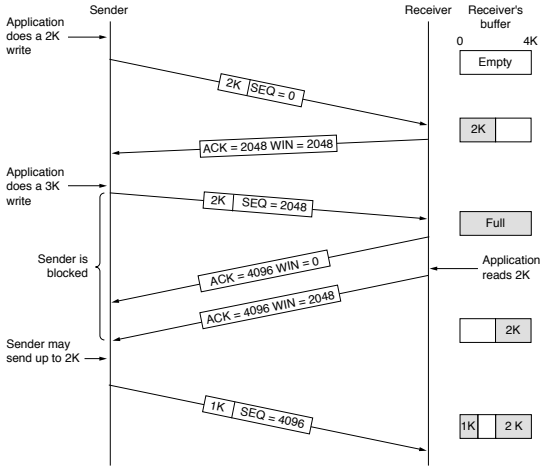
  - ✓ `TailleFen = TailleTampon - DernierOctetRecu + DernierOctetLu`

TCP : Limitation de l'émetteur

Sliding window : l'emetteur limite la transmission de ses données non acquittées



TCP : Contrôle de flux



pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

TCP : temporisation de ré-ouverture de la fenêtre

Persistence timer

- évite que la taille de la fenêtre reste à 0
  - ✓ possible si perte du ACK annonçant une fenêtre non nulle
  - ✓ évité grâce à l'envoi d'un paquet sonde après une temporisation
    - ☞ tempo. initiée à RTT puis double à chaque expiration jusqu'à 60s (puis reste 0 60s)
    - ☞ le paquet sonde est un segment avec 1 octet de données

TCP : Optimisation du contrôle de flux

Send-side silly window syndrome

- Algorithme de Nagle (RFC 896)
  - ✓ agrégation de petits paquets *(nagling)*
  - ✓ attente d'un acquittement ou d'un MSS avant d'envoyer un segment
    - ☞ TELNET : évite d'envoyer un paquet par caractère tapé
    - ☞ désactivable avec l'option TCP_NODELAY des sockets

Receiver silly window syndrome

- Algorithme de Clark
- limiter les annonces de fenêtre trop petites
  - ✓ fermeture de la fenêtre en attendant d'avoir suffisamment de place pour un segment complet

Plan

TCP : exemples d'applications

Les applications suivantes reposent typiquement sur TCP :
- connexion à distance (TELNET, rlogin et ssh)
- transfert de fichiers (FTP, rcp, scp et sftp)
- protocole de routage externe (BGP)
- messageries instantanées (IRC, ICQ, AIM...)
- web (HTTP)
  - ✓ nouvelles applications utilisent HTTP comme service d'accès au réseau
    - ☞ permet de passer les *firewall*

TCP : utilisation spécifiques

TCP doit s'adapter à des flots de qqs **bps** à plusieurs **Gbps** :

- LFN (*Long Fat Network*)
  - ✓ capacité du réseau = **bande passante * délai de propagation**
    - ☞ limitation de taille de la fenêtre (option `WSIZE`, jusqu'à un facteur 2^{14})
    - ☞ rebouclage des numéros de séquence (PAWS, *Protect Against Wrapped Sequence*, utilise l'option `TIMESTAMP`)
    - ☞ acquittements sélectifs pour éviter des retransmissions importantes inutiles (option `SACK`)
  - ✓ satellites
  - ✓ fibres transatlantiques
- réseaux asymétriques (ADSL, Cable)
  - ✓ sous-utilisation du lien rapide

TCP : Interface socket

```
#include <sys/types.h>
#include <sys/socket.h>

# create a descriptor and bind local IP and port
int socket(int domain, int type, int protocol);
#   domain : PF_INET for IPv4 Internet Protocols
#     type : SOCK_STREAM Provides sequenced, reliable, 2-way, connection-based byte streams.
#            An out-of-band data transmission mechanism may be supported.
# protocol : TCP (/etc/protocols)
int bind(int s, struct sockaddr *my_addr, socklen_t addrlen);

# Server : passive queuing mode and connection acceptance
int listen(int s, int backlog);
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);

# Client : active connection
int  connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

# Send and receive data
int send(int s, const void *msg, size_t len, int flags);
int recv(int s, void *buf, size_t len, int flags);

# End : dealocate
int close(int s);
```

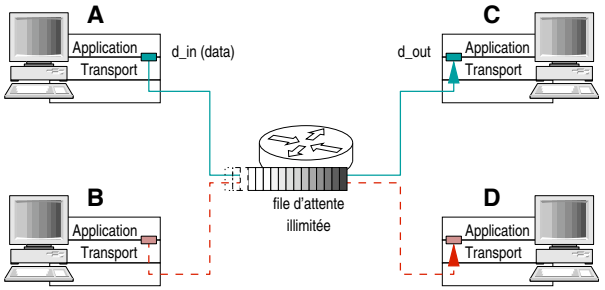

Plan

Contrôle de congestion

Congestion

- trop de flots de données saturent un ou plusieurs éléments du réseau
- différent du contrôle de flux
  - ✓ TCP n'a pas accès à l'intérieur du réseau
- manifestation :
  - ✓ longs délais
    - ☞ attente dans les tampons des routeurs
  - ✓ pertes de paquets
    - ☞ saturation des tampons des routeurs

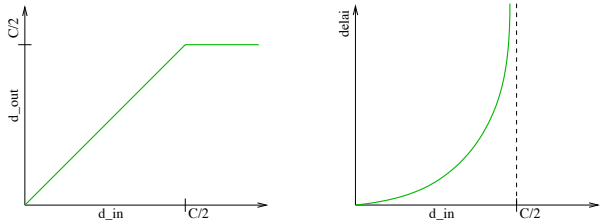
Congestion : scénario 1a



- 2 émetteurs, 2 récepteurs
- 1 routeur
  - ✓ tampons infinis
- pas de retransmission

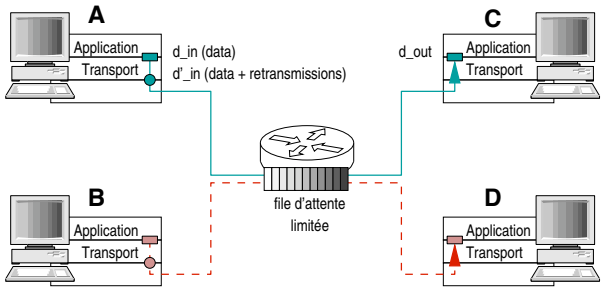
➠ *Que ce passe-t-il quand d_in augmente ?*

Congestion : scénario 1b



- **coût de la congestion :**
  - ✓ débit maximum atteignable
    - ☞ d_in = C/2
  - ✓ delai très élévé proche du maximum
    - ☞ croissance infinie des tampons

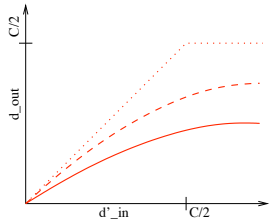
Congestion : scénario 2a



- 2 émetteurs, 2 récepteurs
- 1 routeur
  - ✓ **tampons infinis**
- **retransmission** des segments perdus

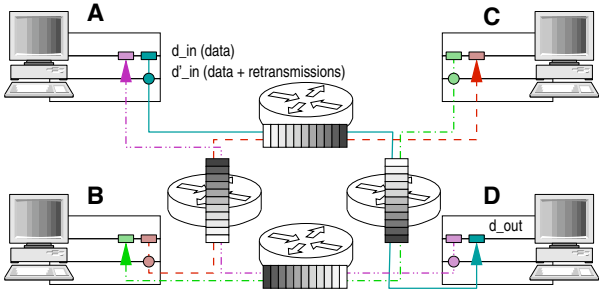
➠ *Que ce passe-t-il quand d'_in augmente ?*

Congestion : scénario 2b



- toujours d_in = d_out *(goodput)*
- coût des retransmissions
  - ✓ **retransmissions utiles** : seulement pour des pertes
    - ☞ d'_in supérieur à d_out
  - ✓ **retransmissions inutiles** : segments en retard
    - ☞ d'_in encore plus supérieur à d_out
- **coût de la congestion** :
  - ✓ beaucoup plus de trafic pour un d_out donné
  - ✓ dupplications de segment inutile

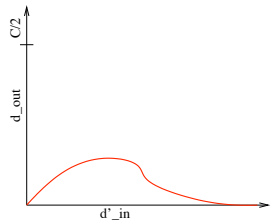
Congestion : scénario 3a



- 4 émetteurs, 4 récepteurs
- 4 routeurs
  - ✓ chemins multi-saut
  - ✓ tampons finis
- retransmission

➠ *Que ce passe-t-il quand d'_in augmente ?*

Congestion : scénario 3b



- **coût supplémentaire de la congestion** :
  - ✓ lors de la perte d'un paquet, toute la capacité amont est gachée

Contrôle de congestion

Deux approches :

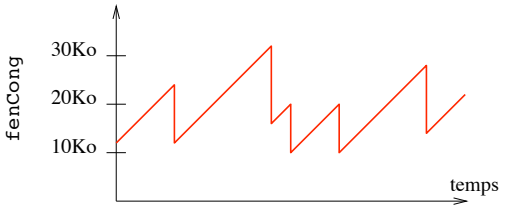
- contrôle de congestion géré par le **réseau**
  - ✓ les routeurs informent les extrémités
    - ☞ bit d'indication de la congestion (SNA, DECbit, ATM, TCP/IP ECN...)
    - ☞ indication explicte du débit disponible (ATM ABR, TCP/IP RSVP + IntServ...)
- contrôle de congestion aux **extrémités** *(end-to-end)*
  - ✓ aucune indication explicite du réseau
  - ✓ **inférence** à partir des observations faites aux extrémités
    - ☞ pertes
    - ☞ délais
  - ✓ **approche choisie dans TCP**

Plan

TCP : Contrôle AIMD

Additive Increase, Multiplicative Decrease

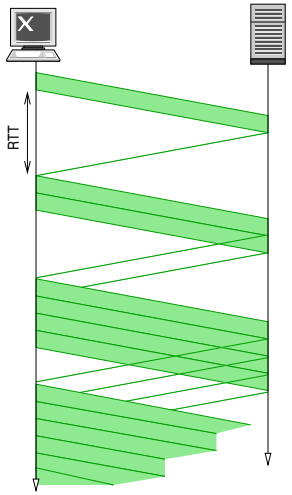
- augmentation progressive du débit de transmission (`fenCong`) tant qu'il n'y a pas de perte
  - ✓ *Additive Increase*
    - ☞ augmenter `fenCong` de 1 MSS à chaque RTT tant qu'il n'y a pas de perte détectée
  - ✓ *Multiplicative Decrease*
    - ☞ diviser `fenCong` par 2 après une perte
  - ✓ comportement en dent de scie :



TCP : Contrôle de congestion

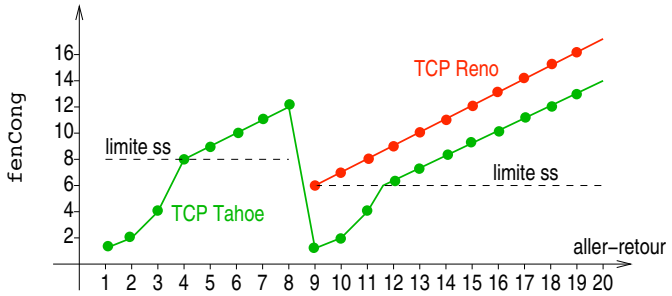
- basé sur la limitation de l'émission de l'émetteur
  - ✓ `dernierOctetEmis - dernierOctetAcq` $\leq$ `fenCong`
  - ✓ approximation du débit :
    - ☞ $d_{TCP} = \frac{\texttt{fenCong}}{RTT}$
- `fenCong` = fonction dynamique de la congestion perçue
  - ✓ perception de la congestion par le récepteur :
    - ☞ expiration de temporisation (RTO)
    - ☞ triple ACK
  - ✓ 3 mécanismes :
    - ☞ AIMD
    - ☞ *Slow Start*
    - ☞ prudence après l'expiration d'une temporisation

TCP : Slow Start



- démarre lentement *(slow start)*
  - ✓ mais croit très vite ! !
- au démarrage de la connexion
  - ✓ `fenCong` = 1 MSS
    - ☞ $d_{init} = \frac{MSS}{RTT}$
  - ✓ croissance exponentielle jusqu'à la première perte
    - ☞ `fenCong` double / RTT
    - ☞ implémenté par : `fenCong` ++ / ACK
    - ☞ $d_{potentiel} \gg \frac{MSS}{RTT}$

TCP : Optimisation



- passage de la croissance exponentielle à lineaire
  - ✓ `fenCong` $\geq$ ancienne valeur de `fenCong` juste avant la perte
    - ☞ implémenté par une limite variable : `limiteSS` = $\texttt{fenCong}_{avant\ la\ derniere\ perte}/2$

`limiteSS` est précisément calculé avec les segments non acquittés *(flightsize)*/2

TCP : Inférence des pertes

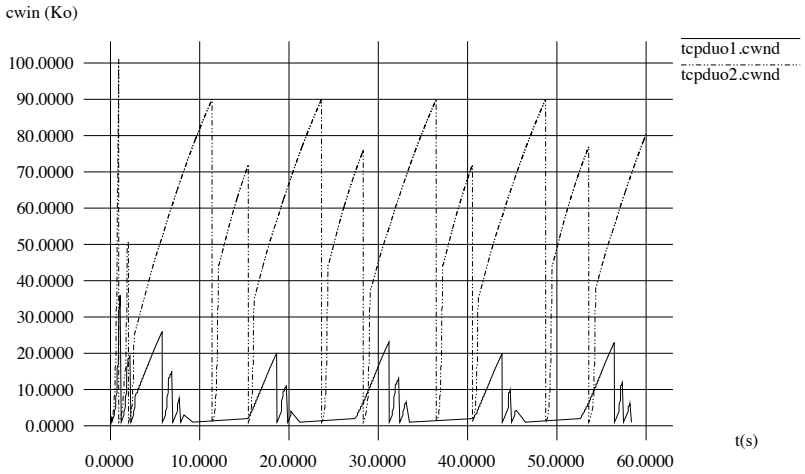
Les ACK dupliqués sont moins graves que les expirations de temporisation

- suite **3 ACK dupliqués** :
  - ✓ indique que le réseau continue à transmettre des segments
    - ☞ `fenCong` divisé par 2
    - ☞ `fenCong` croit ensuite linéairement
- suite **expiration temporisation** :
  - ✓ indique que le réseau se bloque
    - ☞ `fenCong` = 1 MSS
    - ☞ *Slow Start* (croissance exponentielle)
    - ☞ à `limiteSS` = `fenCong`/2 (croissance linéaire)

Contrôle de congestion TCP : synthèse

- quand `fenCong` $<$ `limiteSS` :
  - ✓ émetteur en phase *Slow Start*
  - ✓ `fenCong` croit exponentiellement
- quand `fenCong` $\geq$ `limiteSS` :
  - ✓ émetteur en phase *Congestion Avoidance*
  - ✓ `fenCong` croit linéairement
- quand 3 ACK dupliqués apparaissent :
  - ✓ `limiteSS` = dernière `fenCong` / 2
  - ✓ `fenCong` = `limiteSS`
- quand la temporisation expire :
  - ✓ `limiteSS` = dernière `fenCong` / 2
  - ✓ `fenCong` = 1 MSS

TCP : équité entre flots ?



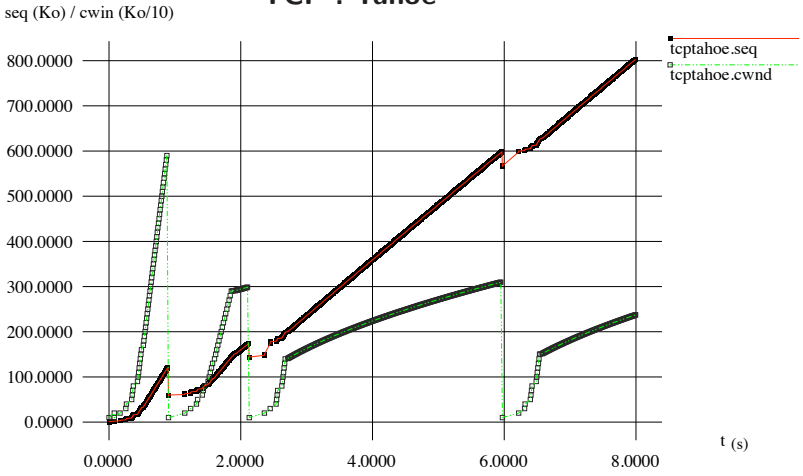
- oscillation de deux flots en phase de congestion

Implémentations

A trip to Nevada :

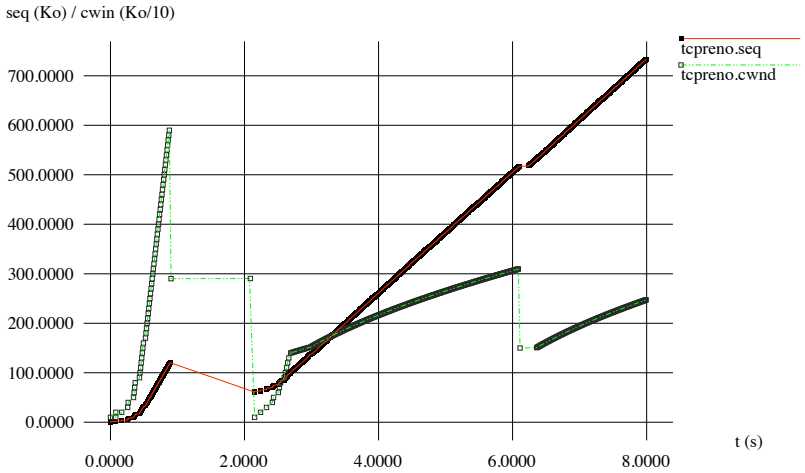
- **TCP Tahoe** 1988
  - ✓ *slow start* + *congestion avoidance* + *multiplicative decrease*
  - ✓ + **fast retransmit** (déclenche la retransmission d'un segment après trois acquittements dupliqués, avant l'expiration de la temporisation)
  - ✓ décrit précédement... problème quand juste un segment est perdu
- **TCP Reno** 1990 (RFC 2581)
  - ✓ idem TCP Tahoe
  - ✓ + **fast recovery** (pas de *slow start* après un *fast retransmit*)
- **TCP newReno** 1996 (RFC 3782)
  - ✓ idem TCP Reno
  - ✓ + pas de *slow start* à la première congestion et ajustement de `fenCong`
  - ✓ + SACK (RFC 2018)
- **TCP Vegas...**
  - ✓ évite la congestion en **anticipant** les pertes
  - ✓ réduction du débit en fonction des variations du RTT

TCP : Tahoe



- *slow start* + *congestion avoidance* + *multiplicative decrease*
- + **fast retransmit** : vers quel débit converge-t-on ?

TCP : Reno



- **fast recovery** (pas de *slow start* après un *fast retransmit*)

Fin

Document réalisé avec LaTeX.
Classe de document `foils`.
Dessins réalisés avec `xfig`.

Olivier Fourmaux, `olivier.fourmaux@upmc.fr`
`http://www-rp.lip6.fr/~fourmaux`

Ce document est disponible en format PDF sur le site :
`http://www-master.ufr-info-p6.jussieu.fr/`