Faculté des Sciences & Techniques

Université de Limoges

Master 1ère année

"On the Internet, nobody knows you're a dog."

Protocoles et Programmation réseau — P-F. Bonnefoi

Version du 3 décembre 2023

Table des matières

Le réseau : différents points de vue

- **vision programmeur** : un système de «couches OSI, *Open Systems Interconnection*» :
  - de la couche **physique**, n°1, à la couche **applicative** n°7 ;
  - des **protocoles de communications** à maîtriser : TCP, UDP, SMTP, POP, SSH *etc.* ;
- **vision humaine et géographique** :
  - des échanges **au sein de la même structure** «humaine» (entreprise, université, *etc*) :
    - matériels administrés par la **même autorité** : segmentation du réseau, DNS «Domain Name System» local, *etc.*
    - demande de **performances** de haut niveau : QoS, partage d'accès à des ressources, applications distribuées, *etc.*
  - des échanges entre **réseaux répartis** sur la planète :
    - de **l'organisation** : organisme internationaux, DNS, *etc.*
    - des **réseaux interconnectés** : INTERconnected NETworks : du routage de haut niveau avec BGP, *etc.*
- **vision théorique** :
  - réseau «point à point» *vs* «diffusion» ;
  - des **mesures** : latence, gigue, débit, bande passante, *etc.*
  - des **principes** : contrôle de flux, contrôle de congestion, contrôle d'erreur, *etc.*
- **vision sécurité** :
  - **surveillance** du réseau : détection et identification du trafic, protection contre les attaques ;
  - **filtrage** des échanges ;
  - mise en place de **tunnels** d'échanges sécurisés ;
- **vision concepts fondamentaux** :
  - adressage : niveau 2, IPv4, IPv6, VLAN, MPLS ;
  - *switching* ou commutation : sélectionner une sortie pour un paquet en entrée ;
  - *forwarding* ou relayage : sélectionner *intelligemment* une sortie pour un paquet en entrée ;
  - *routing* ou routage : construire une route pour l'acheminement d'un paquet.

Plan de la partie Protocoles

Éléments importants

- ▷ Notion de couches et de processus pairs ;
- ▷ Notion de service et d'interface ;
- ▷ Modèle OSI et pile de protocoles TCP/IP ;
- ▷ Modélisation de protocole ;
- ▷ Problème de synchronisation et de programmation.

Organisation en série de couches

But **réduire la complexité** de conception.

Les réseaux sont organisés en série de couches ou niveaux, chacune étant construite sur la précédente.

Rôle d'une couche offrir certains services aux couches plus hautes, en leur masquant l'implémentation de ces services.

Relation entre couches sur différentes machines

La couche n d'une machine gère **la conversation** avec la couche n d'une autre machine.

Notion de «protocole»

Les **règles** et **conventions** utilisées pour cette conversation sont connues sous le nom de «protocole de la couche n» :

- ▷ un **protocole** est un accord entre les parties sur la **façon** de communiquer ;
- ▷ toute **violation** du protocole rend la communication extrêmement difficile voire **impossible**.

Notion de «processus pairs»

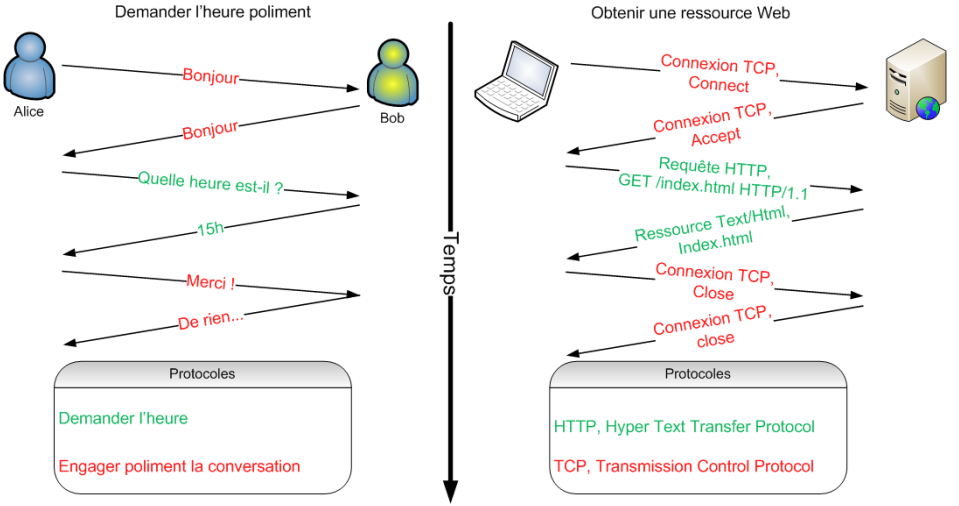
Les couches correspondantes sur différentes machines sont appelés **processus pairs**, *peer*.

Ce sont les processus pairs qui **communiquent** à l'aide du protocole.

En réalité, **aucune donnée** *ne passe directement de la couche n d'une machine à la couche n d'une autre machine, mais chaque couche passe par les données et les contrôle à la* **couche qui lui est immédiatement inférieure***.*

Un protocole humain et un protocole machine

« demander l'heure à quelqu'un » et « demander une ressource sur un serveur Web ».



Les protocoles définissent :

- le **format** des données échangées ;
- l'**ordre** des messages émis et reçus entre les entités réseaux;
- ainsi que les **réactions** à ces messages.

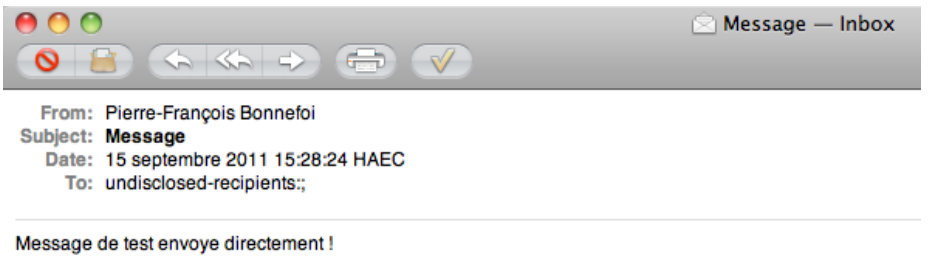
Un protocole correspond à un **comportement** qui **évolue** en fonction des données échangées.

Le protocole SMTP, «Simple Mail Transfer Protocol»

xterm

```
bonnefoi@msi:~$ socat - tcp:smtp.unilim.fr:25
220 smtp.unilim.fr ESMTP Sendmail 8.13.1/8.13.1; Thu, 15 Sep 2011 15:28:24 +0200
HELO msi.unilim.fr
250 smtp.unilim.fr Hello www.msi.unilim.fr [164.81.60.6], pleased to meet you
MAIL FROM: <bonnefoi@unilim.fr>
250 2.1.0 <bonnefoi@unilim.fr>... Sender ok
RCPT TO: <bonnefoi@unilim.fr>
250 2.1.5 <bonnefoi@unilim.fr>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Message

Message de test envoye directement !
.
250 2.0.0 p8FDSOJe031646 Message accepted for delivery
QUIT
221 2.0.0 smtp.unilim.fr closing connection
bonnefoi@msi:~$
```

La commande «`socat`» permet de simplement établir une connexion TCP avec le serveur que l'on a désigné sur le port indiqué.



Le protocole HTTP, «*Hyper Text Transfer Protocol*»

xterm

```
pef@darkstar-8:/Users/pef  socat - tcp:www.unilim.fr:80
HEAD / HTTP/1.0

HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Mon, 11 Sep 2017 10:53:33 GMT
Content-Type: text/html
Content-Length: 178
Connection: close
Location: http://www.cryptis.fr/
```

Le protocole POP, «*Post Office Protocol*»

xterm

```
bonnefoi@msi:~$ socat - tcp:pop.unilim.fr:110
+OK courriel Cyrus POP3 v2.2.13-Debian-2.2.13-14.xm.1 server ready <299345444.1316380363@courriel>
USER bonnefoi
+OK Name is a valid mailbox
PASS bob
-ERR [AUTH] Invalid login
^C
bonnefoi@msi:~$
```

Ce protocole est utilisé pour consulter son courrier et le récupérer dans son logiciel de messagerie.

On lui préfère le protocole IMAP, port 143 en version non sécurisée, qui permet de consulter son courrier tout en le laissant sur le serveur.

Notion d'interface :
entre chaque couche adjacente existe une interface.
L'interface définit :

- ⋆ les **opérations élémentaires**, appelées «primitives» ;
- ⋆ les **services** que la couche inférieure offre à la couche supérieure.

La définition des interfaces doit être **claire** :

- ◦ chaque couche réalise un ensemble de fonctions bien définies ;
- ◦ le changement d'implémentation d'une couche est transparent : *il suffit à la nouvelle implémentation d'offrir exactement à sa voisine du dessus le même ensemble de services que l'ancienne*
  Exemple : changement d'une carte réseau, passage d'une connexion filaire à du sans-fil…

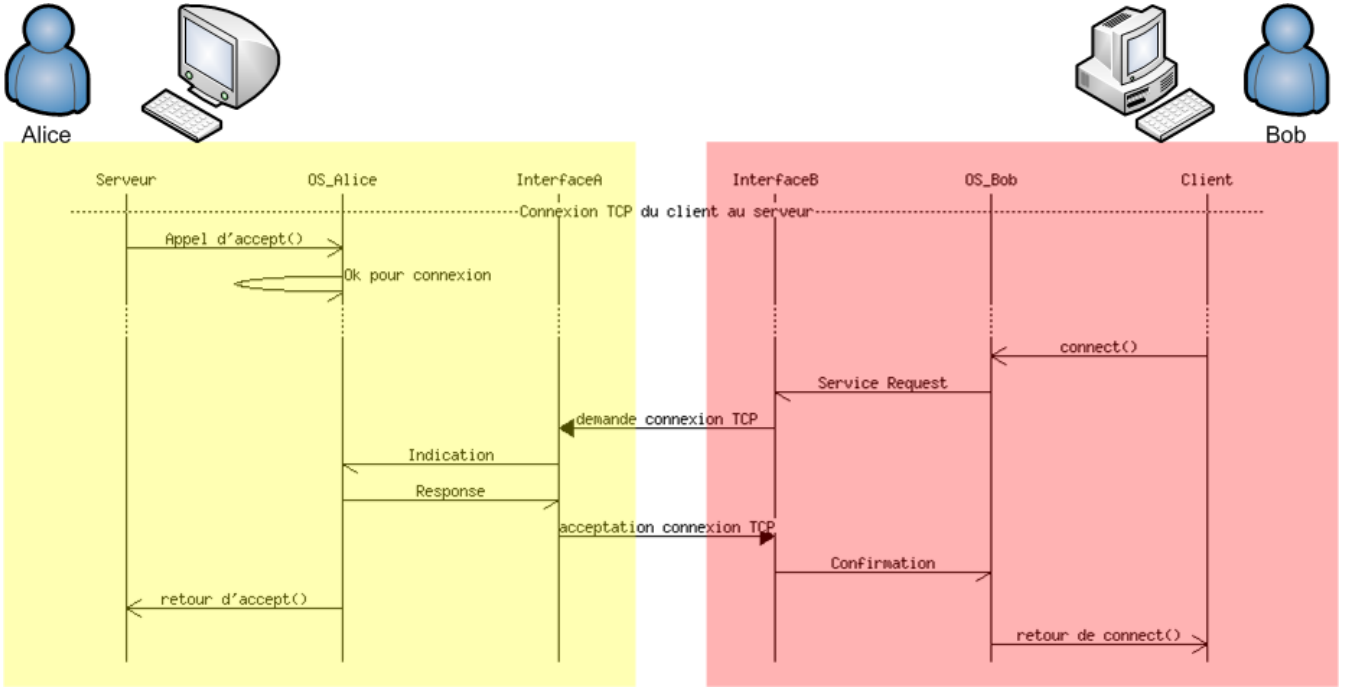
L'ensemble des couches et protocoles est appelé **architecture réseau.**

Au niveau de la conception du réseau :

- ▷ les **spécifications** de l'architecture doivent contenir suffisamment d'information pour permettre d'écrire le logiciel ou de construire le matériel pour chaque couche de manière qu'il obéisse correctement au protocole approprié ;
- ▷ ni les détails de mise en œuvre, ni les spécifications de l'**interface** ne font partie de l'architecture puisqu'ils sont invisibles à l'extérieur.
- ▷ il n'est **pas nécessaire** que les interfaces de toutes les machines du réseau soient identiques, pourvu que chaque machine puisse utiliser correctement les protocoles (windows et GNU/Linux par exemple).

L'**ensemble des protocoles** utilisés par un système, avec un protocole par couche, est appelé souvent **«pile de protocoles»**.

a. Le serveur indique qu'il est prêt et d'accord pour établir une connexion à l'aide de l'instruction **accept** ;
b. le client demande l'accès au service de connexion au travers d'une instruction **connect** :
   - ce connect réalise la primitive de service : requête ;
   - des paquets sont échangés à travers les interfaces;



c. la réception de ces paquets sur le serveur déclenche une indication sur l'OS d'Alice qui possède déjà l'autorisation d'accepter la connexion, *etc.*

Deux concepts importants

a. Relation entre communication virtuelle et effective
b. Différence entre protocoles et interfaces

Exemple

les processus pairs de la couche 5 conçoivent leur communication de façon horizontale, grâce au protocole fournis par la couche 4 :

- chacun des processus utilise :
  - une procédure appelée «envoi à l'autre côté»,
  - une procédure «réception de l'autre côté»,
- **en réalité** : ces procédures communiquent avec les couches inférieures par l'intermédiaire de l'interface 3/4, *et non, comme elles en donnent l'impression, avec l'autre côté !*

Le **concept abstrait de processus pair** est crucial pour la conception des réseaux.

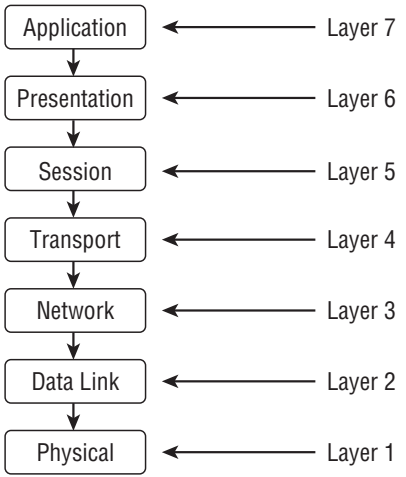
Cette technique d'abstraction permet de passer :

- d'un **problème insoluble** : la conception d'un réseau global,
- à **plusieurs problèmes solubles** : la conception de chaque couche.

L'ensemble des couches a été normalisées par l'ISO en 7 couches.
Ce modèle a été appelé OSI (Open System Interconnection).

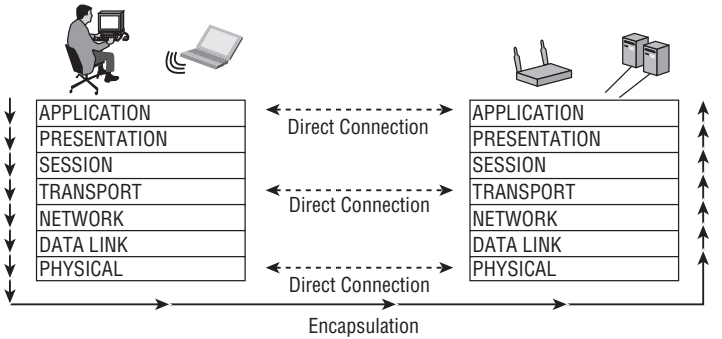
Il existe aussi celui, plus simple, de la **pile TCP/IP.**

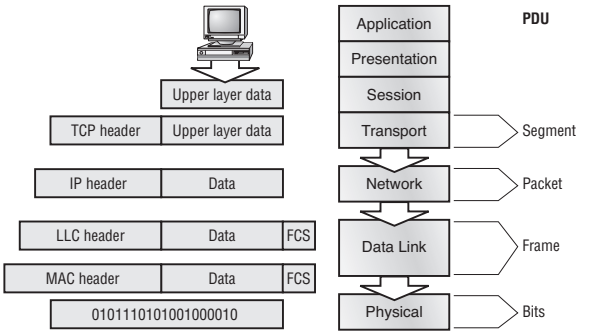
Seul la pile TCP/IP a été implémentée.

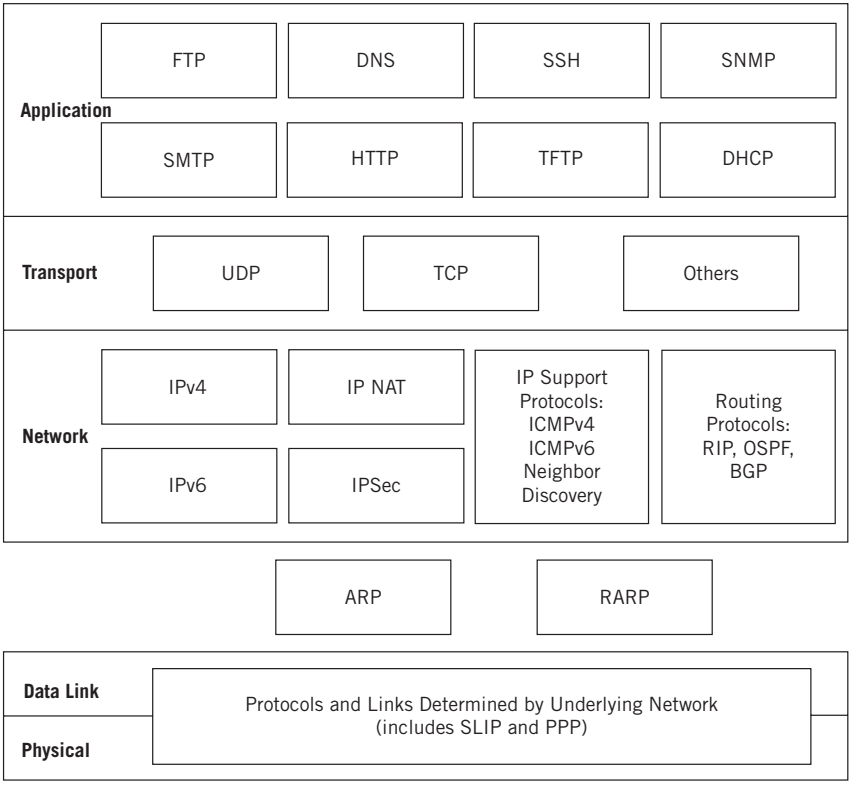


- sert de référence pour décrire le fonctionnement du réseau :
  - 5 couches présentes dans TCP/IP : Physique, Liaison de données, Réseau, Transport, Application ;
  - 2 couches «applicatives» dans TCP/IP : Session, Présentation ;
- chaque couche réalise un travail distinct avec des «moyens» interchangeables (protocoles, technologies) :
  - **Physique** : technologie WiFi, Ethernet, Bluetooth, *etc.*
  - **Liaison de données** : trame 802.3/Ethernet II ;
  - **Réseau** : datagramme IPv4, IPv6 ;
  - **Transport** : UDP, «*User Datagram Protocol*», TCP, «*Transmission Control Protocol*», SCTP, «*Stream Control Transmission Protocol*», *etc.*
  - **Application** : navigateur Web, logiciel de messagerie. *etc.*
  - **Présentation** : encodage des données, représentation ;
  - **Session** : ouverture/fermeture de session, identification des utilisateurs…

Communication «Pair à Pair» et encapsulation

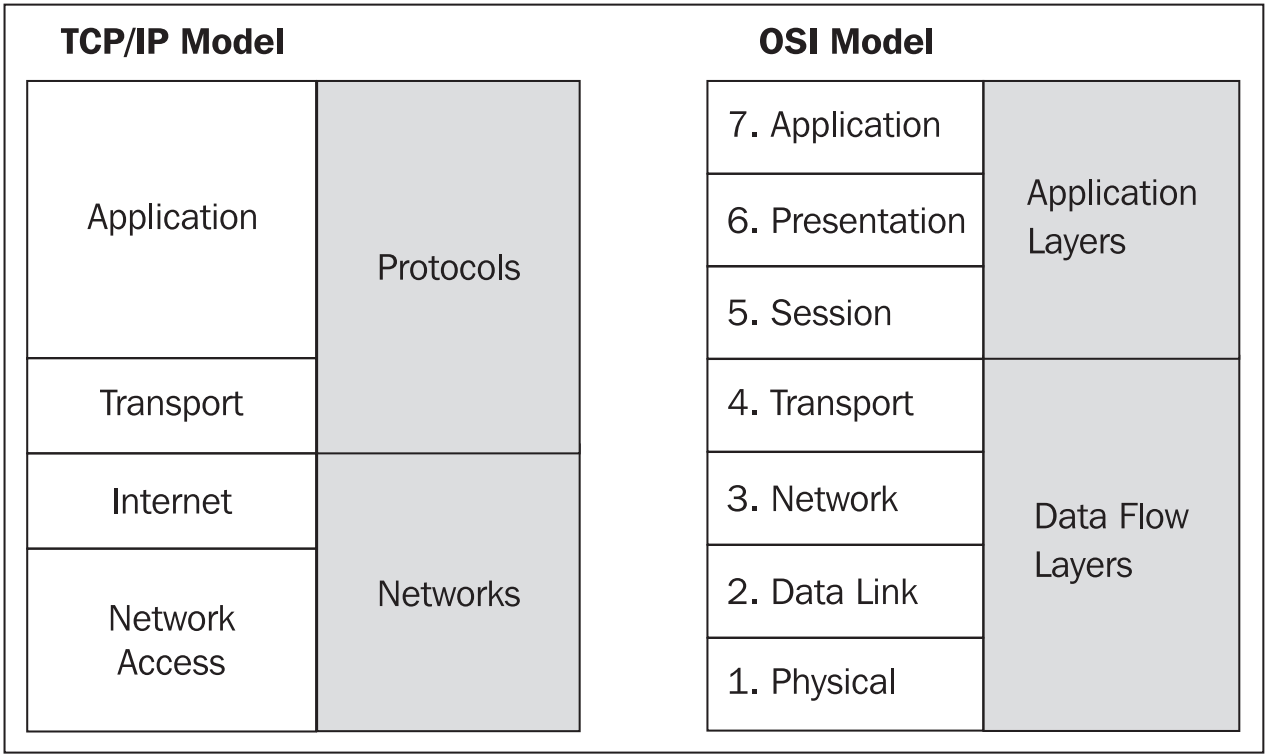




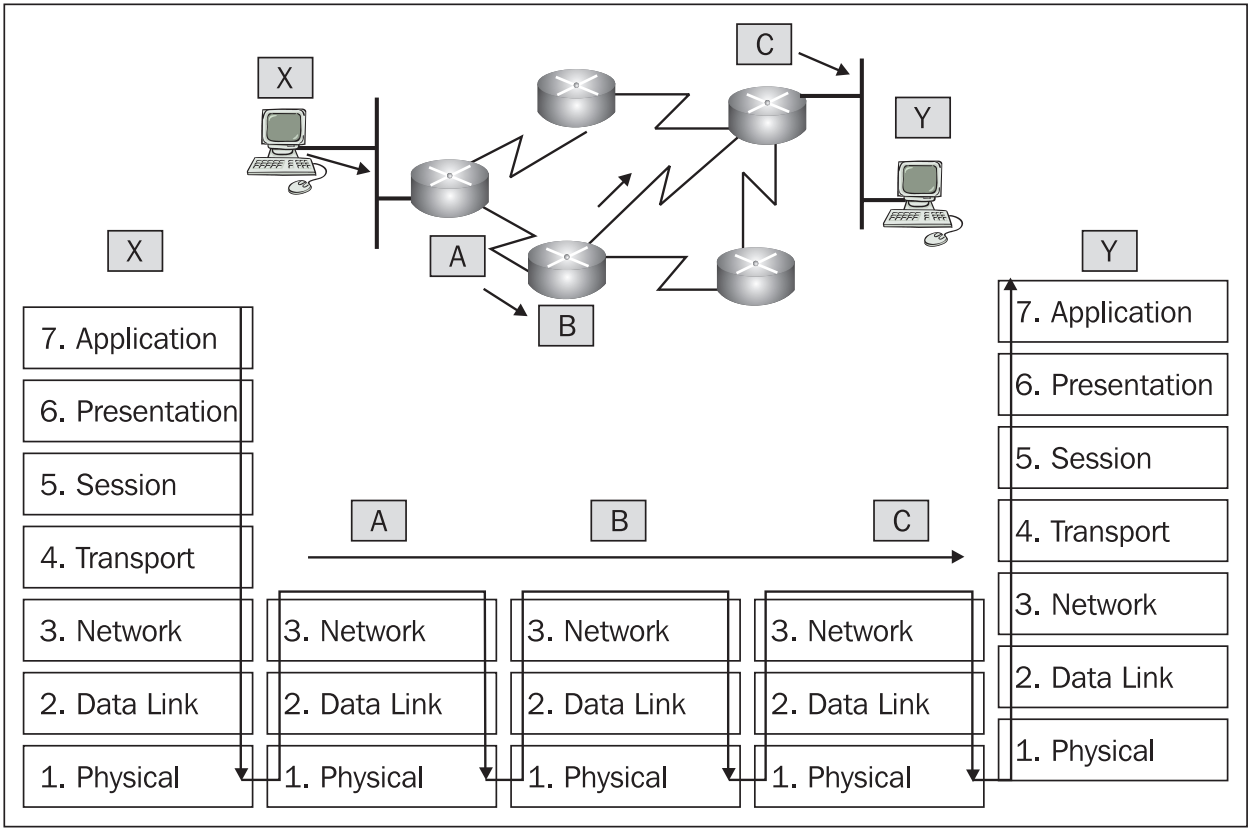


Attention :

- ⋆ certains protocoles comme RIP, OSPF, BGP utilisent des protocoles de niveau 4, «Transport».
- ⋆ le protocole ARP est entre les couches 2 et 3.



Pas de couche «Présentation» et de couche «session» dans TCP/IP par rapport à OSI.



Chaque matériel dispose d'une pile TCP/IP, plus ou moins complète.

Protocoles de haut niveau

Les protocoles dans le monde TCP/IP sont décrits dans les RFC «*Request For Comment*».
RFC 1945 pour HTTP, RFC 821 pour SMTP et 1032 pour DNS, *etc.*
Un protocole est la définition de règles pour la communication entre deux entités paires.
Il est défini par :

- un certain nombre de primitives (listen, socket, bind…),
- la définition d'un ordre d'échange d'utilisation de ces primitives (bind doit être utilisé après socket),
- la définition d'un ensemble de question/réponse attendu (un accept réussi après un connect)

Toute violation du protocole entraîne l'échec de la communication, mais peut également bloquer l'émetteur et/ou le récepteur.

Il est nécessaire de faciliter :

- la **conception** : définir les primitives, l'ordre d'échange et les interactions entre émetteur et récepteur ;
- la **validation** : vérifier qu'il fonctionne en permettant un dialogue tel qu'il a été décidé ;
- la **correction** : faire en sorte qu'il soit possible de sortir ou d'éviter une situation de blocage ;

La solution :
a. Élaboration de scenarii pour définir le protocole ;
b. Modélisation du fonctionnement du protocole ;
c. Obtention d'une spécification formelle du protocole.

Les moyens : Utilisation d'automate fini et de réseaux de Pétri.

Organisation des échanges

- ⋆ choix du modèle de conception générale :
  centralisé ou répartie, « client/serveur » ou « égal à égal »
- ⋆ choix du type de communication :
  orienté connexion ou datagramme, TCP ou UDP

Comportement du serveur si nécessaire

- ◦ gestion d'un seul client à la fois ;
- ◦ gestion de plusieurs clients simultanément ;

Durée du déroulement du protocole

- ⋆ le protocole est limité à une seule transaction ;
- ⋆ le protocole peut s'étendre sur plusieurs transactions :
  Comment mémoriser les étapes du protocole ?

Imaginer des scénarii d'usage du protocole

- ◦ dérouler le fonctionnement suivant les différents comportements prévus ;
- ◦ en cas d'erreur de l'interlocuteur, du réseau (pertes de messages, arrivée des données dans le désordre, *etc.*) ;

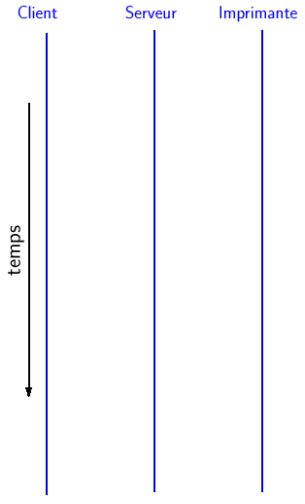
Formaliser le protocole

- ⋆ définir le format des échanges ;
- ⋆ combiner les différents scénarii pour définir le comportement complet ; choisir le comportement pour la fiabilité du protocole (éviter les blocages, garantir la disponibilité, se protéger des comportements malveillants, *etc.*).

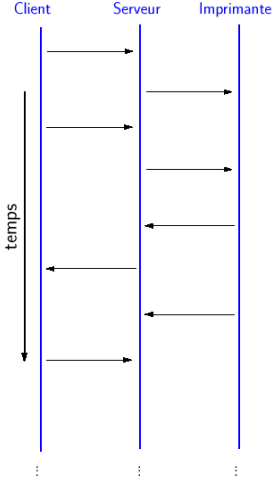
Modélisation d'une transaction : «*Message Sequence Charts*»

- ⋆ utilisés pour modéliser les échanges au cours du temps entre un nombre fini de processus.
- ⋆ permettent de définir des «scenarii» pour détailler le comportement d'un protocole sur des exemples de cas concrets.

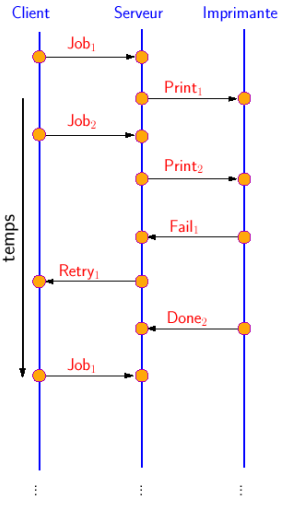
les différents acteurs



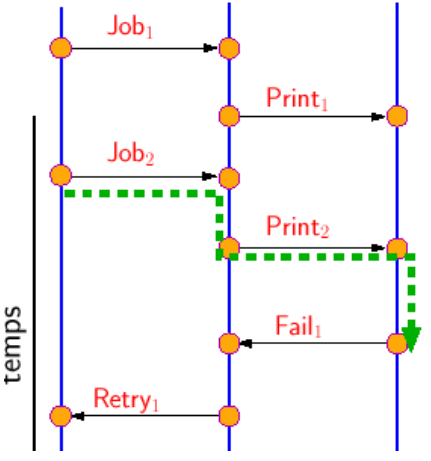
un scenario d'échange



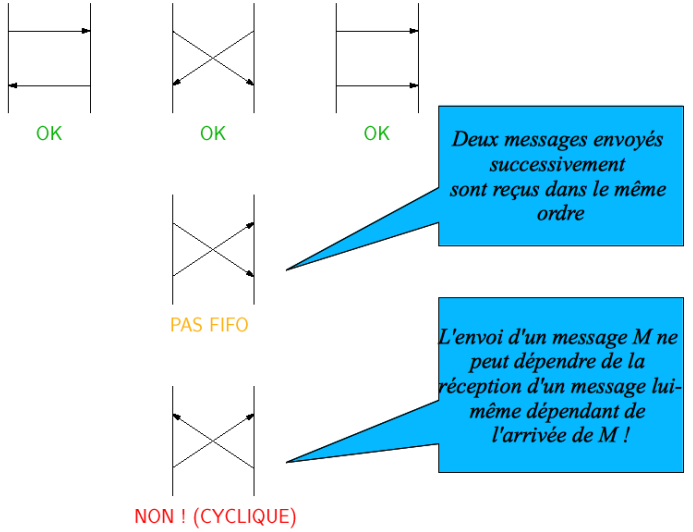
le protocole



une transaction complète



- ⋆ des événements (en orange) : réception et émission placé sur chaque processus ;
- ⋆ des échanges : des flêches ;
- ⋆ le contenu des messages : étiquette sur les flêches.
- ⋆ un ordre sur les échanges ;
- ⋆ le MSC permet de :
  - ◇ définir des **exigences** ;
  - ◇ détecter les **mauvais comportements**.



Limitations

* modélisation limitée à une transaction ou à une session complète (plusieurs transactions la suite) ;
  Pour décrire tout le protocole, il faut envisager de nombreux scenarii, voire tous les scenarii possibles!

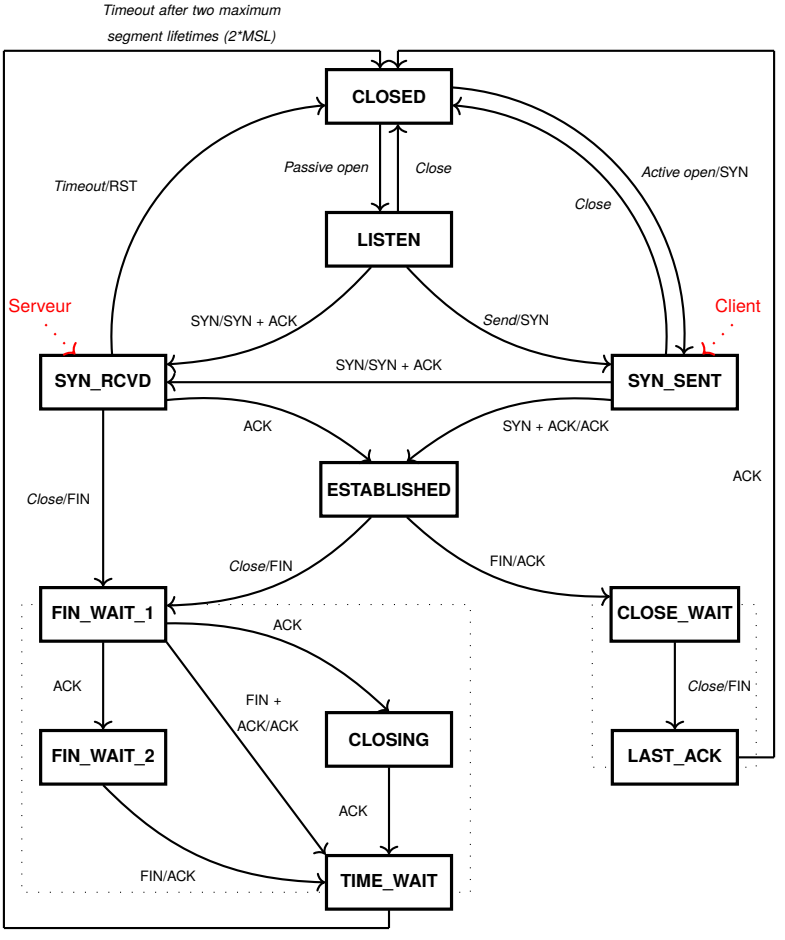
Lorsque plusieurs échanges sont nécessaires pour une même transaction ou lorsque plusieurs transactions sont nécessaires pour une même session :

– nécessité de mémoriser «où on en est» par rapport à ces différents échanges/transactions :
  ◇ chaque mémorisation peut modifier le scenarii : reprendre une transaction peut être impossible à partir d'un certain temps, une session peut s'interrompre automatiquement au bout d'un certain temps, etc.
  ◇ pour chaque mémorisation il faudrait reprendre et définir tous les scenarii...

Un scenario décrit un cas d'usage ou «*use case*» : il sert à illustrer, pour une situation donnée comment le protocole va se comporter.

Pour décrire complètement le protocole il faut un outil permettant de décrire son « comportement » global en tenant compte des « mémorisations » possibles : la modélisation par automate à nombre fini d'états.

La modélisation d'un protocole avec un automate fini



- ⋆ **protocole «texte»** : échange de lignes de commandes au format ASCII 7bits ;
- ⋆ utilise le protocole de transport entre le client et le serveur de type **TCP**.
- ⋆ **très simple**, ce qui explique sa popularité et sa facilité de mise en oeuvre.

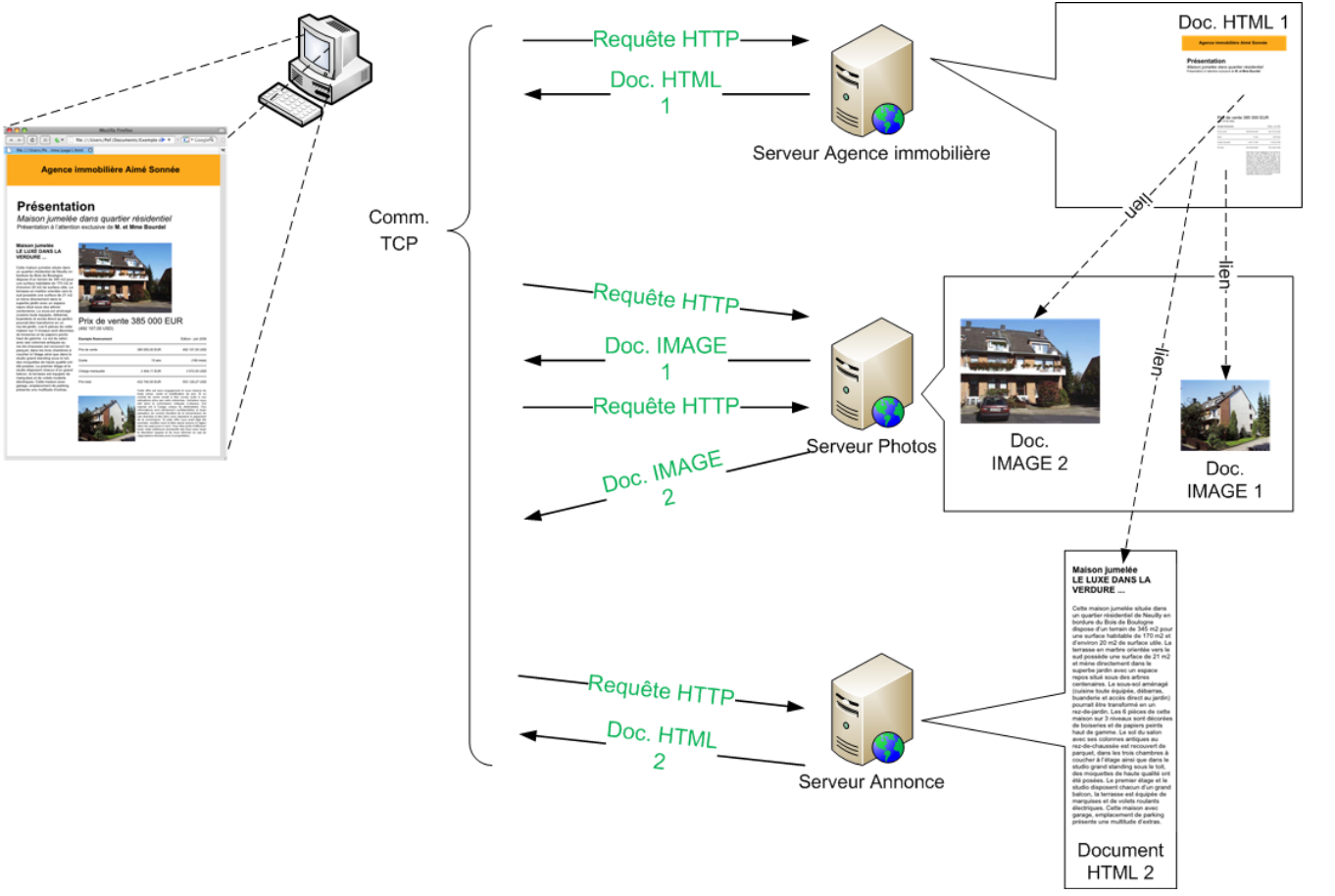
Différentes versions :

- ◦ HTTP/0.9 version de base avec requête/réponse le document est renvoyé directement ;
- ◦ HTTP/1.0 version normalisée (RFC 1945) avec comme amélioration, l'ajout d'en-tête pour la description des ressources échangées (utilisation du format MIME RFC822), d'informations supplémentaires envoyées par le client (format de données supporté ou désiré, description de la version et de la marque du navigateur...)
- ◦ HTTP/1.1 ajout de connexions persistantes entre le client et le serveur en vue de l'échange de plusieurs ressources par l'intermédiaire de la même connexion (transfert des différents éléments d'un même document composite).
- ◦ HTTP/2 les requêtes et les réponses peuvent être fragmentées et ces fragments peuvent être envoyés dans un ordre quelconque pour accélérer leur prise en charge par le navigateur
- ◦ HTTP/3 *aka* QUIC, «*Quick UDP Internet Connections*», basé sur UDP, combine des éléments de TCP/TLS/HTTP2

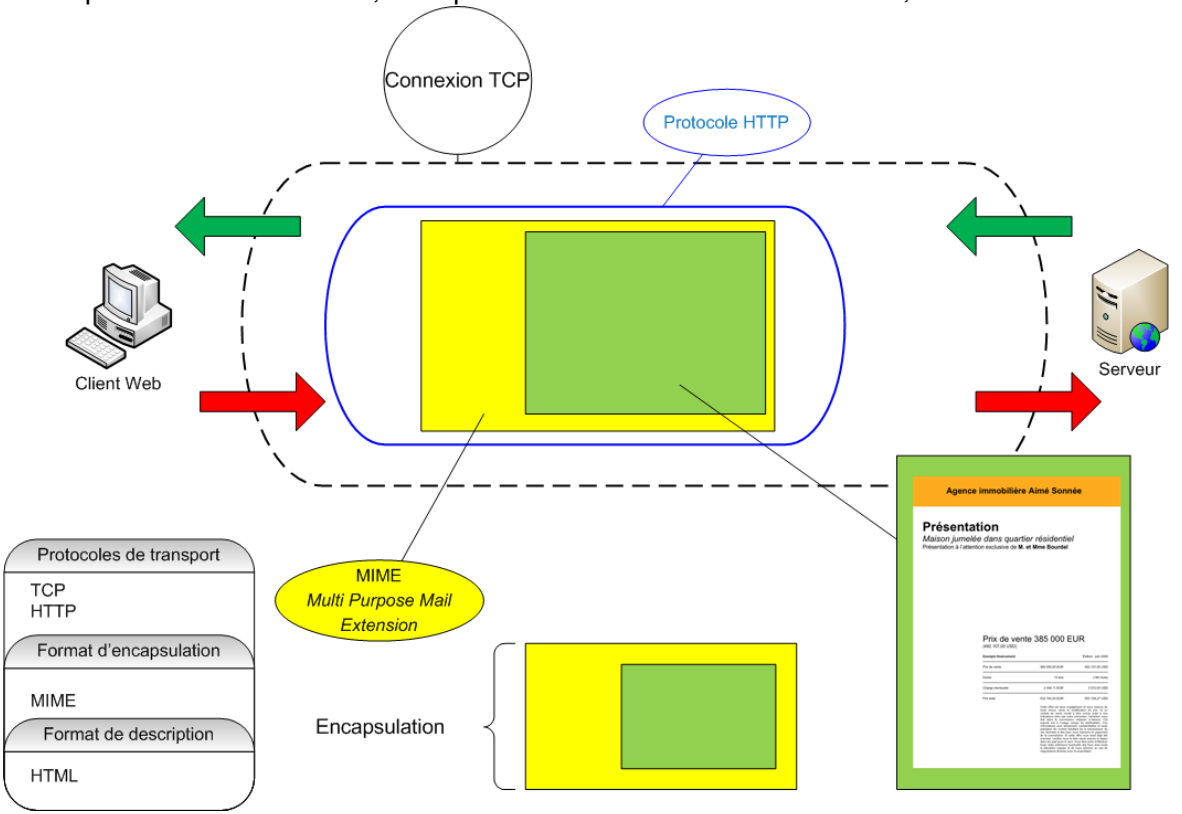
xterm

```
pef@darkstar:~$ socat stdio tcp:p-fb.net:80
GET /toto HTTP/1.0
Host: p-fb.net

HTTP/1.1 301 TYPO3 RealURL redirect for missing slash
Server: nginx
Date: Wed, 04 Sep 2019 21:53:27 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
Location: http://p-fb.net/toto/
```


Le protocole HTTP, «HyperText Transfer Protocol», RFC 1945

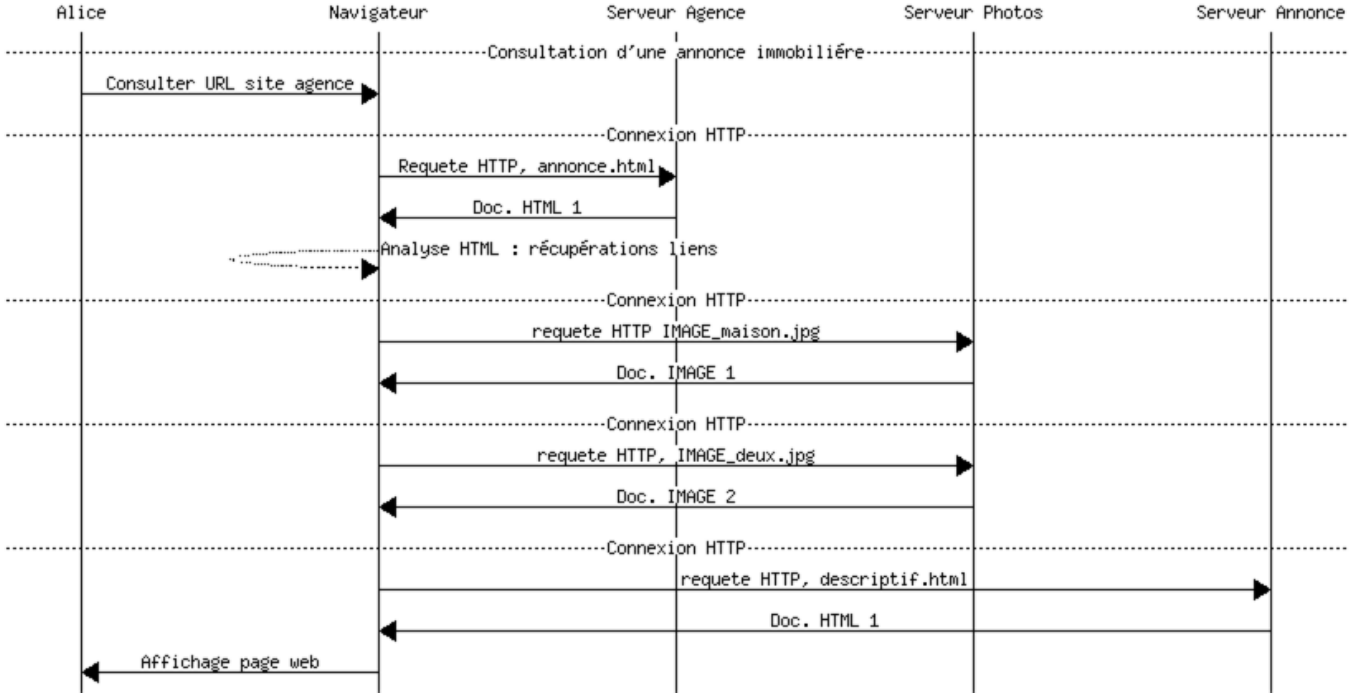


Utilisation du «mode connecté» : protocole de transport TCP, encapsulant le protocole HTTP pour échanger un descripteur de format MIME, encapsulatn un contenu formaté HTML, …



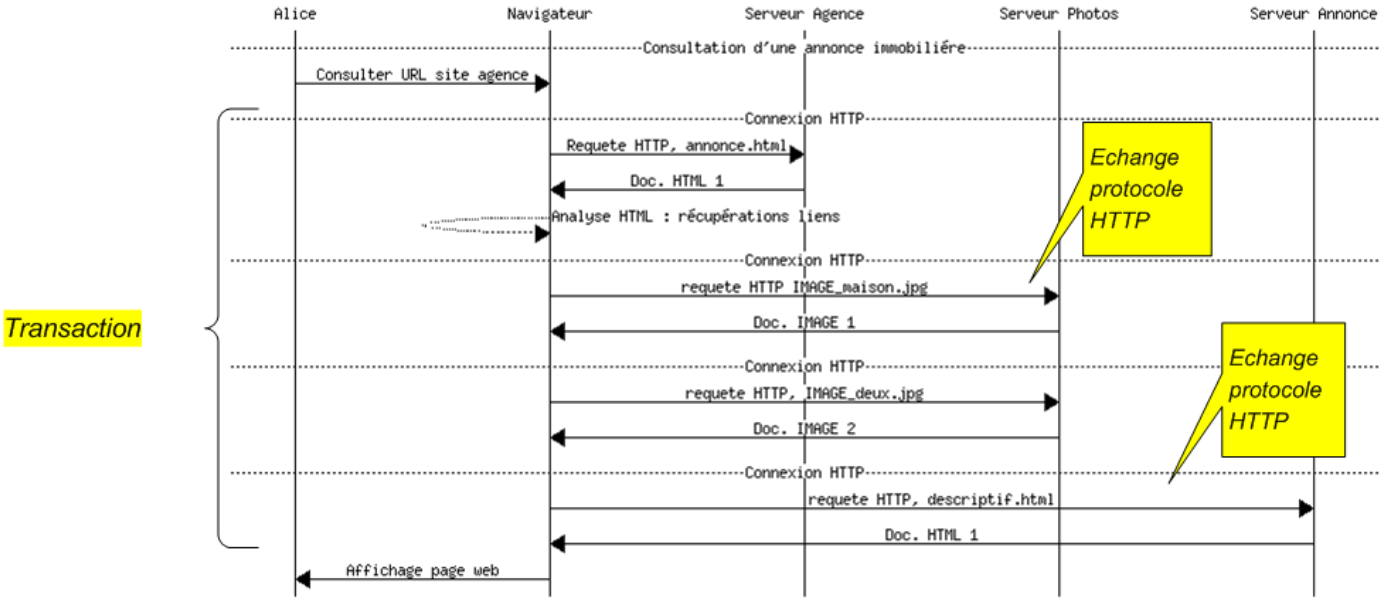
Une pile de protocole

- protocole « utilisateur » ou abstrait : consultation d'une page web ;
- protocole de transport : TCP ;
- protocole d'échange : HTTP ;
- format d'échange : MIME.



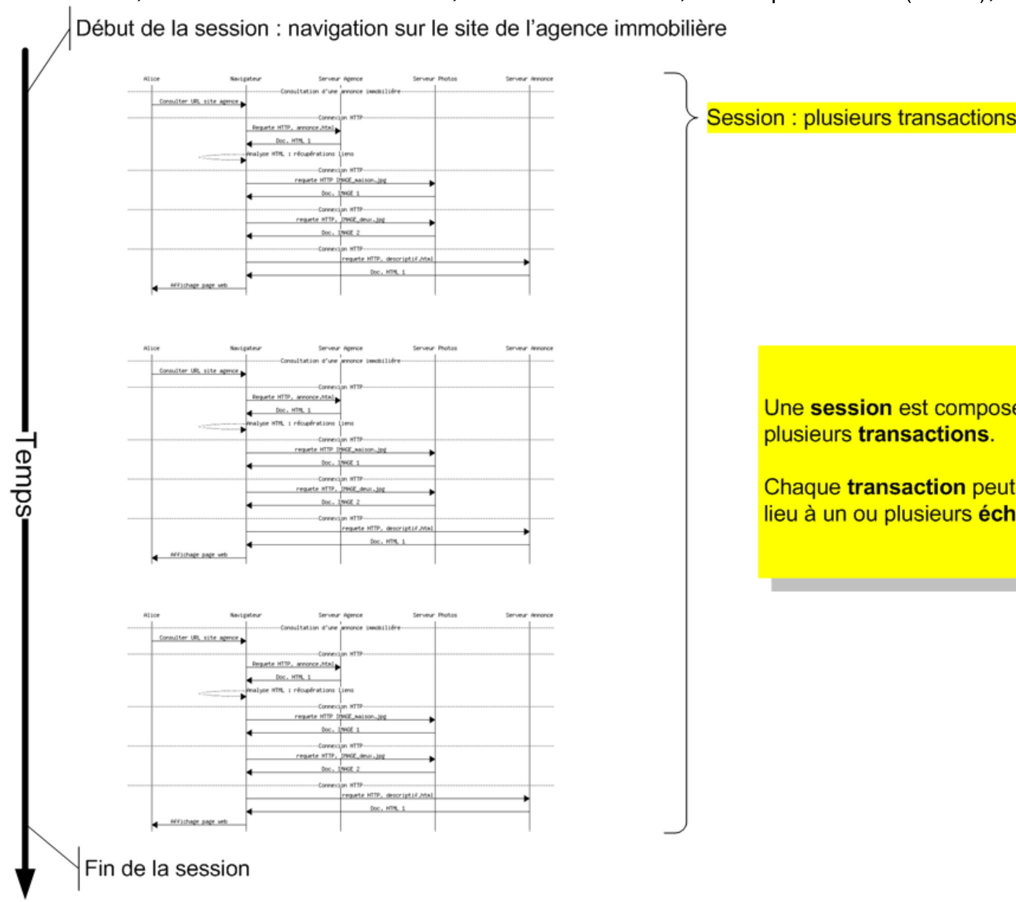
Rapport entre les différents protocoles

Le protocole abstrait est celui qui intéresse l'utilisateur (ici, Alice), c-à-d. « naviguer sur le Web ».
L'unité élémentaire de ce protocole est la transaction (Alice charge une page Web).
Le navigateur d'Alice réalise plusieurs échanges au format HTTP pour récupérer les contenus multimédia.
La notion de session décrit l'ensemble des transactions qui ont un certain lien entre elles.
Par exemple : entrer dans le magasin virtuel, s'identifier, remplir son caddie, payer et quitter le site.



Le protocole HTTP : la notion de session

Utilisation de cookies, de données de formulaires, de contenus JSON, d’URL particulière (REST), *etc.*



Une **session** est composée de plusieurs **transactions**.

Chaque **transaction** peut donner lieu à un ou plusieurs **échanges**.

La commande «`curl`» va récupérer les données au format JSON et ces données vont être formatées par le module «json.tool» :

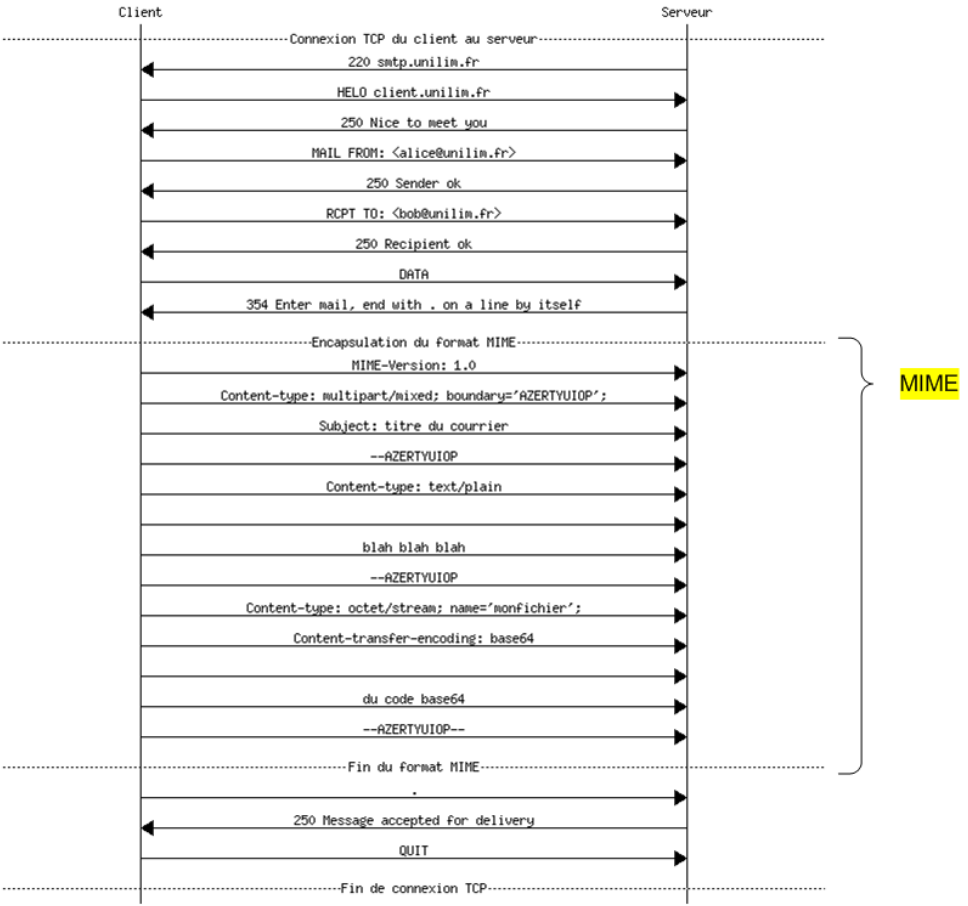
xterm

```
pef@darkstar:/Users/pef $ curl -sH 'Accept: application/json' http://api.icndb.com/jokes/random | python3 -m
json.tool
{   "type": "success",
    "value": {
        "categories": [
            "nerdy"
        ],
        "id": 543,
        "joke": "Chuck Norris's programs can pass the Turing Test by staring at the interrogator."
    }
}
        }
    ],
}
```

xterm

```
pef@darkstar-8:/Users/pef $ curl -vI http://www.unilim.fr/
*   Trying 164.81.1.97...
* TCP_NODELAY set
* Connected to www.unilim.fr (164.81.1.97) port 80 (#0)
> HEAD / HTTP/1.1
> Host: www.unilim.fr
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
HTTP/1.1 301 Moved Permanently
< Server: nginx
Server: nginx
< Date: Mon, 11 Sep 2017 10:56:46 GMT
Date: Mon, 11 Sep 2017 10:56:46 GMT
< Content-Type: text/html
Content-Type: text/html
< Connection: keep-alive
Connection: keep-alive
< Location: https://www.unilim.fr/
Location: https://www.unilim.fr/

<
* Connection #0 to host www.unilim.fr left intact
```

Le protocole SMTP



Le protocole UPnP, «Universal Plug and Play»

- HTTPU, «HTTP unicast»
- HTTPMU, «HTTP multicast»

```
0000   01 00 5E 7F FF FA B4 07  F9 F3 3A 73 08 00 45 60   ..^.......:s..E`
0010   00 F9 00 00 40 00 04 11  DA 04 A4 51 07 44 EF FF   ....@......Q.D..
0020   FF FA 07 6C 07 6C 00 E5  F1 76 4E 4F 54 49 46 59   ...l.l...vNOTIFY
0030   20 2A 20 48 54 54 50 2F  31 2E 31 0D 0A 48 6F 73    * HTTP/1.1..Hos
0040   74 3A 20 32 33 39 2E 32  35 35 2E 32 35 35 2E 32   t: 239.255.255.2
0050   35 30 3A 31 39 30 30 0D  0A 4E 54 3A 20 75 72 6E   50:1900..NT: urn
0060   3A 6E 75 6C 6C 73 6F 66  74 2E 63 6F 6D 3A 64 65   :nullsoft.com:de
0070   76 69 63 65 3A 41 6E 64  72 6F 69 64 3A 31 0D 0A   vice:Android:1..
0080   4E 54 53 3A 73 73 64 70  3A 61 6C 69 76 65 0D 0A   NTS:ssdp:alive..
0090   43 61 63 68 65 2D 43 6F  6E 74 72 6F 6C 3A 6D 61   Cache-Control:ma
00a0   78 2D 61 67 65 3D 33 30  0D 0A 4C 6F 63 61 74 69   x-age=30..Locati
00b0   6F 6E 3A 68 74 74 70 3A  2F 2F 31 36 34 2E 38 31   on:http://164.81
00c0   2E 37 2E 36 38 3A 33 34  34 38 38 0D 0A 69 64 3A   .7.68:34488..id:
00d0   39 37 37 34 64 35 36 64  36 38 32 65 35 34 39 63   9774d56d682e549c
00e0   0D 0A 6E 61 6D 65 3A 73  61 6D 73 75 6E 67 20 47   ..name:samsung G
00f0   54 2D 49 39 30 30 30 0D  0A 70 6F 72 74 3A 33 34   T-I9000..port:34
0100   34 38 38 0D 0A 0D 0A                              488
```

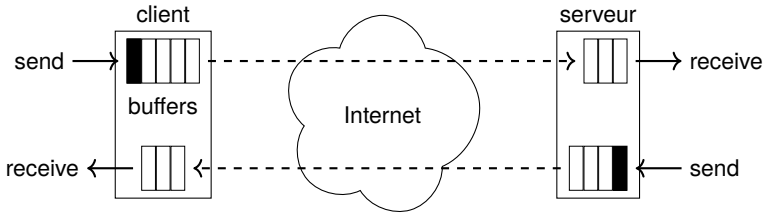
Des processus qui échangent des messages

- ▷ **synchrones**
  - ◇ l'émetteur attend que le récepteur ait reçu le message (*envoyer un fax par exemple*) ;
  - ◇ le récepteur qui attend un message est bloqué jusqu'à sa réception ;
  - ◇ **Avantage :** l'émetteur et le récepteur sont dans un état connu ;
  - ◇ **Inconvénient :** fort couplage entre les processus.
- ▷ **asynchrones**
  - ◇ l'émetteur n'est pas bloqué en attente de réception (*poster une lettre*) ;
  - ◇ le récepteur peut fonctionner suivant deux modes :
    - ⋆ réception bloquante si pas de message ;
    - ⋆ réception non bloquante avec témoin de réception (*boîte aux lettres américaine*) ;
  - ◇ **Avantage :** l'émetteur et le récepteur sont indépendants au cours du temps
  - ◇ **Inconvénients :**
    - ⋆ pas d'acquittement implicite
    - ⋆ pas de relation entre les états de l'émetteur et du récepteur
    - ⋆ difficultés en cas d'erreurs !

Solution : le modèle de l'Invocation à distance, «*Remote Procedure Call*» ou «rendez-vous»

- ◦ correspond à la demande d'exécution d'une fonction à un autre processus (*téléphoner*) ;
- ◦ est accompagnée d'un passage de messages entre processus ;
- ◦ comme pour l'appel d'une fonction :
  - ◇ l'appelant attend la réponse de l'appelé (la réponse est facultative) ;
- ◦ correspond au **modèle client/serveur** ;
- ◦ peut être **mis en oeuvre** par messages synchrones ou asynchrones (utilisation de tampons, modèle producteur/consommateur).

Modèle producteur/consommateur en mode message asynchrone



Deux points de vue simultanés

- **asynchronisme** entre sites distants : propagation sur le réseau (le réseau n'est pas modélisé mais il pourrait l'être) ;
- **synchronisation locale** sur les tampons d'émission et de réception.

Asynchronisme entre l'émetteur et le récepteur

- L'émetteur
  - effectue un envoi ;
  - reprend son exécution immédiatement après.

  Le message est transmis par le réseau de façon asynchrone par rapport à l'émetteur.

  Attention

  Le message est transmis **uniquement** à travers le réseau :
  - si le buffer d'envoi est **plein** ;
  - ou si le programmeur le **décide**.

- Le récepteur
  - décide de traiter un message ;
  - prend le premier message disponible ;

  Le message était dans une file d'attente de réception.

Synchronisation locale

Le **processus émetteur** a fourni l'adresse d'un tampon (contenant le message) partagé avec l'interface réseau.
Différents cas possibles :

- ▷ l'émetteur reste bloqué tant que le message n'a pas été envoyé (attente du tampon redevenu libre) ;
  ⟹ *émission bloquante (celle que l'on utilisera en TP !)*
- ▷ l'émetteur reste bloqué tant que le message n'a pas été recopié dans l'interface réseau ;
  ⟹ *émission bloquante moins longtemps (pas accessible en Python)*
- ▷ l'émetteur reprend le contrôle alors que l'interface réseau utilise le tampon, il sera avertit par un signal quand il sera réutilisable.
  ⟹ *émission non bloquante (déconseillée dans le cas d'un échange : on n'a pas de garantie sur l'émission effective, et on peut attendre une réponse qui ne viendra jamais)*

Le **processus récepteur** a fourni l'adresse d'un tampon partagé avec l'interface réseau.
Différents cas possibles :

- ▷ le récepteur reste bloqué tant qu'un message reçu n'a pas été écrit dans le tampon ;
  ⟹ *réception bloquante (celle que l'on utilisera en TP !)*
- ▷ le récepteur continue son exécution et se bloque quand il ne peut plus avancer sans message reçu ;
  ⟹ *réception non bloquante + attente (déconseillé car compliqué à mettre en œuvre)*
- ▷ le récepteur continue mais il peut savoir si un message a été reçu ;
  ⟹ *réception non bloquante + opération de test de message (déconseillé, car débouche souvent sur de l'«attente active»)*

Attention

Gestion du buffer d'envoi et de réception dans le protocole TCP : rôle du bit PUSH !

Éléments importants

- TSAP, «*Transport Service Access Point*» ;
- Mode «orienté connexion» *vs* mode datagramme ;
- Protocoles TCP et UDP ;
- Mode «client/serveur» ;

Une **interface de programmation** définie pour mettre en place **simplement** des communications :

- chaque communication a lieu avec :
  - **un interlocuteur** : communication «point à point», ou «unicast» ;
  - **plusieurs interlocuteurs** : communication par «diffusion» ou «multicast» ;
- la communication correspond à l'échange de données entre les interlocuteurs :
  - des **données en continu** : flux d'octets de taille indéfinie, non connue à l'avance ;
  - des **paquets** : données de taille fixe et réduite connue à l'avance.

Deux types de communication uniquement en TCP/IP

1. mode «**connecté**»
   - elle ne concerne que **deux interlocuteurs** : un de chaque côté (point à point) ;
   - les données arrivent les unes après les autres dans «l'ordre d'émission» ;
   - la communication est **bi-directionnelle** (dans les deux sens) ;
   - elle est «full-duplex», les deux interlocuteurs peuvent échanger **simultanément** ;
   - il y a une **garantie contre la perte de données**.
     *C'est le mode offert par le protocole **TCP**, «Transmission Control Protocol».*
2. mode «**datagramme**»
   - elle peut concerner un ou plusieurs interlocuteurs (unicast ou multicast) ;
   - les données sont groupées dans des **paquets de taille limitée** ;
   - il peut y avoir des **pertes de paquets**.
     C'est le mode offert par le protocole UDP, «User Datagram Protocol».

Attention

Le mode «connecté» est simulé par TCP sur un réseau en mode «datagramme».

Modèle Client/Serveur

Un logiciel «serveur» **attend** la communication en provenance d'un logiciel «client».

Localisation du logiciel serveur

- un ordinateur est localisable sur Internet grâce à son adresse IP ;
- un ordinateur ne possède habituellement qu'une adresse IP joignable ;
- un ordinateur peut exécuter plusieurs programmes qui peuvent vouloir communiquer simultanément ;
- il faut **multiplexer ces communications** en «sachant» avec quel programme communiquer : notion de «port» !

À chaque processus communiquant est associé un port

Pour une communication en «mode connecté» :

- un Serveur qui **attend** la connexion du client ;
- un Client qui **effectue** la connexion au serveur.

Pour localiser le Serveur ? Connaître le numéro de port où attend la communication !

Comment connaître le numéro de port ?

Le point sur les communications sur un ordinateur :

- chaque communication est associée à **un seul programme donné** (logiciel de messagerie, navigateur web, client de chat, *etc*) ;
- chaque communication se fait suivant un **protocole donné** (SMTP, POP pour récupérer le courrier, HTTP, *etc*) ;
- chaque protocole est associé à un «serveur» particulier : serveur SMTP pour l'envoi de courrier, serveur Web, serveur FTP, *etc*.
- un **numéro de port identifie un serveur donné** : il faut rendre **standard** les numéros de port !
  Exemple : http : 80, ftp : 21, smtp : 25, DNS : 53 *etc*, la liste dans le fichier `/etc/services`.

Le client veut communiquer avec un serveur donné ? il utilise le port standard associé !

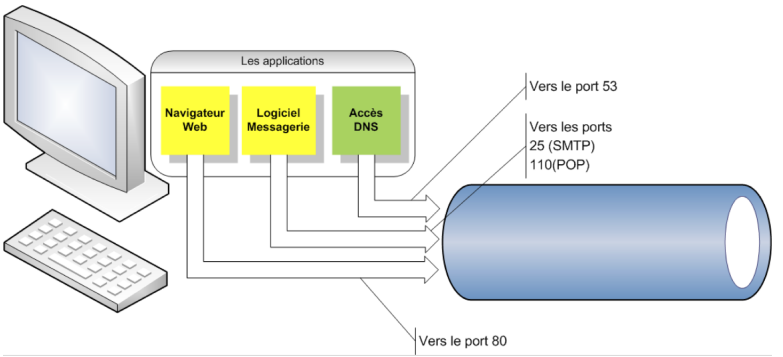
Notion de numéro de port

- ◇ différentes communications peuvent avoir lieu pour des protocoles différents, donc des programmes différents, donc des numéros de port différents ;
- ◇ chaque communication sur une machine est identifiée par un **TSAP**, «*Transport Service Access Point*», c-à-d un couple (@IP, numéro de port).

Comment un ordinateur peut-il voir plusieurs communications simultanément ?

On ajoute également la notion de **numéro de port** :

- ⋆ il varie de 1 à 65535 (sur 16 bits) ;
- ⋆ il est associé à un seul programme ;
- ⋆ du côté de la machine *cliente*, il peut prendre n'importe quelle valeur ;
- ⋆ du côté de la machine *serveur*, il permet à la machine cliente de désigner le programme que l'on veut contacter ;



Le port permet de **multiplexer** les communications :

- ◦ chaque datagramme sera identifié par le TSAP_{source} duquel il transporte les données ;
- ◦ tous les datagrammes utilisent le même lien de communication ;
- ◦ lors de leur arrivée sur la machine destination, ils sont identifiés par leur $\text{TSAP}_{destination}$ et remis au bon processus.

Objectifs

- ⋆ **fournir des moyens de communications** entre processus, IPC, «Inter Processus Communication», utilisable en toutes circonstances : échanges locaux sur la même machine (boucle) ou sur le réseau.
- ⋆ **masquer les détails d'implémentation** des couches de transport ;
- ⋆ fournir une **interface d'accès** qui se rapproche des **accès fichiers** pour simplifier la programmation

La notion de socket ou de «prise»

C'est un **point d'accès pour les services de transport** :

- ◇ Elle possède un type : *Quel protocole de transport va être utilisé ? Pour quel mode de communication ?*
- ◇ Elle permet d'utiliser un ensemble de **primitives de service** ;
- ◇ Elle **encapsule** des données : un descripteur et des files d'attente des messages en entrée et en sortie ;
- ◇ Elle est **identifiée** par un nom unique : le TSAP, c-à-d «le numéro de port» et une @IP.

Le protocole TCP

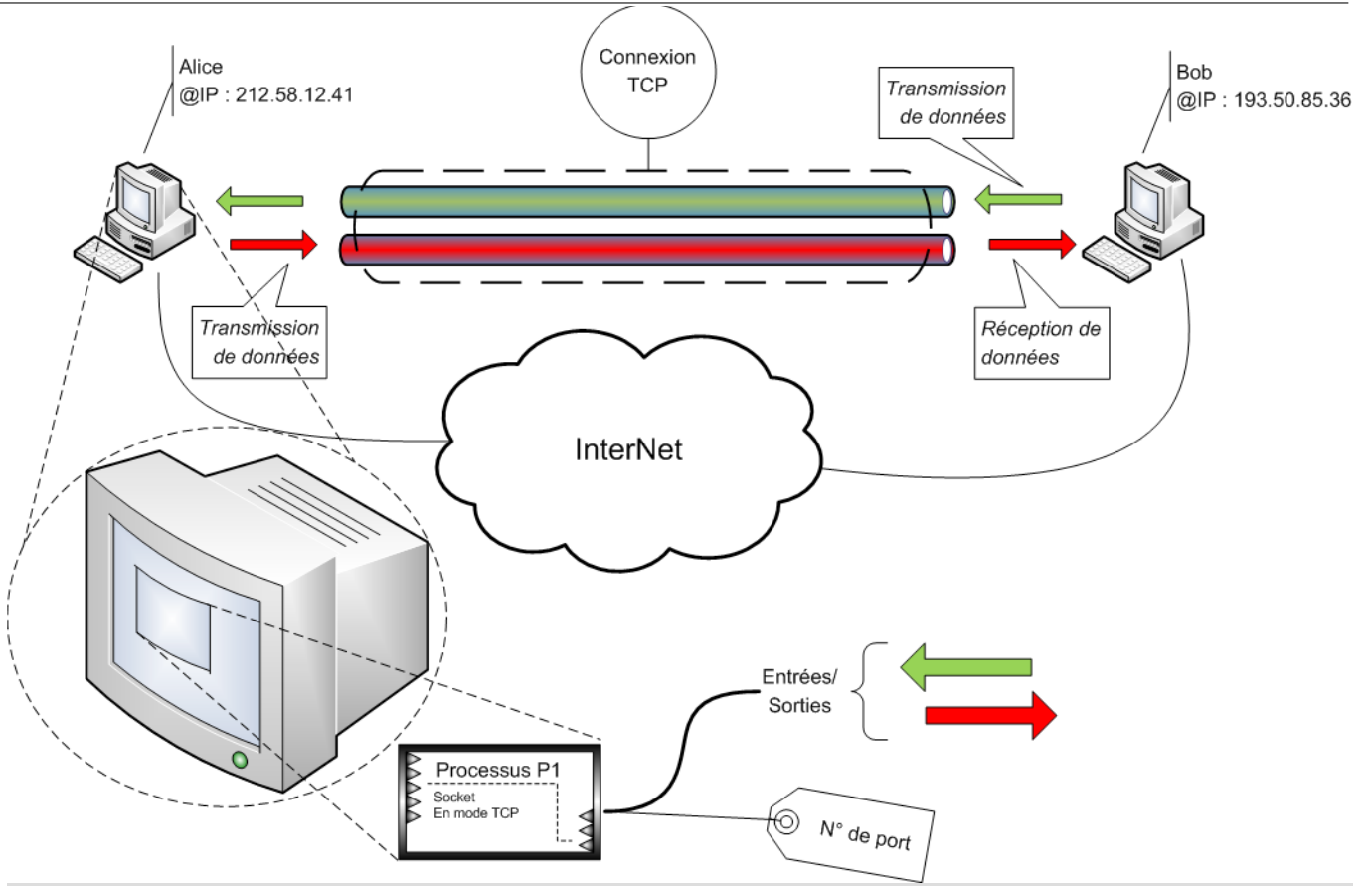
- ⋆ C'est un protocole de transport fiable, en mode connecté, en mode bidirectionnel.
- ⋆ Une socket TCP peut être utilisée par plusieurs connexions TCP simultanément du côté serveur :
  - ◇ on peut traiter simultanément plusieurs clients ;
  - ◇ on peut configurer une «file d'attente» des clients : si un client se présente il est mis en attente ou renvoyer directement.
- ⋆ Une communication est identifiée par le couple d'adresse IP/port, TSAP, des deux extrémités :

$$TSAP_{client} \Longleftrightarrow TSAP_{serveur}$$

- ⋆ Un échange TCP est un flot continu d'octets. Les données sont reçues dans l'ordre de leur transmission.

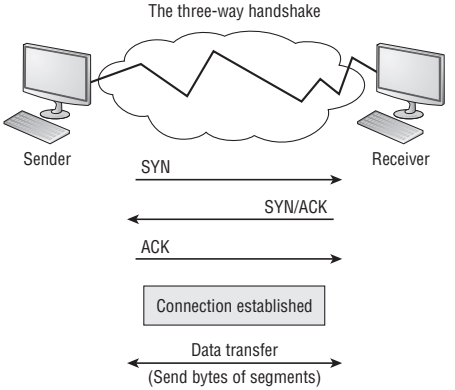
*Une optimisation de TCP est de **temporiser** l'envoi des données dans des tampons pour augmenter la taille des envois, mais il est **possible de demander l'émission immédiate** des données.*

Le protocole TCP : connexion et échanges



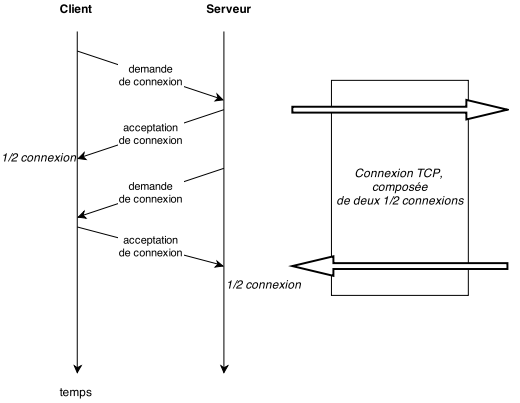
Une communication «orientée connexion» correspond à :

- une **demande d'accord** de la part de l'interlocuteur **avant de lui envoyer des données**, c-à-d une $1/2$ connexion ;
- un envoi de données **sans perte** et **sans erreur** ;
- une communication **bi-directionnelle** (d'un interlocuteur vers l'autre et vice-versa) et «*full duplex*» (chaque interlocuteur peut communiquer simultanément avec l'autre) ;



«*The three-way handshake*», correspond à l'établissement d'une communication depuis le client vers le serveur :

- une demi-connexion du client vers le serveur ;
- une demi-connexion du serveur vers le client ;



Celui qui **initie** la communication est le client.
Celui qui **attend** la communication est le serveur.

Firewall & Filtrage

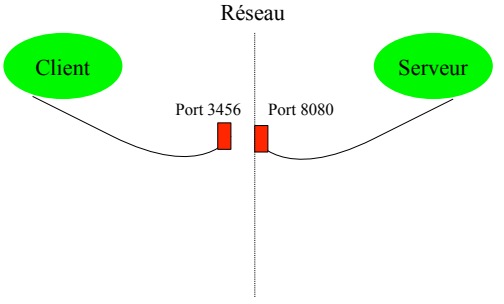
Communication autorisée (non filtrée) :
Si le client est dans le LAN et le serveur sur Internet $\Longrightarrow$ Communication Intérieur$\longrightarrow$Extérieur
L'appartenance du Client au LAN est déduite grâce à la présence du «`SYN`».

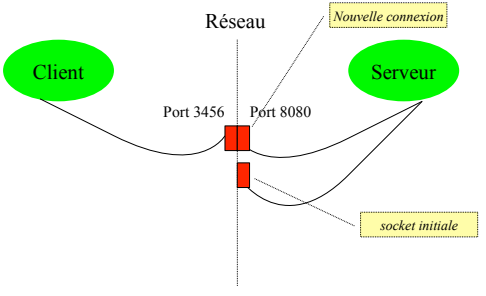
À noter : la demande de 1/2 connexion du serveur (`SYN`), ainsi que son acceptation de la demande de 1/2 connexion du client (`ACK`), sont transmises dans le même message, d'où la simplification en 3 échanges seulement.

Les différentes étapes pour l'établissement de la connexion :

Les instructions réseaux à utiliser sont indiquées dans un cadre en fin de ligne.

1. Le serveur attend sur le SAP `[@IP serveur, numéro de port]`
   `socket, bind, listen, accept`
2. Le client obtient automatiquement un numéro de port libre (par ex. `3456`)
   `socket`





3. Le client se connecte au serveur `connect`
   Le système d'exploitation du client et du serveur, mémorise la connexion par un couple :
   $(TSAP_{client} \iff TSAP_{serveur})$
   `[@IP client, 3456]` ⟺ `[@IP serveur, 8080]`
   Cette connexion peut être affichée avec la commande Unix « `netstat` » ou « `ss` » sous Linux.

Remarques :

- La primitive de programmation `accept` retourne au serveur une nouvelle *socket* associée à la connexion avec le client.
  C'est par cette socket que l'on communique avec le client.
- Le serveur peut recevoir la connexion de nouveaux clients sur la socket initiale.
  Un serveur peut avoir plusieurs communications simultanées avec différents clients.
  Chacune de ces communications correspond à un couple différent de TSAP (le même du côté du serveur associé à un TSAP côté client différent pour chaque communication).

Schéma de fonctionnement en TCP

Serveur

```
1 socket
2 bind
3 listen
4 accept
5    recv, send
6 close
```

Client

```
1 socket
2 connect
3    recv, send
4 close
```

Le protocole UDP

- C'est un protocole de transport **non fiable**, **sans connexion**.
- L'échange de données se fait par datagrammes : un «datagramme UDP» = un «datagramme IP».
- L'ordre dans lequel les paquets sont envoyés peut ne pas être respecté lors de leur réception.

Schéma de fonctionnement en UDP

Serveur

```
1 socket
2 bind
3    recvfrom, sendto
4 close
```

Client

```
1 socket
2    recvfrom, sendto
3 close
```

Les primitives de l'interface socket

- **`socket`** : permet de créer un TSAP, «Transport Service Access Point», c-à-d un nouveau point d'accès de service de transport

Trois paramètres d'appel :

1-famille d'adresses utilisée :

```
1 AF_UNIX (communication locale à une machine)
2 AF_INET (communication réseau avec IPv4)
3 AF_INET6 (communication réseau avec IPv6)
```

2-type de service demandé :

```
1 SOCK_STREAM (flot d'octets en mode connecté)
2 SOCK_DGRAM (datagramme en mode non connecté)
```

3-protocole de transport utilisé

```
1 IPPROTO_TCP
2 IPPROTO_UDP
3 IPPROTO_ICMP
```

Le prototype de la fonction socket :

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4
5 int socket(int famille, int type, int protocole);
```


Le choix d'un numéro de port n'est pas systématique lors de la création de la socket :

- *un client n'a pas besoin de fixer un numéro de port particulier (choisi automatiquement par le système) ;*
- *un serveur qui attend des connexions doit définir sur quel numéro de port il les attend.*

⋆ **bind** : permet d'attribuer un numéro de port à la socket.

Trois paramètres d'appel :

- ◇ le numéro de descriptif de la socket (retourné par la fonction `socket`) ;
- ◇ une structure de données adresse de socket de type `sockaddr_in` ;
- ◇ la taille de cette structure ;

```
1 #include <sys/socket.h>
2 struct sockaddr_in   {
3    short           sin_family;
4    u_short         sin_port;
5    struct in_addr  sin_addr;
6    char            sin_zero[8];
7    };
```

Exemple d'utilisation :

```
 1 struct sockaddr_in adresse_socket;
 2 adresse_socket.sin_family = AF_INET;
 3 adresse_socket.sin_port = 16;
 4 /* Conversion htonl dans le sens reseau */
 5 adresse_socket.s_addr = htonl(INADRR_ANY);
 6
 7 if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
 8 { /* cas d'erreur */ }
 9 if (bind(s, &sin, sizeof(sin)) < 0)
10 { /* cas d'erreur */ }
```

- **`listen`** :
  - Utilisée dans le **mode connecté** lorsque plusieurs clients sont susceptibles de demander une ou plusieurs connexions avec le serveur.
  - Il permet de fixer le nombre d'appel maximum que pourra traiter le serveur avant de les rejeter (les appels non gérés immédiatement sont alors mis en attente).

```
1 int listen (int descripteur_socket, int max_connection);
```

- **`accept`** :
  - Utilisée dans le **mode connecté**, permet de se bloquer en attente d'une nouvelle demande de connexion.
  - Après l'accept, la connexion est complète entre les deux processus.
  - Pour chaque nouvelle connexion entrante, la primitive `accept` renvoie un pointeur sur une **nouvelle socket** de structure identique à la précédente :
    - la socket originale sert à établir une nouvelle connexion ;
    - la nouvelle socket permet l'échange avec le client associé.

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 int accept (int nouvelle_socket, struct sockaddr_in *adresse_client,
4       int longueur_adresse_client);
```


* **`connect`** :
  - Cette primitive permet à un client de se **connecter** à un serveur.
  - Elle ouvre une connexion entre le client et le serveur.
  - On doit lui fournir l'adresse IP du serveur (la partie locale est renseignée automatiquement).
  - Pour chaque opération d'écriture/lecture, seul le descripteur de la socket est à fournir à chaque fois.

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 int connect(int descripteur_socket,
4             struct sockaddr_in *adresse_serveur,
5             int longueur_adresse);
```

* **`send`**, **`recv`**
  - Ces primitives permettent l'**échange d'information** au travers de la socket.
  - Elles s'utilisent de la même façon que les instructions `read` et `write` sur fichier avec une option supplémentaire pour préciser des options de communication.

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3
4 int send (int socket, char *zone, int longueur_zone, int options);
5 int recv (int socket, char *zone, int longueur_zone, int options);
```

Les options permettent d'indiquer si les données urgentes, etc.

Le client

```
1  import socket, sys
2
3  adresse_symbolique_serveur = 'localhost' # la machine elle-meme
4  numero_port_serveur = 6688 # le numéro de port sur le serveur
5
6  # réalisation de la requête DNS pour obtenir l'adresse IP
7  adresse_serveur = socket.gethostbyname(adresse_symbolique_serveur)
8
9  ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10
11 try:
12    ma_socket.connect((adresse_serveur, numero_port_serveur)) #TSAP designant le serveur
13
14 except Exception as e:
15    print (e.args)
16    sys.exit(1)
17
18 while 1:
19    entrée_clavier = input(':>')
20    if not entrée_clavier:
21       break
22    ma_socket.sendall(bytes(entrée_clavier,encoding='UTF-8')+b'\n')
23
24 ma_socket.close()
```


Le serveur

```
 1 import socket
 2
 3 masque_acces = '' # filtre les clients, ici aucun n'est filtre
 4 numero_port_serveur = 6688 # identique à celui du client
 5
 6 # création de la socket d'attente de connexion
 7 ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
 8
 9 # Permet de ne pas attendre pour réutiliser le numéro de port
10 ma_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
11
12 # Accroche le numéro de port à la socket
13 ma_socket.bind((masque_acces, numero_port_serveur))
14
15 # Configure la file d'attente
16 ma_socket.listen(socket.SOMAXCONN)
17
18 # L'accept renvoie une nouvelle socket
19 (nouvelle_connexion, tsap_depuis) = ma_socket.accept()
20 print ("Nouvelle connexion depuis ", tsap_depuis)
21 while 1:
22    ligne = nouvelle_connexion.recv(1000) # au plus 1000
23    if not ligne :
24       break
25    print (ligne)
26 nouvelle_connexion.close() # fermeture de la connexion vers le client
27
28 ma_socket.close() # fermeture de la socket d'attente de connexion
```

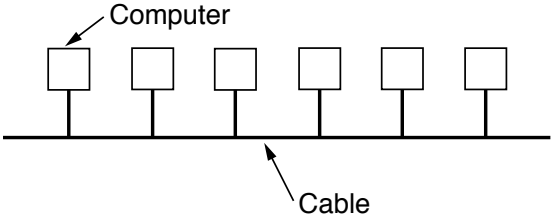

Plan de la partie Fondamentaux

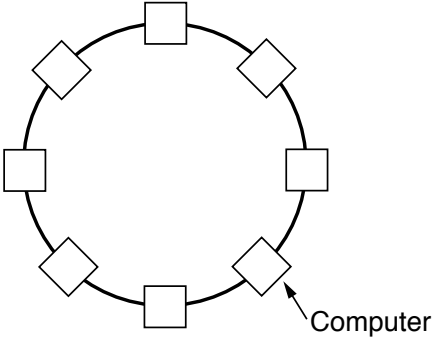
Éléments importants

- ▷ Deux topologies théoriques : diffusion et «point-à-point» ;
- ▷ Les réseaux utilisés et le matériel d'interconnexion ;
- ▷ La gouvernance d'Internet : les organisations et les RFCs ;
- ▷ Le réseau TCP/IP : adressage, encapsulation, routage direct & indirect ;
- ▷ Le DNS : global et local ;
- ▷ La configuration du poste de travail.

Le réseau en mode «diffusion»

Les réseaux à diffusion, *broadcast network*, n'ont qu'un **seul canal de communication** que toutes les machines partagent.





Une machine envoie de **petits messages** qui sont reçus par toutes les autres machines :

- ⋆ dans le message un **champ d'adresse** permet d'identifier le destinataire
- ⋆ à la réception du message, une machine teste ce champ :
  - ◇ si le message est pour elle, elle le traite ;
  - ◇ sinon, elle l'ignore.

Exemple :

un couloir sur lequel débouche un certain nombre de portes de bureau.
quelqu'un sort dans le couloir et appelle une personne,
tout le monde entend l'appel mais une seule personne répond à l'appel
cas des annonces dans les gares ou les aéroports

Le réseau en mode «diffusion»

Contraintes

⋆ chaque machine appartenant au réseau **doit disposer** d'une adresse.

Avantages

◦ envoyer un message **vers tout le monde** en utilisant une adresse particulière :
Ce message est traité par toutes les machines.
Ce procédé est appelé «diffusion générale» ou «broadcasting».

◦ transmettre un message à **un sous-ensemble** de machines :
Ce procédé est appelé «diffusion restreinte» ou «multipoint» ou «multicast».
Une façon de faire consiste à utiliser les n bits d'adresse de la manière suivante :
⋆ *associer un bit à l'indication de mode multipoint*
⋆ *utiliser les n-1 bits restants pour l'identification du groupe.*
⋆ *permettre à chaque machine d'appartenir à un ou plusieurs groupes.*

◦ connaître le **temps de transmission** d'un message :
permet de simplifier des algorithmes de communication : «je suis sûr que le récepteur a reçu mon message, mais je ne connais pas son état et s'il a pu le traiter.»

Inconvénients

⋆ la **rupture** du support de transmission **entraîne l'arrêt du réseau**.
Le «hub» ou «switch» tombe en panne.

⋆ la **panne d'un des matériels** connectés au réseau **ne provoque pas de panne du réseau** (en général…).

Dans le cas d'Ethernet, mais aussi de WiFi, de Bluetooth…

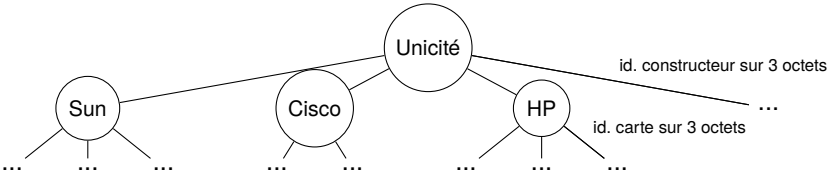
Chaque carte réseau possède une adresse matérielle appelée adresse MAC (Medium Access Control) :

- ▷ **unique** par rapport à toutes les cartes réseaux existantes !
- ▷ **exprimée sur 48 bits** ou 6 octets.
  - ◇ **Syntaxe** : `08:22:EF:E3:D0:FF`
  - ◇ **Adresse de Broadcast** : FF:FF:FF:FF:FF:FF (en IPv4).

Pour garantir l'**unicité** :

a. des «tranches d'adresses» sont affectées aux différents constructeurs :

00:00:0C:XX:XX:XX	**Cisco**
08:00:20:XX:XX:XX	**Sun**
08:00:09:XX:XX:XX	**HP**
00:09:BF:XX:XX:XX	**Nintendo**
00:D0:F1:XX:XX:XX	**Sega**

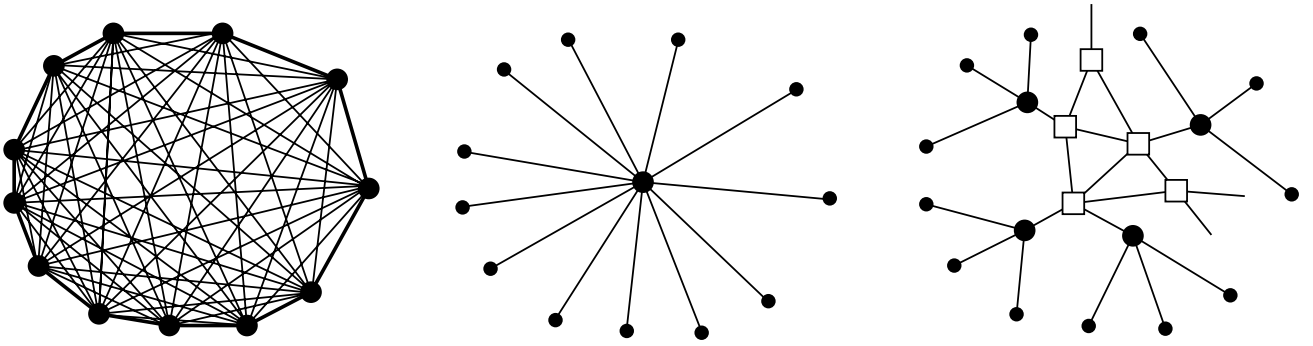


Ce préfixe est appelé OUI, «Organization Unique Identifier».
La liste est consultable à `http://standards.ieee.org/regauth/oui/index.shtml`.

b. **chaque constructeur** numérote différemment chaque carte réseau qu'il construit.

Avantage	**impossible de trouver deux fois** la même adresse dans un même réseau
Inconvénient	elle ne donne **aucune information sur la localisation** d'une machine «*dans quel réseau est la machine avec qui je veux parler ?*»
Solution	**utilisation de l'adresse IP !**

Réseaux en mode «point à point»



(a) (b) (c)

Ces réseaux sont formés d'un **grand nombre de connexions** entres les machines prises **deux à deux**.

Le trajet des communications est rendu plus complexe :

- ▷ pour aller de la source au destinataire, un message doit alors **passer par un plusieurs intermédiaires**.
- ▷ il existe plusieurs routes de **longueurs différentes** pour joindre ces deux machines, il est nécessaire d'utiliser de **bons algorithmes d'acheminement des messages** ;

Inconvénient

- ▷ le **temps de transfert** d'un message devient presque impossible à prévoir.
- ▷ il est nécessaire de faire du **routage** des messages dans le réseau.

Le **support physique** relie uniquement **une paire** d'équipements à la fois.

La communication est :

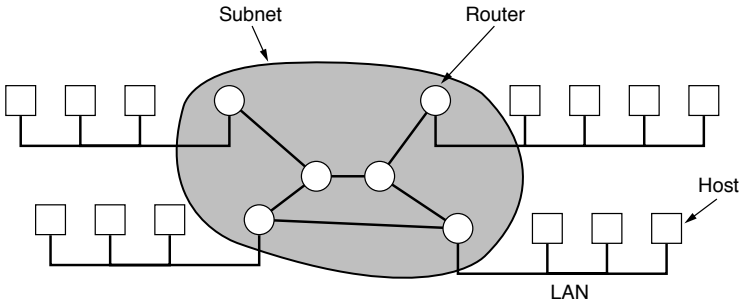
- ▷ **directe** : si les deux équipements sont connectés entre eux ;
- ▷ **indirecte** sinon : deux équipements ne peuvent communiquer que par **l'intermédiaire d'autres nœuds** du réseau.

Le besoin de «routage»

Problème ?	**Comment choisir** par quels intermédiaires passer, ce qui s'appelle du «*routage*» ?
Cas d'un réseau en «étoile»	L'algo. de routage est simple : le site central reçoit et renvoie tous les messages. *Le fonctionnement est simple, mais la panne du site central paralyse tout le réseau.*
Cas d'une «boucle simple»	chaque nœud recevant un message de son voisin en **amont** le réexpédie à son voisin en **aval**. En cas de **non remise**, le message est **retiré** par le nœud émetteur pour éviter des boucles inutiles. Si l'un des nœuds **tombe en panne**, le réseau est **bloqué**. *Une solution partielle est d'utiliser une «double boucle»*.

Ça se complique, on a deux possibilités :

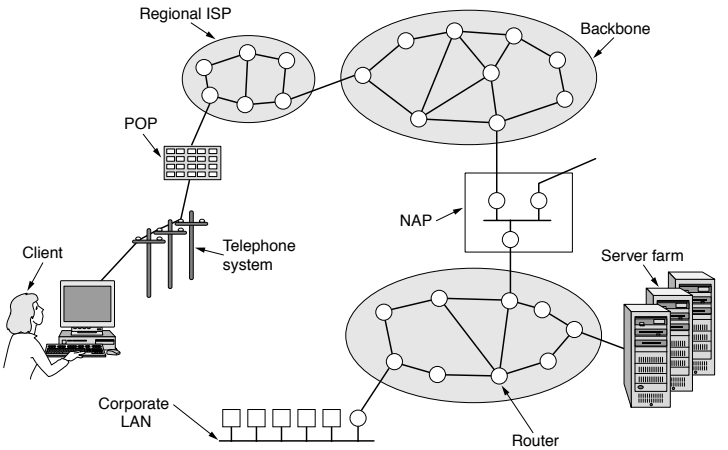
Maillage régulier	l'interconnexion est **totale** (plus besoin de passer par un intermédiaire) la **fiabilité** est **maximale** le **coût** en câblage est **maximal**
Maillage irrégulier	l'interconnexion **n'est plus totale** la **fiabilité diminue** le **routage** des messages peut devenir **complexe** **impossible de prévoir le temps de transfert** d'un nœud à l'autre.



- ▷ **diffusion** : réseau de petite taille, LAN, *Local Area Network* ; *Exemple :Ethernet*
- ▷ **point-à-point** : **réseau d'interconnexion**, constitué uniquement de routeur et de ligne de transmission *Exemple : liaison satellite.*
- ▷ la **combinaison des deux** : WAN, *Wide Area Network*.

Inter(connexion)Net(work) :

- ⋆ du client à la maison ;
- ⋆ de la ligne téléphonique au POP, «*Point of Presence*» vers ATM ;
- ⋆ ou en passant par de la fibre optique pour une liaison IP directe vers l'ISP ;
- ⋆ en passant par l'ISP, *Internet Service Provider* ;
- ⋆ au réseau national : *backbone* ;
- ⋆ par une connexion à un réseau, *Network Access Point* ;
- ⋆ vers le LAN de l'entreprise…



- **IEEE**, «*Institute of Electrical and Electronics Engineers*» : organisation internationale chargée de superviser le développement et l'adaptation de standards internationaux.
  Par exemple dans le cadre des communications sans fil avec l'IEEE 802.11.

- **ANSI**, «*American National Standards Institute*» : organisation non gouvernementale à but non lucratif contribuant à l'élaboration de standards pour l'industrie en protégeant les intérêts du public.
  Par exemple, le code ASCII, American Standard Code for Information Interchange.

- **EIA**, «*Electronic Industries Association*» : organisation à but non lucratif proche de l'ANSI dédiée à la résolution des problèmes de fabrication de composants électroniques.
  La prise du «twisted-pair» appelé RJ-45.

- **ISO**, «*International Standards Organization*» : *le nom vient du grec «isos» qui veut dire «égaux»*, organisation de standardisation internationale dont les membres appartiennent à des comités de standardisation de différents pays. Elle est basée sur le volontariat (pour les Etats-Unis, c'est l'ANSI qui participe).
  Dans le cadre des réseaux, sa contribution principale est le modèle OSI : «Open Systems Interconnection Reference Model» qui sert de base à l'analyse et la conception des ***protocoles*** *de communication.*

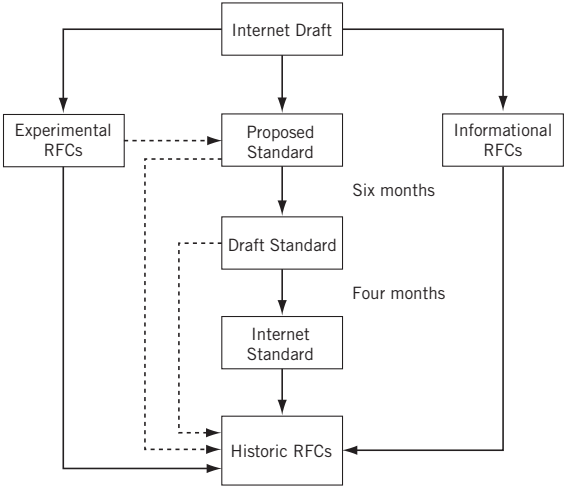
- **ITU-T**, «*International Telecommunications Union-Telecommunication Standards Sector*» : permettre une infrastructure mondiale non seulement dans les réseaux de données mais également dans la téléphonie, PSTN, «public switched telephone network». Les Nations Unies ont formées un comité le CCITT, «Consultive Committee for International Telegraphy and Telephony» compris dans l'ITU, «International Telecommunications Union».
  Tout communication qui traverse les frontières d'un pays doit se conformer aux recommandations de l'ITU-T.
  *Une technologie comme ATM, «*Asynchronous Transfer Mode*» est un standard ITU-T.*

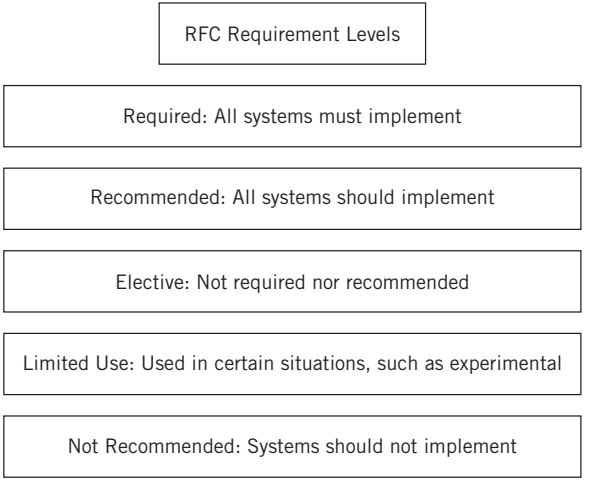
- Des *forums* : promouvoir une technologie et sa standardisation.
  *Exemple le MEF, «*Metro Ethernet Forum*».*

- Obligations de se conformer aux régulateurs nationaux, comme le **FCC**, «*Federal Communications Commission*» qui surveille, par exemple, l'utilisation des différents fréquences dans le cas de transmission sans fil.

Les «Request for Comment» & l'IETF, «Internet Engineering Task Force»

- IETF : organisme responsable du développement des standards de l'Internet.
  - son propre système de standardisation des **protocoles** utilisés par les matériels connectés à Internet.
    Les protocoles concernés sont plus proches de l'utilisateur (applications) que du matériel.
- Les standards Internet : des spécifications testées et qui doivent être suivies.
  - la procédure d'élaboration est stricte :
    - «Internet draft» : document de travail, souvent modifié sans statut particulier et de durée de vie d'au plus 6 mois ;
    - les développeurs travaillent à partir de ces «drafts» ;
    - un draft peut être publié sous la forme d'un RFC (juste une appellation, il n'y a plus besoin de retour ou *feedback*).
  - RFC identifiée par un numéro, librement consultable et, suivant le niveau de *requirements*, **obligatoire**, ou pas.
  - Tous les RFCs ne sont pas des standards, même ceux définissant des protocoles entiers.
  - Après un certain temps, la RFC peut arriver à maturité et finir dans les «RFC historiques».





- **InterNIC**, «*Internet Network Information Center*» entre 1992-1998 :
  - organisme public américain chargé de la gestion centrale des adresses et des noms de domaines Internet et de l'accréditation d'un organisme homologue dans chaque pays, les organismes délégués :
    - AfriNIC (Afrique),
    - APNIC (Asie, Pacifique),
    - ARIN (Amérique du Nord),
    - LACNIC (Amérique du Sud, îles Caraïbes),
    - RIPE NCC (Europe, Moyen-Orient)
    - NIC France ou afnic, NIC Angleterre, etc.

- **ICANN**, «*Internet Corporation for Assigned Names and Numbers*» :
  - Organisation créée en octobre 1998, pour s'ouvrir à la concurrence
  - traite les noms de domaine et leur délégation (par exemple VERISIGN Inc. : zone « .com ») ;
  - exploitation des serveurs de la racine du DNS (ceux qui font autorité) ;
  - allocation de blocs de numéro IP ;
  - en France, les prestataires (fournisseurs d'accès) font l'intermédiaire avec l'afNIC

- **IANA**, «*Internet Assigned Numbers Authority*» :
  - tient l'annuaire : adresses IP & numéros de protocoles ;
  - adresses IP et numéros d'AS : déléguées aux RIR régionaux, «Regional Internet Registries» ;
  - numéros de protocoles et de ports (entre 1 et 1023) ;
  - déléguées aux LIRs, «Local Internet Registry» (eg. FAI).

- Les «**Registrar**» :
  - Un registrar (bureau d'enregistrement) est une société ou une association permettant le dépôt de noms de domaine internet, dans les TLD, «Top Level Domain», où il n'y a pas de vente directe.
  - Il faut payer un certain montant pour acquérir et protéger un nom de domaine.

- **GIP Renater** (Groupement d'Intérêt Public) :
  - Réseau de la recherche en France, «Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche»

La conception

Contraintes du protocole IP «Internet Protocol» RFC 791 :

- ⋆ utiliser la topologie réseau **point à point** (pour permettre de franchir des grandes distances c'est obligatoire) ;
- ⋆ la panne d'un équipement du **sous-réseau d'interconnexion** ne doit pas entraîner une rupture du réseau ;
- ⋆ privilégier la **disponibilité** du réseau : il doit servir au **maximum**.

Le but est que les **échanges persistent** :

- ▷ du moment que l'ordinateur **source** et l'ordinateur **destination fonctionnent** ;
- ▷ même si certains **routeurs** ou certaines **lignes de transmission** tombent en **panne** (origine militaire de la création d'Internet par le DoD, «*Department of Defense*», états-uniens).

Les moyens

- ◦ privilégier la **décentralisation** : pas de nœud central, redondance et distribution des informations nécessaires au fonctionnement du réseau (routeurs, DNS par exemple) ;
- ◦ privilégier le côté **dynamique** : chaque appareil connecté au réseau recherche/découvre tout le temps les matériels nécessaires à sa communication (routeurs, lignes de transmission, chemins, *etc.*) ;
- ◦ définir une **architecture très souple** pour pouvoir mettre en œuvre des applications très diverses comme le transfert de fichiers ou la transmission de la parole en temps réel (TCP et UDP) ;
- ◦ faciliter le **routage** : construire une méthode simple et rapide (opérations binaires par exemple) ;
- ◦ permettre le **regroupement de machines** pour les gérer ensemble (regroupement en réseau) ;
- ◦ faciliter le travail de l'administrateur (*sisi…*).

Adressage pour le protocole IP (IPv4)

Chaque ordinateur et chaque routeur du réseau Internet possède une adresse IP.

L'adresse IP est une **adresse binaire** composées de deux parties `<id. réseau><id. machine>`:

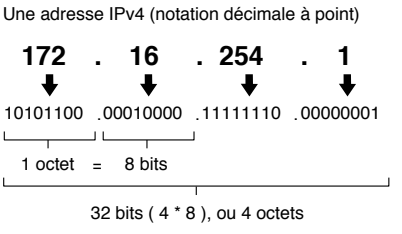
- ⋆ un **identifiant de réseau** ;
- ⋆ un **identifiant machine** pour la distinguer dans le réseau.
  Chaque adresse IP doit être unique pour permettre de la localiser sur la planète.

- ⋆ Il existent **différentes répartitions** des 32 bits entre identifiant réseau et identifiant machine :
  - ◇ ces **différentes répartitions** définissent un ensemble de **classes de réseaux** ;
  - ◇ ces classes **ne sont plus utilisées** en CIDR, où on indique uniquement le nombre de bits de la partie réseau ;

Propriétés

- ▷ Codée sur **32 bits.**

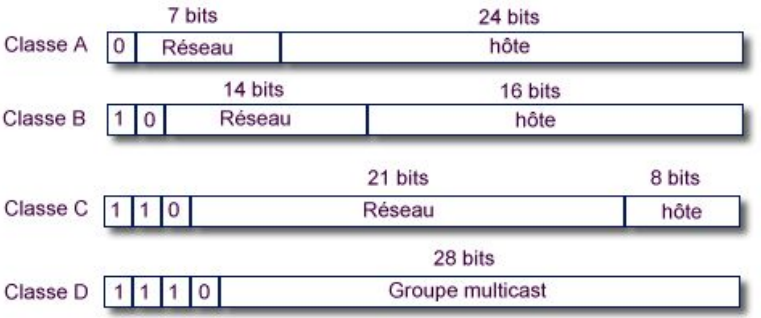
- ▷ Représentée par **commodité** en «*décimale pointée*» : 4 entiers variant entre 0 et 255 séparés par des points
  exemple : 164.81.1.4



- ▷ un **organisme officiel**, le NIC, «*Network Information Center*», est seul habilité à délivrer des numéros d'identification des réseaux.

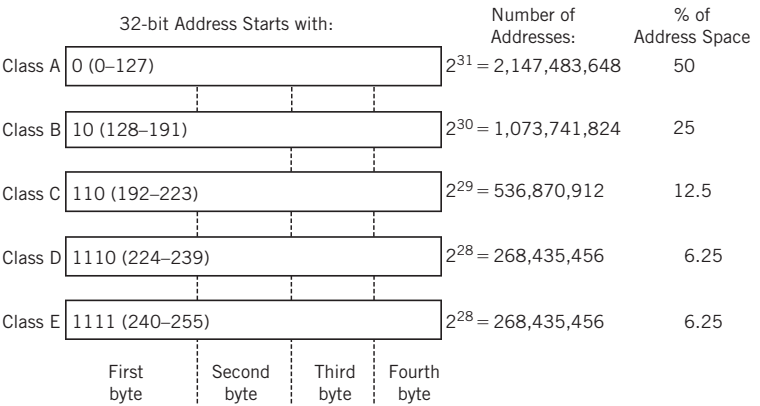
- ▷ il y a, **en général**, une **seule adresse IP** par interface réseau.
  Dans le cas d'un routeur interconnectant 2 réseaux différents, il possède une adresse IP pour chacune de ses interfaces connectées à un réseau.

Les classes de réseaux définies par un «préfixe»



Les adresses de classe A sont peu utilisées. Exemple : 3.0.0.0/8, AS80, GE-CRD - General Electric Company.

Aux niveaux des adresses IP



Ces **adresses** permettent :

- ⋆ des envois de messages **multi-destinataires** ;
- ⋆ désigner la **machine courante** ;
- ⋆ désigner le **réseau courant**.

Tout à zéro		L'ordinateur lui-même
Tout à zéro	id. de machine	Un ordinateur sur le réseau lui-même
Tout à 1		Diffusion limitée au réseau lui-même
Id. de réseau	Tout à 1	Diffusion dirigée vers ce réseau
127	Nombre quelconque	Boucle

L'**adresse de «boucle»** (127.X.Y.Z) permet d'effectuer :

- ◇ des communications inter-programme sur la même machine
- ◇ des tests de logiciels réseaux. *Dans ces cas là, les paquets ne sont pas réellement émis sur le réseau.*

D'autres **adresses particulières** :

- ◇ 0.0.0.0 est utilisé par une machine pour connaître sa propre adresse IP lors d'un processus d'amorçage (BOOTP). *Elle devra se procurer une adresse IP par l'intermédiaire d'une autre machine.*
- ◇ 255.255.255.255 est une adresse de diffusion locale car elle désigne toutes les machines du réseau auquel appartient l'ordinateur qui utilise cette adresse⟹ ***pas besoin de connaissance du réseau.***

Réseaux privés, RFC1918

Les adresses pour **réseau privé** ou **intranet** (sans **accès direct** à l'extérieur) :

- ⋆ `10.0.0.0/8` : de `10.0.0.0` à `10.255.255.255` ⟹ *classe A*
- ⋆ `172.16.0.0/12` : `172.16.0.0` à `172.31.255.255` ⟹ *pas de classe*
- ⋆ `192.168.0.0/16` : `192.168.0.0` à `192.168.255.255` ⟹ *classe B*

Ces réseaux ne sont pas «routables» !

Faire le point…

- sur un **réseau à datagramme**, il circule…**des datagrammes !** ;
- le **réseau à datagramme** est appelé **réseau IP** : il utilise des algorithmes, des formats de messages définis dans la norme IP, «*Internet Protocol*» ;
- un **réseau à diffusion** fait circuler des messages de format différent : les **trames** (on parle de **trame Ethernet** ou IEEE 802.3) ;
- un datagramme doit emprunter un réseau à diffusion pour atteindre un ordinateur :
  - principe **d'encapsulation** : le datagramme est «inclus» dans une trame Ethernet :
  - à l'**adresse IP** d'une machine doit correspondre l'identifiant de cette machine dans le réseau à diffusion : une **adresse MAC** :
    - l'adresse MAC est attachée à la carte réseau et est choisie par le constructeur de cette carte ;
    - l'adresse IP est choisi par l'administrateur réseau suivant la configuration qu'il veut donner à son réseau ;

Comment faire la correspondance entre @MAC et @IP ?

a. c'est **l'ordinateur** qui connaît l'adresse MAC de sa carte réseau ;

b. c'est **l'ordinateur** qui connaît son adresse IP ;

c. **Qui** peut dire à quelle adresse IP correspond tel adresse MAC ? **L'ordinateur lui même** !

d. **Définition d'un protocole** pour « questionner » les ordinateurs : ARP, «*Address Resolution Protocol*»

Transmission physique des datagrammes IP

La **couche liaison de données** est chargée de :

- ▷ la mise en correspondance des **@IP** avec les **@MAC** des interfaces physiques.
- ▷ l'**encapsulation** des datagrammes IP afin qu'ils puissent être transmis sur un support physique particulier.
  Lorsque le protocole IP doit envoyer un datagramme à un équipement relié à un réseau à diffusion, la couche liaison de donnée doit construire une trame ethernet avec l'@MAC du destinataire.

Correspondance entre adresses physiques, @MAC, et adresses IP, @IP

Le **protocole ARP**, «*Address Resolution Protocol*» fournit une **correspondance dynamique** entre une adresse IP connue et l'adresse matérielle correspondante.

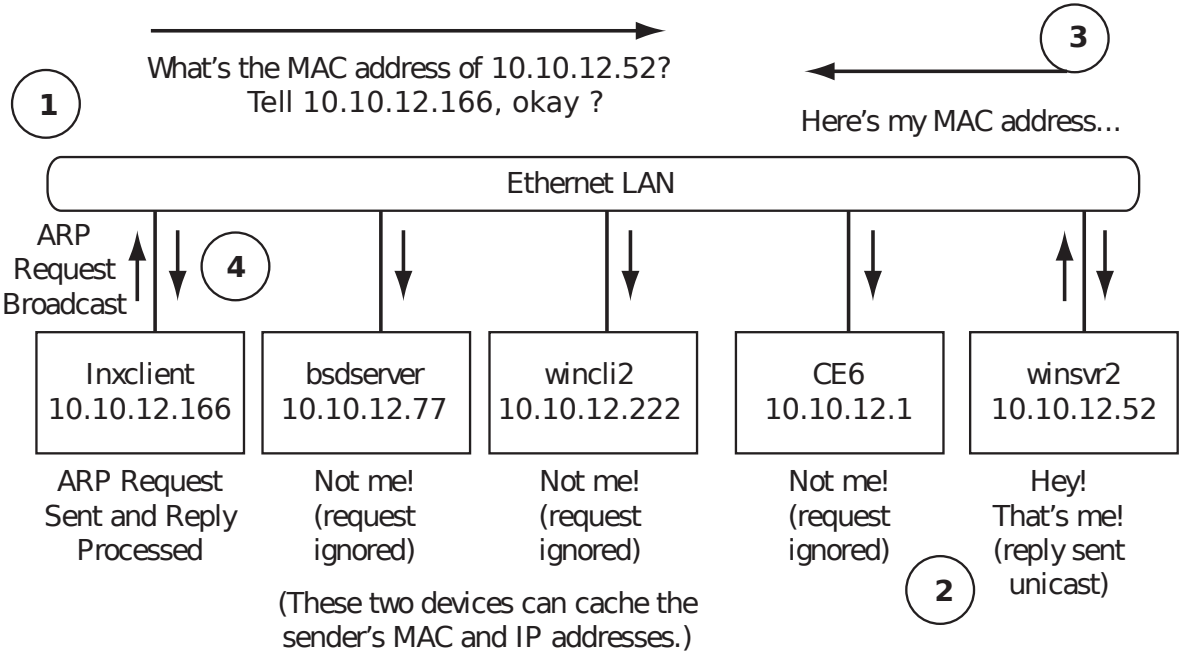
Fonctionnement :

- ▷ ARP dispose d'une **mémoire cache** : lors de la demande de l'@MAC associée à une @IP, ilconsulte sa **mémoire cache ARP** pour voir si l'@IP distante y est mise en correspondance avec @MAC.
  - ◇ Si c'est le cas le datagramme IP est émis immédiatement, enveloppé dans une **trame Ethernet** envoyée à l'adresse physique destination (@MAC).
  - ◇ Sinon la couche liaison de données construit une **requête ARP**.
- ▷ ARP utilise le principe de «*diffusion*» du réseau local : la requête ARP est transmise en «*broadcast*».
- ▷ Lorsqu'un **message ARP** est reçu, la couche liaison de donnée fait :
  - ◇ une **première vérification** pour voir si c'est une **requête ARP** et que **l'@IP demandée correspond à l'@IP locale** alors une **réponse ARP** est renvoyée à destination de l'@MAC de l'expéditeur.
    La machine répond parce qu'elle est concernée : c'est son adresse qui est demandée.
  - ◇ une **seconde vérification** pour vérifier si l'adresse IP de l'émetteur se trouve déjà dans la **mémoire cache ARP locale** sinon il y a **mise à jour** de la mémoire cache avec cette nouvelle association.
    Elle apprend l'association, comme dans le cas d'un «gratuitous ARP»*, c-à-d une réponse ARP non sollicitée envoyée en* ***broadcast***.

Illustration d'ARP

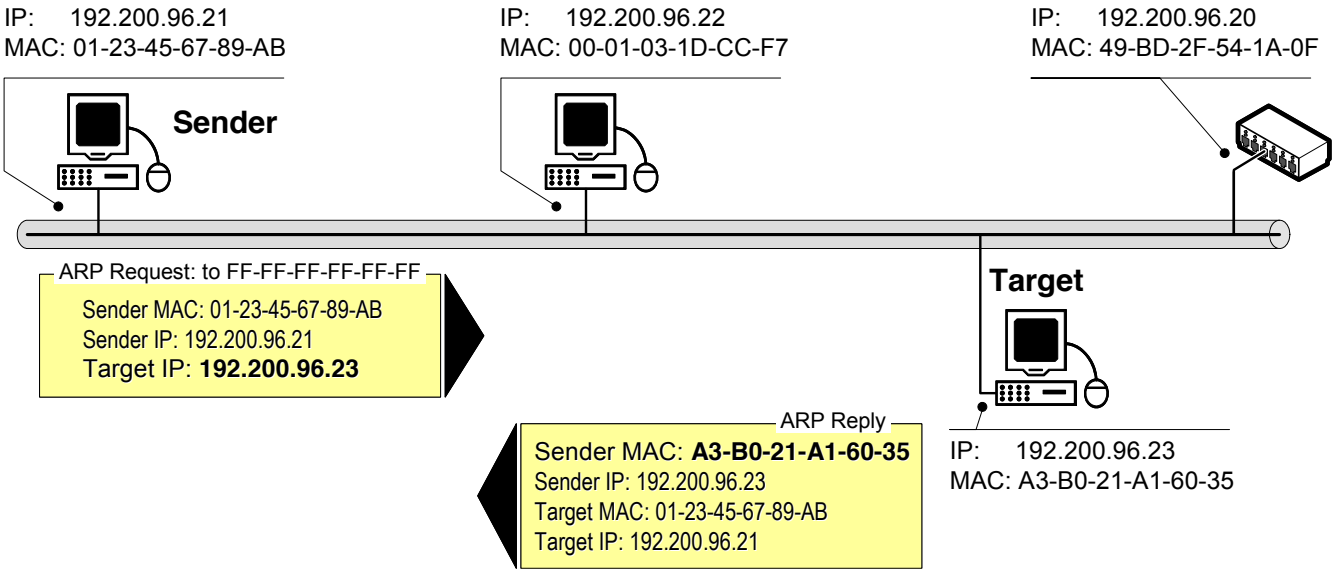
Comment échanger réellement sur un réseau local à diffusion ?

- Les machines ont chacune une carte réseau ;
- Chaque carte a une **adresse MAC unique** donnée par le constructeur ;
- Chaque machine dispose d'une **adresse IP** donnée par l'administrateur du réseau.



La machine `10.10.10.166` *demande l'@MAC de la machine* `10.10.12.52`.

Illustration d'ARP : les échanges



- La requête de la machine «`192.200.96.21`» demande l'@MAC de la machine `192.200.96.23`;
- La réponse de la machine `192.200.96.23` donne la réponse @MAC `A3:B0:21:A1:60:35`.

Le protocole RARP «*Reverse Address Resolution Protocol*»

Il réalise l'opération inverse :

- une machine sans adresse IP connue peut envoyer une requête RARP pour demander son adresse IP.
- une machine particulière (un serveur gérant le réseau) lui répond et lui affecte son adresse IP.
- cette machine dispose d'une table de correspondance : (adresse physique, adresse IP).

Le protocole RARP est utile pour amorcer une station sans disque, un Terminal X-Window, ou une imprimante.

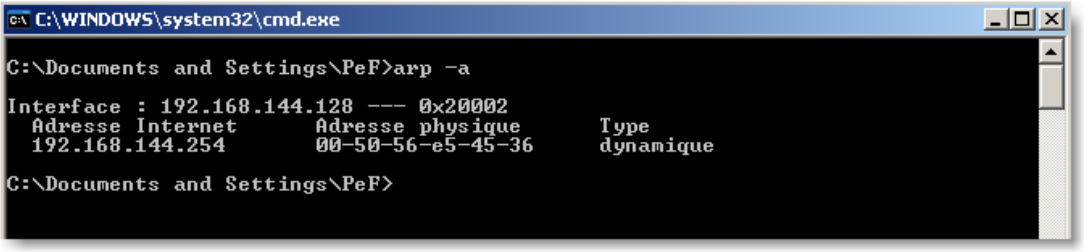
Pour consulter la table ARP

Sous Linux

xterm

```
pef@solaris:~$ ip neighbor
192.168.42.254 dev eth0 lladdr 00:07:cb:cc:d4:e5 STALE
192.168.42.122 dev eth0 lladdr 64:b9:e8:d2:23:ba REACHABLE
```

Sous Windows



Pour envoyer un datagramme d'une source vers une destination, il faut savoir **localiser** la machine destination.

Deux possibilités :

- ▷ les deux machines **font partie** du même réseau local : on parle de **routage direct** (sur Ethernet, on utilisera le protocole ARP et l'envoi direct sur le réseau à diffusion) ;
- ▷ les deux machines **ne font pas partie** du même réseau local : on parle de **routage indirect**.
  On doit passer par un **intermédiaire** qui permet de sortir du réseau local pour aller vers l'extérieur : le **routeur** (ou appelé «passerelle» ou *gateway*).

Pour faire du routage direct ou indirect pour un datagramme

- ▷ connaître l'@IP d'un **routeur de sortie** ;
- ▷ savoir si les deux machines font **partie du même réseau** local :
  - ◇ si elles **sont** dans le même réseau local : remettre **directement** le datagramme à la machine destination ;
  - ◇ si elles **ne sont pas** dans le même réseau local : remettre le datagramme **au routeur** pour l'envoyer **indirectement** à la machine destination.

Comment savoir si Source et Destination sont dans le même réseau local ?

Il faut **comparer** l' `<id. réseau>` des deux adresses : si c'est **la même** $\Longrightarrow$ les deux sont dans le **même réseau local**.

Comment remettre le datagramme au routeur

Il faut utiliser le mécanisme **d'encapsulation** d'un datagramme dans une trame :

- ▷ la **trame** sert à remettre des données d'une machine connectée à un réseau local à une autre machine connectée au même réseau local ;
- ▷ la **trame** possède une @MAC de destination **indépendante** de l'@IP : il est possible d'envoyer la trame à une machine dont l'@IP **ne correspond pas** à son @IP !

Par exemple : on peut envoyer une datagramme à destination de l'extérieur du réseau local à l'@MAC du routeur.

Attention : *les attaques MiTM, «Man-in-the-Middle», opèrent sur l'association @MAC* $\Leftrightarrow$ @IP du routeur !

Localisation de la machine destinataire

Chaque ordinateur connecté au réseau Internet dispose :
- ▷ d'une **@IP** sur 32 bits ;
- ▷ d'un **masque de sous-réseau** : répartition des 32 bits d'@IP entre `<id. réseau><id. machine>`.

Algorithme de choix routage direct/indirect

Envoi d'un datagramme :
- ▷ de S : `164.81.128.34` et masque `255.255.192.0` ;
- ▷ à destination de D : `193.50.185.12`.

*Le masque de réseau de la machine D **n'est pas connu** de S.*

ET	0	1
0	0	0
1	0	1

⋆ combinaison avec l'opérateur binaire «ET», `&`, du **masque de sous-réseau de S** et des @IP de S et D ;

@IP de S	`164.81.128.34`	`10100100`	`01010001`	`10000000`	`00100010`		
masque de S	`255.255.192.0`	`11111111`	`11111111`	`11000000`	`00000000`	ET	
@IP réseau	`164.81.128.0`	`10100100`	`01010001`	`10000000`	`00000000`		
@IP de D	`193.50.185.12`	`11000001`	`00110010`	`10111001`	`00001100`		
masque de S	`255.255.192.0`	`11111111`	`11111111`	`11000000`	`00000000`	ET	
@IP réseau	`193.50.128.0`	`11000001`	`00110010`	`10000000`	`00000000`		

id. réseau = (pour S) ; id. réseau = (pour D)

⋆ comparaison des **@IP réseau d'appartenance** des adresses de S et D :

`164.81.128.0`	`10100100`	`01010001`	`10000000`	`00000000`	
`193.50.128.0`	`11000001`	`00110010`	`10000000`	`00000000`	

◇ Si **égalité** alors la destination D est **dans le même réseau local** : ⟹ **routage direct**
Si ce n'est pas vrai alors il y a un problème de choix de l'adresse réseau : l'administrateur s'est planté !

◇ Si **différence** alors D **n'est pas dans le réseau local** ⟹ **routage indirect** :
⟹ envoi par **l'intermédiaire d'un routeur de sortie** du réseau local.

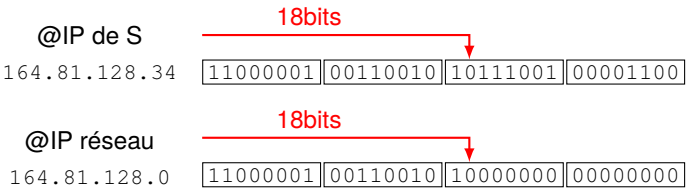
Sur l'exemple, `164.81.128.0` $\neq$ `193.50.128.0` ⟹ *routage indirect.*

Nouvelle version : la notion de préfixe CIDR : *Classless Inter-Domain Routing)*

Plus de notion de *classe* : on indique la répartition `<id. réseau><id. machine>` directement à l'aide d'un «`/n`».

Exemple : `164.81.128.34/18` où `/18` indique l'affectation des 18 premiers bits de l'adresse pour indiquer le réseau.

un préfixe «`/18`» correspond au masque : `255.255.192.0`



Si :

▷ (@IP réseau de S) = (@IP réseau de D suivant le préfixe de S) ⟹ **routage direct**.

▷ (@IP réseau de S) ≠ (@IP réseau de D suivant le préfixe de S) ⟹ **routage indirect**.

Mise en œuvre du routage direct

Utilisation du réseau à diffusion et donc de la fameuse @MAC et du protocole ARP…

Mise en œuvre du routage indirect

▷ il faut connaitre l'adresse d'un **routeur de sortie** ;

▷ il faut remettre le datagramme à ce **routeur** en **routage direct**.

Sous Linux

Commandes `ip address` & `ip link`

permet de connaître la configuration des interfaces actives : carte réseau, pont (bridge), interface virtuelle…

```
xterm
pef@solaris:~$ ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether 00:11:de:ad:be:ef brd ff:ff:ff:ff:ff:ff

pef@solaris:~/RESEAUX_I/FIREWALL$ ip address show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether 00:11:de:ad:be:ef brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.83/24 brd 192.168.42.255 scope global eth0
    inet6 fe80::211:deff:fead:beef/64 scope link
       valid_lft forever preferred_lft forever
```

Pour :

* activer une interface :
  `sudo ip link set dev eth0 up`
* désactiver :
  `sudo ip link set dev eth0 down`
* enlever les adresses IP associées à une interface :
  `sudo ip address flush dev eth0`

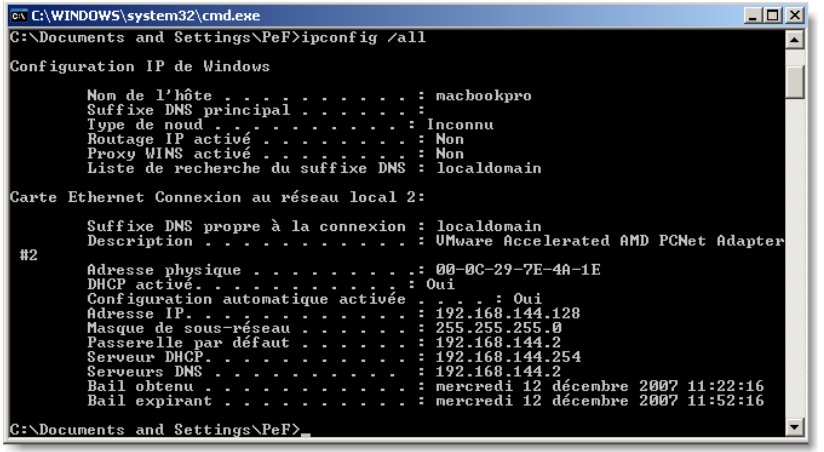
Commande `ip route`

```
xterm
pef@solaris:~$ ip route
137.204.212.128/25 dev bridge_dmz proto kernel scope link  src 137.204.212.129
137.204.212.0/25 dev bridge_internal proto kernel scope link  src 137.204.212.1
192.168.42.0/24 dev eth0  proto kernel scope link src 192.168.42.83  metric 1
169.254.0.0/16 dev eth0  scope link  metric 1000
default via 192.168.42.254 dev eth0  proto static
```

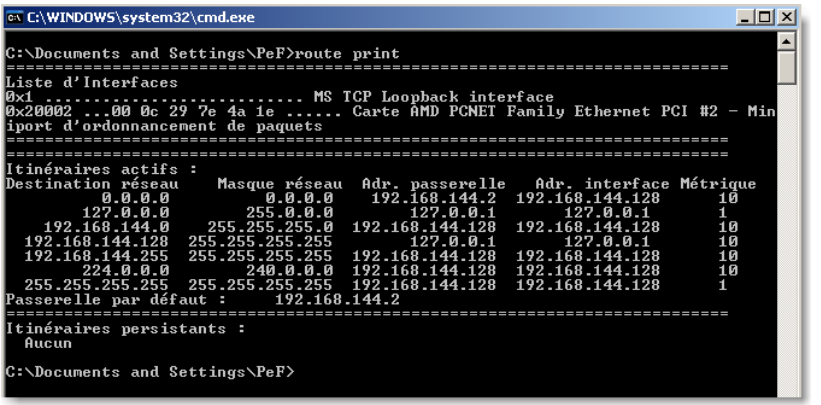
On remarque que la route par défaut pointe vers `192.168.42.254`, c-à-d que tout datagramme qui n'est pas destiné à un réseau connu de la table de routage sera transmis vers le routeur associé à cette adresse.

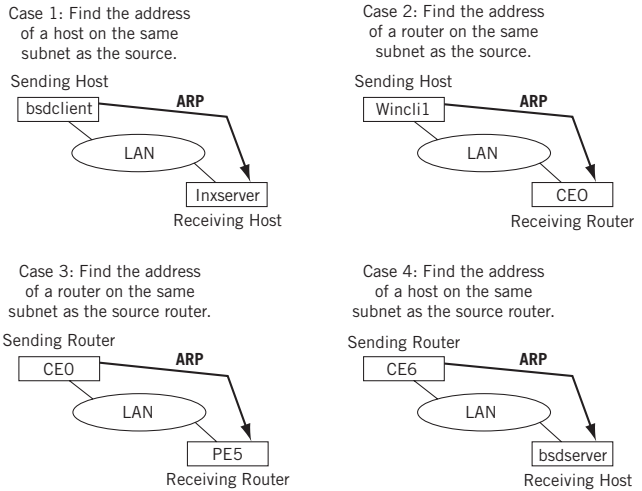
Sous Windows

Commande `ipconfig`



Commande `route print`





1. **hôte vers hôte** : l'émetteur est un hôte qui veut envoyer un paquet à un autre hôte dans le même réseau local.
   Dans ce cas, l'@IP de destination est connue et l'@MAC de destination doit être trouvée.

2. **hôte vers routeur** : l'émetteur est un hôte et veut envoyer un paquet à un autre hôte **n'appartenant pas** au réseau local.
   Il doit consulter la table de routage, «*forwarding table*» ou «*Forwarding Information Base*», pour trouver l'@IP du routeur.
   L'@IP du routeur est connue et l'@MAC du routeur doit être trouvée.

3. **routeur vers routeur** : l'émetteur est un routeur et veut «faire suivre» un paquet à un autre routeur connecté dans le même réseau local.
   La table de routage est utilisée pour trouver l'@IP du routeur.
   L'@IP du routeur est connue et l'@MAC du routeur destination doit être trouvée.

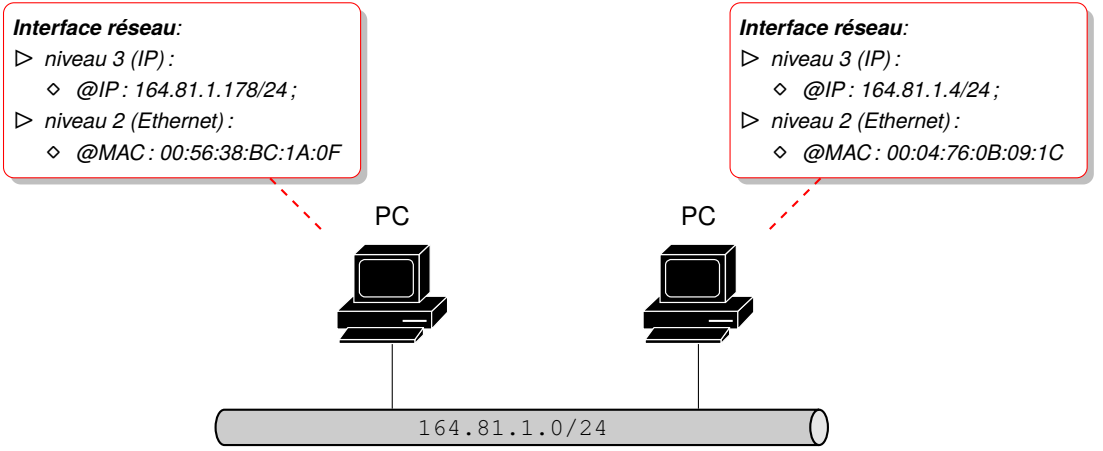
4. **routeur vers hôte** : l'émetteur est un routeur et veut «faire suivre», «*forward*», un paquet vers un hôte dans le même réseau local.
   *L'@IP de l'hôte est connue, **elle est contenue dans le paquet** et l'@MAC de l'hôte doit être trouvée.*

Attention

«...dans le même réseau local» ! ⟹ ARP ne sert quand dans un réseau local.

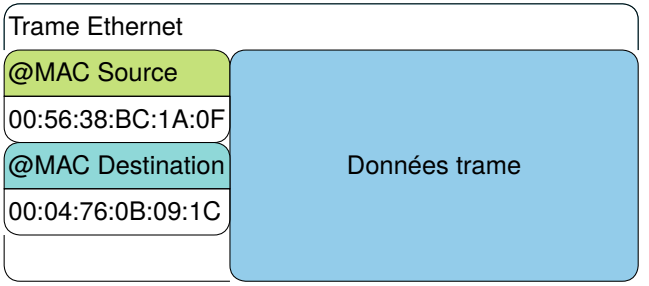
Chaque machine est identifiée par :

- une adresse de niveau 2 (@MAC) ;
- une adresse de niveau 3 (@IP) ;
- un réseau d'appartenance connu :
  - à l'aide du **préfixe** «`/n` indiquant n le nombre de bits de l'identifiant réseau»
  - ou à l'aide du **masque réseau**, *netmask*, adresse où chaque bit de l'identifiant réseau est à 1, les autres sont à 0.

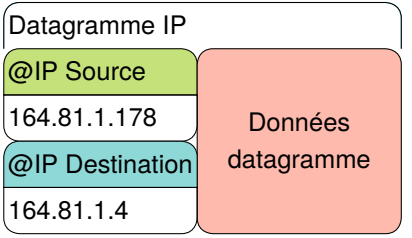


Pour être échangé, les datagrammes IP sont encapsulés dans des trames Ethernet

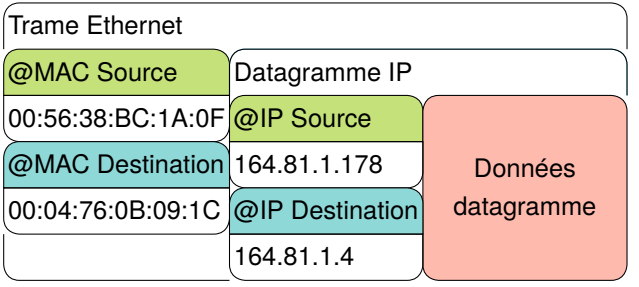
- la **trame** contient :
  - une @MAC source ;
  - une @MAC de destination ;



- le **datagramme** contient :
  - une @IP source ;
  - une @IP destination et des **données** ;



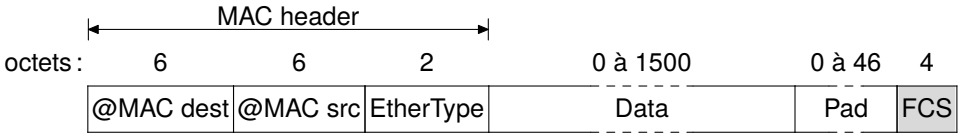
- la **trame encapsule** le datagramme IP :



Encapsulation en technologie Ethernet : les détails

Le réseau à diffusion utilise une technologie différente du réseau IP :

- ⋆ il dispose de **ses propres adresses** : @MAC ;
- ⋆ il utilise des messages sous un **format particulier** : la trame Ethernet ou IEEE 802.3 :



Où :

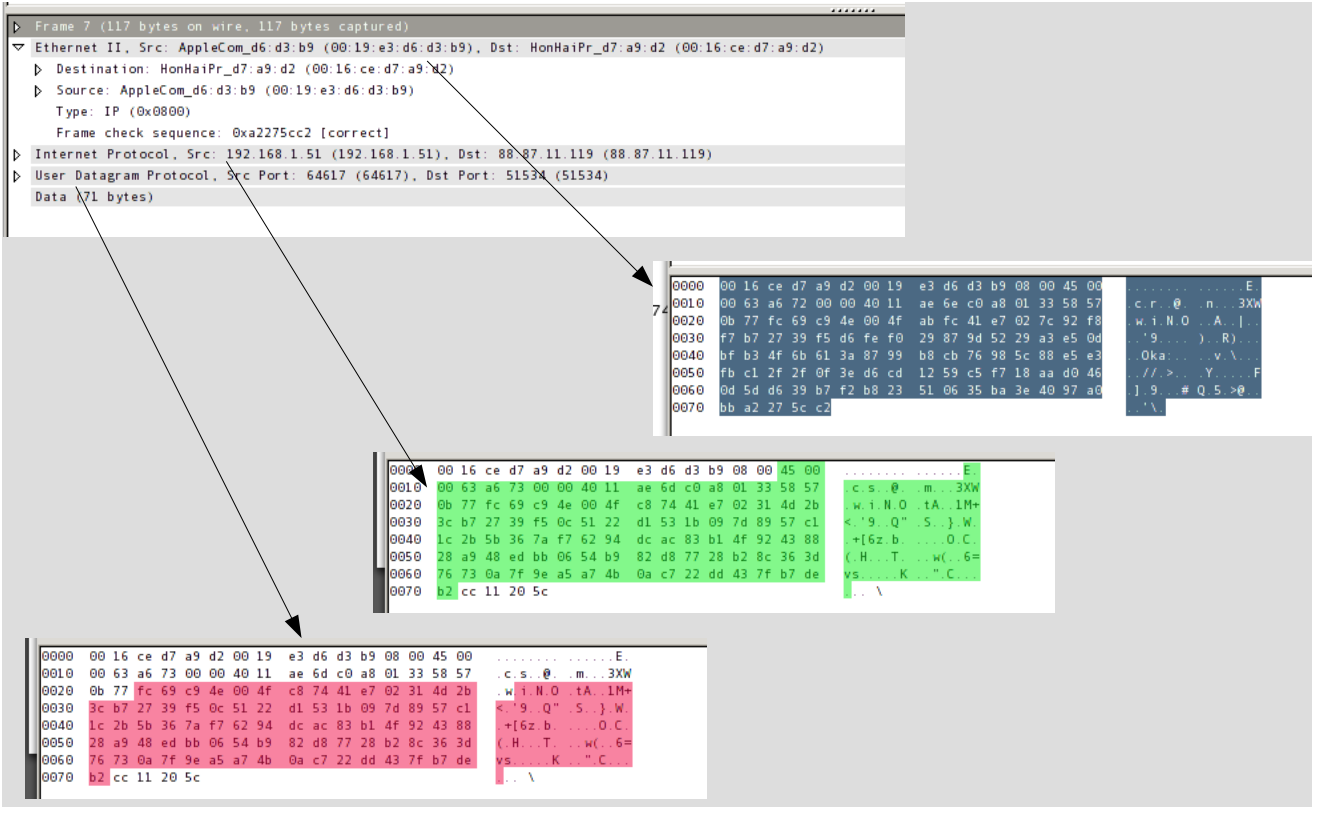
- ◇ **@MAC destination puis @MAC source** : ⟹ le récepteur détermine immédiatement s'il est destinataire ;
- ◇ **EtherType** (valeur>1536) :
  - ⋆ `0x0800` pour un datagramme IPv4, `0x08DD` pour un datagramme IPv6, `0x806` pour un message ARP, `0x888E` pour du 802.1x, `0x8100` pour du VLAN…
  - ⋆ **mais, si la valeur < 1500 ⟹ trame 802.3 avec LLC**, «Logical Link Control» ;
- ◇ **Data** : les données de la trame, complétées éventuellement par du bourrage (la longueur de ce champ est comprise entre 0 et 1500 octets) ;
- ◇ **PAD** ou bourrage : octets de bourrage sans signification, insérés **si la longueur du champ Data est insuffisante** (inférieure à 46 octets) ;
- ◇ **FCS**, *frame check sequence* ou Checksum : champ pour la **détection d'erreurs** (rarement transmis aux programmes).

Attention

- ⋆ Si le FCS est incorrect alors la trame n'est **pas transmise** par la carte réseau au système d'exploitation.
- ⋆ La trame a une taille de 64 à 1518 octets ($6+6+2+46+4=64$ à $6+6+2+1500+4=1518$).

Encapsulation en technologie Ethernet vu dans Wireshark

Un datagramme IP est contenu dans une trame, par exemple une trame Ethernet :

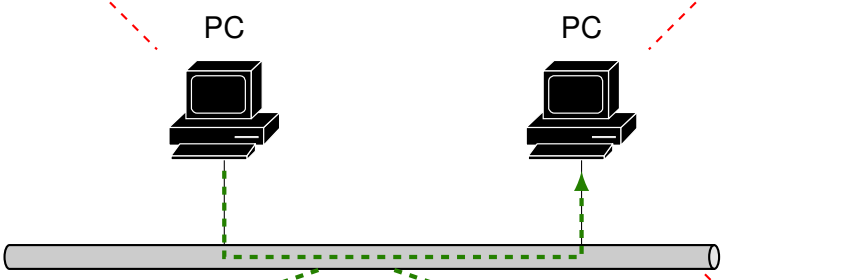


Le datagramme IP encapsule lui-même un datagramme UDP…

Routage direct illustré

***Interface réseau*:**
- *niveau 3 (IP) :*
  - *@IP : 164.81.1.178/24 ;*
- *niveau 2 (Ethernet) :*
  - *@MAC : 00:56:38:BC:1A:0F*

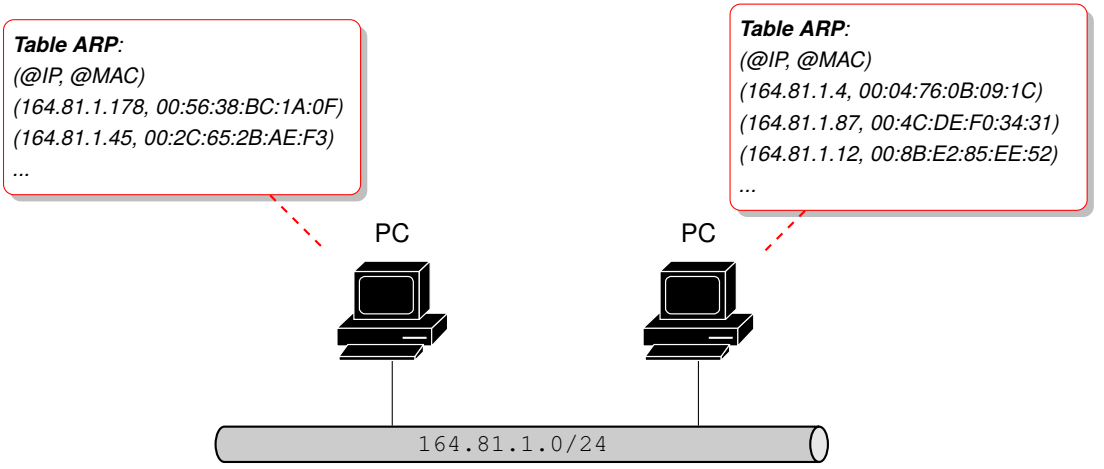
***Interface réseau*:**
- *niveau 3 (IP) :*
  - *@IP : 164.81.1.4/24 ;*
- *niveau 2 (Ethernet) :*
  - *@MAC : 00:04:76:0B:09:1C*



Trame Ethernet		
@MAC Source	Datagramme IP	
00:56:38:BC:1A:0F	@IP Source	Données datagramme
@MAC Destination	164.81.1.178	
00:04:76:0B:09:1C	@IP Destination	
	164.81.1.4	

Pour connaitre la correspondance entre adresse IP et adresse MAC :

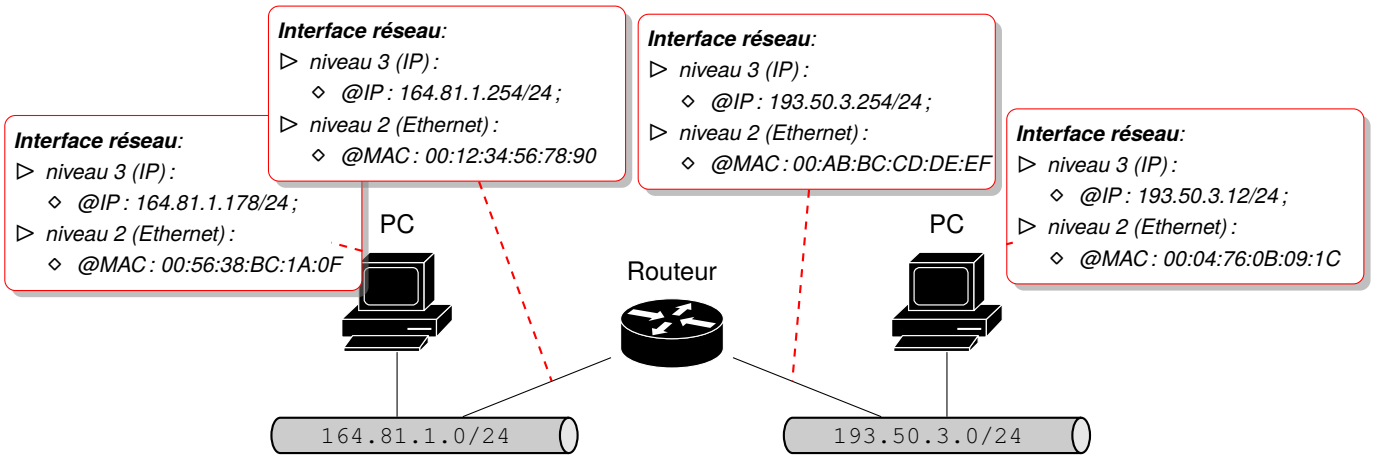
- ▷ mise en oeuvre du protocole ARP (Address Resolution Protocol) ;
- ▷ construction d'une table de correspondance entre @ IP et MAC sur chaque machine (cache ARP).



La **modification malveillante** *de cette table est possible : «ARP Spoofing» (usurpation d'identité), «ARP Cache Poisoning» (insertion d'association erronée).*

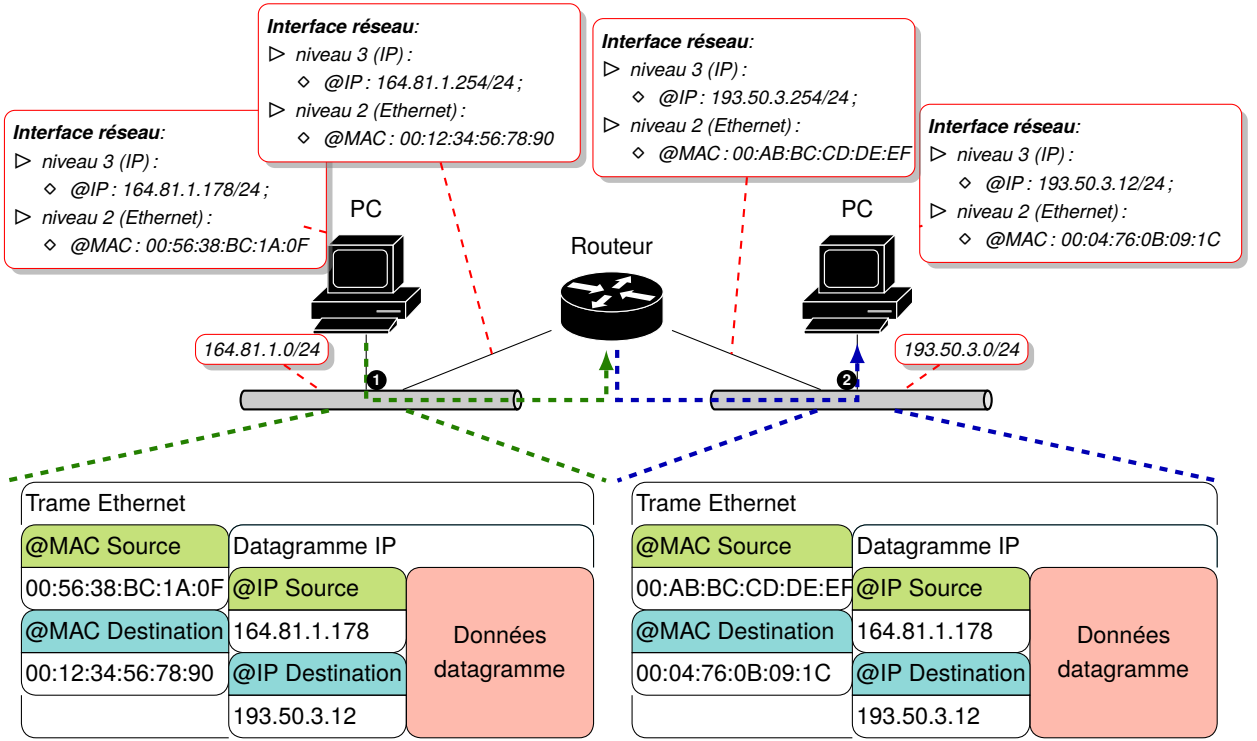
Routage indirect illustré

Le paquet de la machine `164.81.1.178` est routé par l'intermédiaire du routeur vers la machine `193.50.3.12`.



Le datagramme IP est **encapsulé** :

- par la machine `164.81.1.178`, dans une trame à **destination du routeur** ;
- puis, par le routeur, dans une nouvelle trame à destination de la machine `193.50.3.12`.



L'encapsulation permet la redirection vers le routeur sans modifier les @IP du datagramme.

Le DNS, «*Domain Name Server*» : Principe de délégation

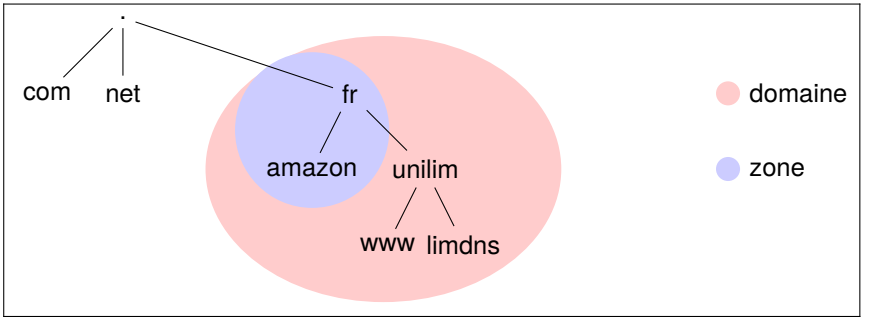
Le système DNS est entièrement distribué au niveau planétaire en utilisant la **délégation de domaine**.

À tout domaine est associé une **responsabilité administrative**.

Une organisation responsable d'un domaine peut :

- ▷ **découper** le domaine en sous-domaines ;
- ▷ **déléguer** les sous-domaines à d'autres organisations :
  - ◇ qui deviennent responsables du (des) sous-domaine(s) qui leurs sont délégué(s) peuvent, à leur tour, déléguer des sous-domaines des sous-domaines qu'elles gèrent.
  - ◇ *Le domaine parent contient alors seulement un pointeur vers le sous-domaine délégué;*

- ⋆ Les serveurs de nom enregistrent les données propres à une partie de l'espace nom de domaine dans une **zone**.
- ⋆ le serveur de nom à **autorité administrative** sur cette zone ;
- ⋆ un serveur de nom peut avoir **autorité** sur plusieurs zones ;
- ⋆ une **zone** contient les informations d'un domaine **sauf** celles qui sont **déléguées** :



Whols «unilim.fr»

```
darkstar:~ pef$ whois unilim.fr
%%
%% This is the AFNIC Whois server.
%%
%% complete date format : DD/MM/YYYY
%% short date format    : DD/MM
%% version              : FRNIC-2.5
%%
%% Rights restricted by copyright.
%% See http://www.afnic.fr/afnic/web/mentions-legales-whois_en
%%
%% Use '-h' option to obtain more information about this service.
%%
%% [2a01:0e35:8a71:bec0:66b9:e8ff:fed2:23ba REQUEST] >> unilim.fr
%%
%% RL Net [##########] - RL IP [#########.]
%%
domain:      unilim.fr
status:      ACTIVE
hold:        NO
holder-c:    UDL3-FRNIC
admin-c:     JPL1325-FRNIC
tech-c:      GRST1-FRNIC
tech-c:      NV70-FRNIC
tech-c:      GU245-FRNIC
zone-c:      NFC1-FRNIC
nsl-id:      NSL5796-FRNIC
registrar:   GIP RENATER
Expiry Date: 01/01/2016
created:     01/01/1995
last-update: 15/12/2014
source:      FRNIC

ns-list:     NSL5796-FRNIC
nserver:     limdns.unilim.fr [164.81.1.4]
nserver:     limdns2.unilim.fr [164.81.1.5]
nserver:     cnudns.cines.fr [193.48.169.40 2001:660:6301:301::2:1]
source:      FRNIC

registrar:   GIP RENATER
type:        Isp Option 1
address:     23-25 Rue Daviel
address:     PARIS
country:     FR
phone:       +33 1 53 94 20 30
fax-no:      +33 1 53 94 20 31
e-mail:      domaine@renater.fr
website:     http://www.renater.fr
anonymous:   NO
registered:  01/01/1998
source:      FRNIC

nic-hdl:     JPL1325-FRNIC
type:        PERSON
contact:     Jean-Pierre Laine
address:     Unvi. de Limoges
address:     123, Aveneu Albert Thomas
address:     87060 Limoges
country:     FR
phone:       +33 5 55 45 77 08
e-mail:      jean-pierre.laine@unilim.fr
registrar:   GIP RENATER
changed:     24/10/2013 nic@nic.fr
anonymous:   NO
obsoleted:   NO
source:      FRNIC
```

Le responsable du domaine : Jean-Pierre Laine

La requête DNS : traduction du «*nom de domaine*» en «*adresse IP*»

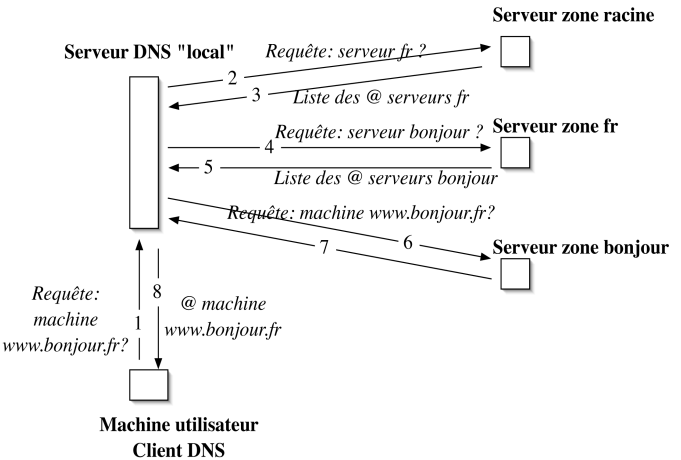
Modèle de fonctionnement client / serveur

- ▷ l'utilisateur utilise un **nom de domaine** dans une application (exemple : ping www.bonjour.fr) ;
- ▷ l'application cliente demande la **traduction du nom de domaine** auprès d'un serveur de nom (DNS) :
  cette opération s'appelle la « résolution de nom», ou «name resolver»
  - ◇ si le serveur connait la réponse il répond ;
  - ◇ sinon,
    - ⋆ s'il **fait autorité** pour le domaine demandé, alors pas de réponse (la machine n'existe pas) ;
    - ⋆ s'il ne fait pas autorité, le serveur de nom **interroge d'autres serveurs de nom** (de plus grand suffixe commun) jusqu'à ce que l'association nom de domaine / adresse IP soit trouvée ;
  - ◇ le serveur de nom retourne **l'adresse IP** au logiciel client : @IP de la machine «www» (il mémorise la réponse dans un cache et pour une certaine durée) ;

⟹ Le **logiciel client** contacte le **serveur**, comme si l'utilisateur avait spécifié une adresse IP.

Exemple :

```
xterm
$ socat - tcp:ishtar.msi.unilim.fr:6688
connexion établie avec 164.81.60.43
```



Resolver

Les «resolvers» sont les processus clients qui contactent les serveurs de nom Fonctionnement :

- contacte un « name server » (dont l'(les) adresse(s) est (sont) configurées sur la machine exécutant ce resolver) ;
- interprète les réponses et retourne l'information au logiciel appelant ;
- gère un cache (dépend de la mise en oeuvre).

Le serveur de nom interroge également d'autres serveurs de nom, lorsqu'il n'a pas autorité sur la zone requise (fonctionnement **itératif** ou **récursif**).

Si le serveur de nom est en dehors du domaine requis, il peut être amené à contacter un serveur racine.

Amélioration des performances

Mécanisme de cache dans les serveurs pour limiter le nombre d'interrogations

- évite la surcharge du réseau ;
- diminue les délais de réponse ;
- baisse la charge des serveurs de haut niveau (les serveurs racines).

Remplissage du cache lors des requêtes des clients et durée de vie limitée dans le cache

- TTL (Time To Live) spécifié dans les réponses pour éviter qu'une association soit conservée trop longtemps.

Types de serveur de nom

Serveur de nom **primaire** :

- maintient la base de données de la zone dont il a l'**autorité administrative**

Serveur de nom **secondaire** :

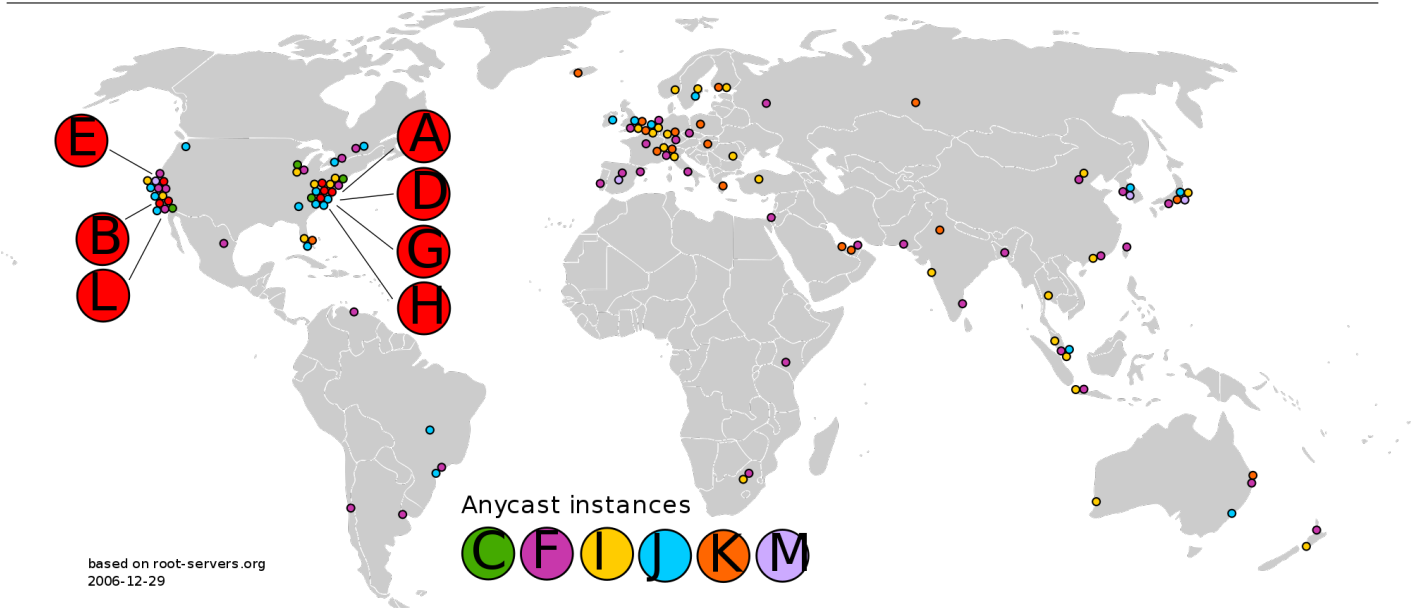
- interroge périodiquement le serveur de nom primaire et met à jour les données

Il y a **un** serveur primaire et généralement **plusieurs** secondaires.

La redondance permet la défaillance éventuelle du primaire et du (des) secondaire(s)

Un serveur de nom peut être primaire pour une (des) zone(s) et secondaire pour d'autre(s).

Les 13 serveurs racines



Le terme «anycast» permet d'offrir des services DNS de proximité à l'aide de cette capacité offerte par IPv6.

On peut remarquer que ce service de proximité permet même de délocaliser géographiquement les serveurs et améliore la disponibilité et la sécurité du système DNS.

- Configuration de l'adresse IP et du réseau de connexion :

xterm
```
root@solaris:~# ip address flush dev eth0
root@solaris:~# ip address show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:11:de:ad:be:ef brd ff:ff:ff:ff:ff:ff
root@solaris:~# ip address add 192.168.42.57/24 brd + dev eth0
root@solaris:~# ip address show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:11:de:ad:be:ef brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.57/24 brd 192.168.42.255 scope global eth0
root@solaris:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:11:de:ad:be:ef
          inet adr:192.168.42.57  Bcast:192.168.42.255  Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:2222687 erreurs:1723210 :7968 overruns:0 frame:0
          TX packets:915273 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:1148981328 (1.1 GB) Octets transmis:807289372 (807.2 MB)
          Interruption:19 Adresse de base:0x2024
```

- Configuration de la **route par défaut** :

xterm
```
root@solaris:~# ip route add default via 192.168.42.254
root@solaris:~# ip route
192.168.42.0/24 dev eth0  proto kernel  scope link  src 192.168.42.57
default via 192.168.42.254 dev eth0
```

- Tester le routage et trouver **l'interface de sortie** (celle considérée comme «*default*») :

xterm
```
root@solaris:~# ip route get 164.81.1.4
164.81.1.4 via 192.168.42.254 dev eth0 src 192.168.42.57
    cache
```

Il ne manquera plus qu'à configurer la résolution DNS avec le fichier `/etc/resolv.conf`.

- On édite le contenu de `/etc/network/interfaces`

xterm
```
root@solaris:~# more /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 192.168.42.57
netmask 255.255.255.0
gateway 192.168.42.254
```

- le fichier `/etc/resolv.conf` :

xterm
```
root@solaris:~# more /etc/resolv.conf
domain pefnet
search pefnet
nameserver 192.168.42.53
```

Le «search» indique le nom de domaine à ajouter au nom d'une machine entrée sans le FQDN (exemple : `msi` au lieu de `msi.unilim.fr`

- on relance le «service» réseaux :

xterm
```
root@solaris:~# /etc/init.d/networking restart
```

- configuration de l'interface pour utiliser le client DHCP, «*Dynamic Host*», dans le fichier `/etc/network/interfaces` :

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
```

- en ligne de commande :

xterm
```
$ sudo dhclient eth0
```

Le client DHCP configure l'interface, le fichier `resolv.conf` et la route par défaut si un serveur DHCP prend en charge sa demande.

Le serveur DHCP **présent dans le réseau local** répond et prend en charge la machine :

xterm
```
root@solaris:~# ip address show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 00:11:de:ad:be:ef brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.83/24 brd 192.168.42.255 scope global eth0

root@solaris:~# ip route
192.168.42.0/24 dev eth0  proto kernel  scope link  src 192.168.42.83
default via 192.168.42.254 dev eth0
```

La machine a reçu sa configuration IP : @IP et @IP de la passerelle.
Elle reçoit également la configuration des serveurs DNS à utiliser.

Distinction entre ordinateur et routeur

- ⋆ un **ordinateur** est un équipement relié à **un seul réseau** ;
- ⋆ un **routeur** est un équipement relié à **au moins deux réseaux**, éventuellement à un réseau où ne sont connectés que des routeurs (réseau d'interconnexion) ;
- ⋆ chacun dispose pour chaque connexion d'une carte réseau ;
- ⋆ chacun dispose pour chaque carte réseau d'une @MAC et d'une @IP ;
- ⋆ l'ordinateur est soit l'**expéditeur initial**, soit le **destinataire final** d'un datagramme ;
- ⋆ le routeur réémet, **relaie**,«*forward*», des datagrammes :
  - ◇ provenant d'une de ses interfaces (carte de connexion à un réseau) ;
  - ◇ vers une autre de ses interfaces ;
    c'est à lui de choisir une étape sur le chemin que devra emprunter le datagramme pour atteindre l'ordinateur destinataire.

Généralisation : algorithme de routage par sauts successifs, «*next hop routing*»

Le datagramme va passer d'intermédiaire en intermédiaire, d'une «entité réseau» à une autre :
il fait des «sauts» ou *hop* :

- ▷ l'entité réseau (ordinateur ou routeur) exécute le même algorithme : **décider entre routage direct et indirect**
  Dans le cas d'un routeur, le routage indirect peut faire le choix entre différentes adresses de routeurs.
- ▷ l'entité réseau doit déterminer **l'adresse de prochain saut**, c-à-d la prochaine étape du chemin d'acheminement du datagramme à transmettre.
  Un saut correspond à la transmission d'un datagramme à un routeur ou à la machine destinataire.

Acheminement des messages ou routage

Pour acheminer un datagramme de la source à la destination, il faut **déterminer un chemin** allant du réseau origine au réseau destinataire :

- ⋆ pour sortir du réseau origine, il faut un premier routeur (passerelle ou «gateway»)
- ⋆ il faut ensuite trouver le **routeur** qui est connecté au réseau destination.

Deux cas possibles :

1. le routeur destination est **directement accessible**, c-à-d le réseau destination est directement connecté au réseau origine par l'intermédiaire du même routeur ;
2. le routeur destination **n'est pas directement accessible** : le message doit circuler via un ou plusieurs routeurs intermédiaires.

Ce qui permet d'appliquer l'algorithme de base de recherche du prochain saut

- ▷ Routage **direct** : le datagramme est transmis à une machine dans le même réseau local ;
- ▷ Routage **indirect** : le datagramme est échanger entre routeurs jusqu'au réseau destination (pour le routeur connecté au réseau destination, la remise du datagramme se fait de manière directe).

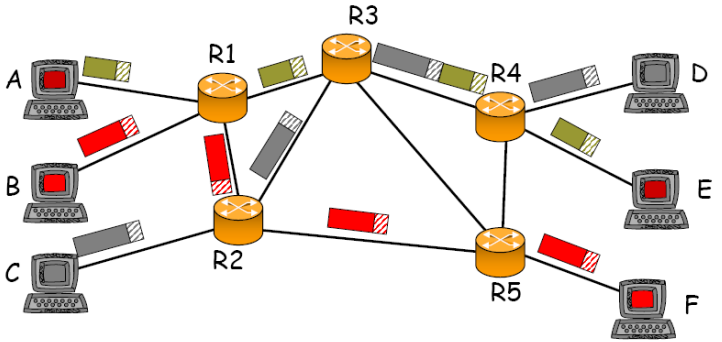
Comment trouver le routeur destination ?

En théorie, le routage devrait se faire en tenant compte de paramètres difficiles à évaluer comme l'encombrement du réseau, la longueur du datagramme ou le type de service mentionné dans l'en-tête du datagramme.

En pratique, l'acheminement des datagrammes se fait en fonction :

- ⋆ de la connaissance par un routeur, des autres routeurs auxquels il est connecté : **table de routage** ;
- ⋆ d'hypothèses statiques utilisées dans des **algorithmes de calcul du plus court chemin** (*utilisation d'algorithmes pour construire la table de routage ou seulement de l'intelligence de l'administrateur réseau*).

Chaque routeur prend une **décision au mieux**, «*best effort*», pour l'étape de routage qu'il réalise.



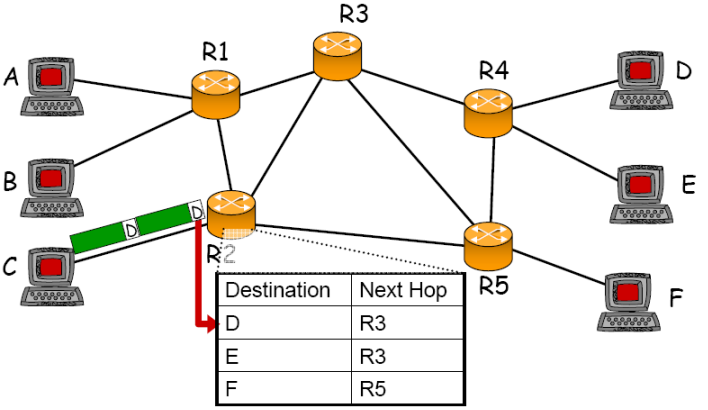
Le routage peut se faire suivant des routes **différentes.**

Ce qui explique que :

- ⋆ *certains datagrammes peuvent* **se perdre** *;*
- ⋆ *qu'ils arrivent dans un ordre différent (***temps d'acheminement différents***).*

Fonctionnement du routage sur un routeur :

- ▷ il faut connaître des routeurs destinations pour accéder à d'autres réseaux ;
- ▷ ces routeurs sont indiqués dans une **table de routage** ;
- ▷ chaque entrée de la table contient :
  - ◇ un **réseau de destination** ;
  - ◇ une **adresse de prochain saut**, *next hop* :
    - ⋆ celle du prochain routeur à emprunter pour atteindre la destination (routage indirect) ;
    - ⋆ celle de l'interface du routeur s'il est connecté au réseau (routage direct).
- ▷ la table de routage contient une **route par défaut** (pour les destinations inconnues).



Destination	Next Hop
D	R3
E	R3
F	R5

R2 atteint D par l'intermédiaire de R3 :
⟹R3 est son «next-hop» sur le chemin vers D.

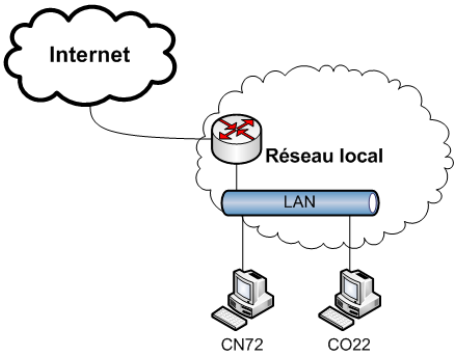
Comment créer une méthode de routage «simple» pour un ordinateur ?

- Appliquer le principe de «*localité*» : connaissance locale $\Longrightarrow$ décision locale ;
- Ne considérer que le réseau auquel est connecté l'ordinateur ;
- Prendre en compte que c'est un réseau à diffusion ;
- Savoir que ce réseau est interconnecté par un réseau point à point aux autres réseaux (existence d'une sortie) ;
- Ignorer l'identité de tous ces autres réseaux (cette connaissance sera réservée aux routeurs).

Alors, cette méthode simple ?

- La seule possibilité de sortir du réseau local ? un **routeur** connecté à ce réseau ;
- **Conclusion** :
  - soit l'ordinateur **communique avec le réseau local** et il le fait suivant la méthode d'un **réseau à diffusion** ;
  - soit l'ordinateur **communique avec l'extérieur** et il **passe par le routeur** !

    *Ce routeur est souvent appelé «passerelle» ou «*gateway*» ou «route par défaut»*
  - **l'ordinateur doit connaître l'adresse IP de ce routeur !**



Lorsqu'un hôte envoie un datagramme, il a le choix du mode :

- **Unicast** : pour un seul destinataire ;
- **Brodcast** : à tous les noeuds connectés à un réseau ;
- **Multicast** : à tous les noeuds qui appartiennent à un même groupe.
  Le multicast dans IPv4 est limité au réseau local, mais peut être retransmis par un routeur s'il est configuré pour le faire.

Une **classe** est réservée à la définition d'adresse multicast :

▷ la classe de réseau D : `224.0.0.0/4` pour la plage d'adresses `224.0.0.0` à `239.255.255.255`.

Il existe des groupes prédéfinis :

Adresse	Destinations
224.0.0.1	All hosts on a subnet
224.0.0.2	All routers on a subnet
224.0.0.5	All OSPF routers (DR Others)
224.0.0.6	All OSPF Designated Routers
224.0.0.9	All RIPv2 routers
224.0.0.10	All EIGRP routers
224.0.0.12	DHCP Server
224.0.1.1	NTP
224.0.1.39	Cisco RP Announce
224.0.1.40	Cisco RP Discovery
224.0.0.251	Multicast DNS

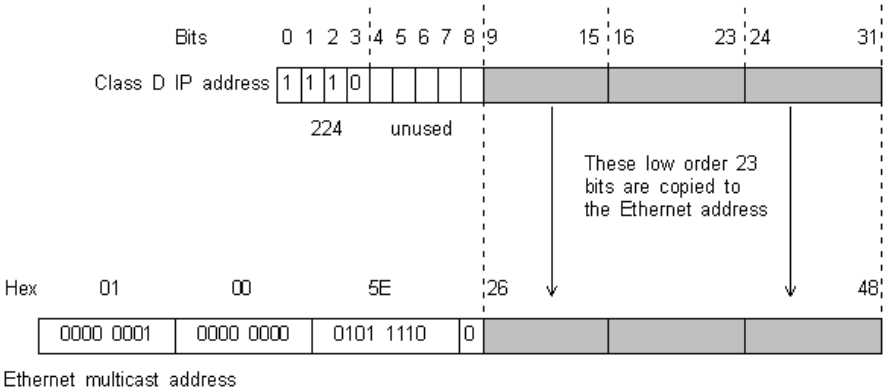
Il est *souhaitable* de réduire la diffusion d'un datagramme envoyé à un groupe d'hôtes à ces seuls hôtes. Ainsi, lors de la réception du datagramme :

- ⋆ l'hôte fait partie du groupe destinatire : il traite le datagramme ;
- ⋆ l'hôte ne fait pas partie du groupe : il ignore le datagramme.

Comment faire ?

Utiliser une @MAC destination particulière :

a. utiliser un préfixe particulier pour ce type d'@MAC : `01 00 5E` ;
b. mapper les 23 derniers bits de l'@IP du groupe, sur l'@MAC.



Il peut y avoir des risques de chevauchement sur les 5 bits ignorés mais c'est rare donc sans risque…

Attention

Sous IPv6, le broadcast et la diffusion de groupe change avec une @MAC de la forme `33:33:00:00:00:01` pour le «groupe» `ff02::1`

Pour pouvoir utiliser le «multicast» dans un programme, il est nécessaire d'informer le système d'exploitation que l'on veut joindre un groupe, ce qui est fait par les instructions suivantes en Python :

```
gestion_mcast = struct.pack("4sl", socket.inet_aton("224.0.0.127"), socket.INADDR_ANY)
ma_socket.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, gestion_mcast)
ma_socket.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_LOOP, 0)
```

Attention : dans la chaîne de format du `pack`, c'est la lettre «l» et non le chiffre «1».

L'option «`IP_MULTICAST_LOOP`» permet de choisir si l'émetteur reçoit le paquet qu'il envoie.

Si la machine dispose de plusieurs interfaces, il faut utiliser l'option «`IP_MULTICAST_IF`» pour la sélectionner.

Multicast & Routeur : protocole IGMP & TTL

Lorsque la machine rejoint un groupe, elle **diffuse un message au format IGMP**, «*Internet Group Management Protocol*», afin de prévenir les routeurs présents dans le réseau :

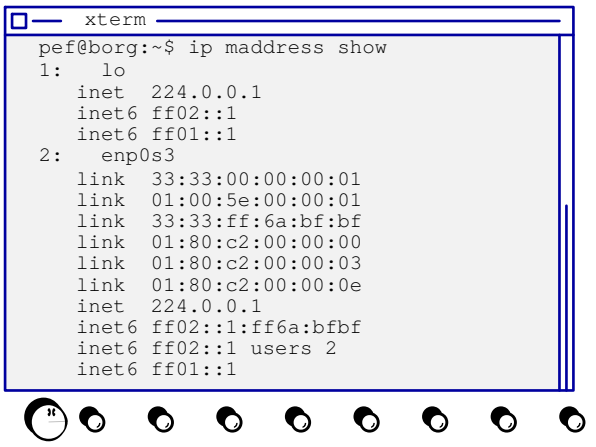
- que la machine joint un groupe (un paquet sera également diffusé lorsque la machine quitte le groupe) ;
- qu'ils pourront avoir à relayer des paquets multicast à destination de l'adresse du groupe choisi.

Pour restreindre le nombre de routeurs pouvant être traversé, on choisit la TTL, «Time To Live» des paquets multicast :

```
ma_socket.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 2)
```

Ici, le paquet pourra traverser uniquement 1 seul routeur.

Pour afficher la **liste des groupes auxquels appartient** la machine :

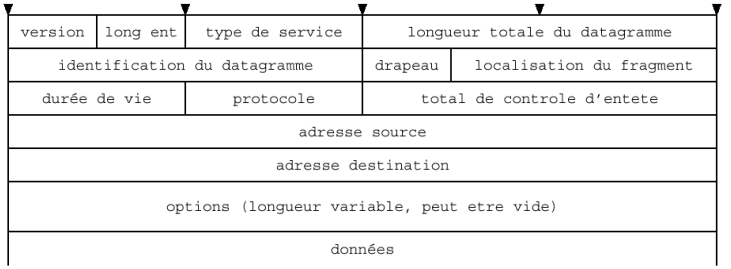


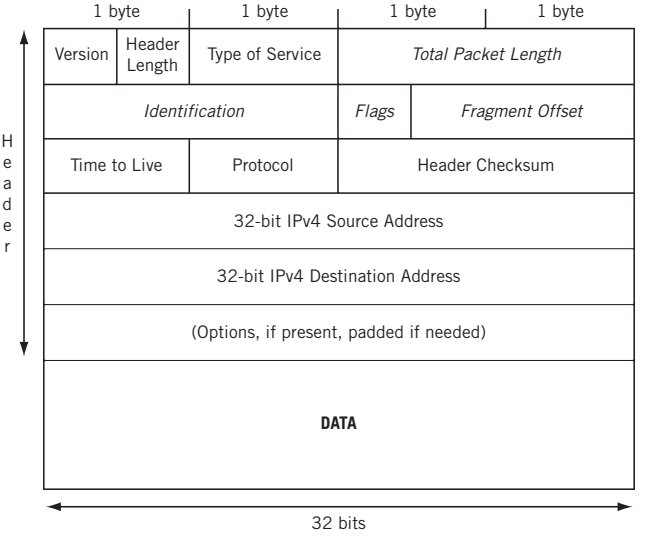
6 Le format du datagramme IP

Le datagramme IP est une séquence d'octets, dont l'**interprétation** est réalisée :

- ▷ soit par groupe de 2 ou de 4, pour obtenir une valeur sur 16 ou 32 bits ;
- ▷ soit bit par bit, où chaque bit à un sens particulier.

Sur le schéma en français, la « fléche » vers le bas indique une coupure du datagramme tous les octets, chaque ligne représentant 4 octets ou 32 bits.





- ◦ **version** sur 4 bits : le numéro de version (en général 4, mais bientôt 6…).
- ◦ **longueur de l'en-tête** sur 4 bits : longueur en nombre de **mots de 32bits** (une en-tête est au minimum d'une longueur de 20 octets, mais peut être plus grande en présence d'options).

- le **type de service** sur 8 bits : indique la manière dont doit être géré le datagramme par les routeurs :

0 1 2 3	4	5	6	7
priorité	D	T	R	C

  - le champ priorité varie de : 0 (priorité normale) à 7 (priorité maximale : supervision du réseau) ;
  - ne s'applique qu'à l'intérieur d'un réseau sous même administration et n'est pas pris en compte par tous les routeurs ;
  - les 4 bits D, T, R & C spécifient ce que l'on veut privilégier (RFC 1349, remplacée par la 2474) : D minimiser le délai d'acheminement, T maximiser le débit de transmission, R fiabilité, C coûts de transmission, DTRC tout à 1 pour la sécurité.

  En fonction des différents services :

application	minimise le délai	maximise le débit	maximise la fiabilité	minimise le coût
telnet/rlogin	1	0	0	0
FTP				
contrôle	1	0	0	0
transfert	0	1	0	0
NNTP	0	0	0	1
SNMP	0	0	1	0

  Actuellement :

  - on recycle les bits 6&7 pour l'ECN, *Explicit Congestion Notification*, définie dans la RFC 3168.
    Cela permet de détecter l'entrée en congestion du réseau, c-à-d qu'il va bientôt saturer.
  - DSCP, «*Differentiated Services Code Point*» : RFC 2474, proposée par CISCO pour faire de la QoS.

- **longueur totale** sur 2 octets : contient la taille en octet du datagramme (inclus la taille de l'en-tête)

- **identification, drapeaux & déplacement de fragments** : gestion de la fragmentation du datagramme IP.

- **durée de vie** (TTL) : indique le nombre de routeurs que peut traverser le datagramme avant d'être détruit.
  La valeur d'initialisation est de 128.
  - ◇ lorsqu'un routeur reçoit un datagramme avec la valeur 0, le datagramme est détruit et envoie à l'expéditeur un message ICMP, «*Internet Control Message Protocol*» pour l'informer.
  - ◇ permet de «**décongestionner**» automatiquement le réseau : un paquet ne peut rester bloqué dans le réseau indéfiniment.

- **protocole** sur 8 bits : indique le protocole de haut niveau qui est contenu dans le datagramme

  `1 ICMP 6 TCP 2 IGMP 17 UDP`

- **somme de contrôle d'en-tête** (*header checksum*) : assure l'**intégrité** de l'en-tête.
  - ◇ complément à 1 de la somme des valeurs de l'en-tête, considérées comme une suite d'entiers sur 16 bits ;
  - ◇ permet de faire de la «**détection d'erreur**» : le récepteur peut savoir si le datagramme a été endommagé pendant la transmission.

- **adresses IP** source et destination
  - ◇ permet de faire le routage :
    - ⋆ chaque routeur choisie une ligne de sortie en fonction de l'adresse de destination ;

- **options** : c'est une liste de longueur variable, mais toujours complétée par des bits de bourrage, pour obtenir une taille multiple de 32 bits (la taille de l'en-tête étant exprimée en mots de 32bits).
  - ◇ très peu utilisées : information concernant l'enregistrement de la route, estampille horaire…
  - ◇ la présence d'option est **déduite** par la différence de la taille de l'en-tête avec la taille de l'en-tête minimale :
    - ⋆ $HL - 20 = 0$, pas d'options
    - ⋆ $HL - 20 > 0$, présence d'options

Problème

Une trame a une **taille limite** de 1500 octets en Ethernet 10 et 100 Mbits (plus grande en gigabits).
Si la taille du datagramme à encapsuler est **supérieure** à 1500 octets $\Longrightarrow$ le datagramme **ne peut pas être** encapsulé !

Solution

Découper le datagramme IP !

Problème

Un **morceau** de datagramme IP **n'est pas** un datagramme IP !
Si le morceau doit traverser un routeur $\Longrightarrow$ il doit être vu comme un datagramme IP.

Le procédé doit être **reproductible** : si un morceau doit passer dans un réseau incapbale de le transporter, il doit être découpé de nouveau.
Si on a découpé le datagramme IP initial en **deux morceaux***, le deuxième morceau doit posséder les entêtes nécessaires à son routage*.

Solution

- ▷ Le datagramme IP doit être **fragmenté** avant d'être transmis.
- ▷ La **fragmentation** d'un datagramme IP doit donner des datagrammes IP avec toutes les entêtes nécessaires !

Encapsulation du datagramme IP ⇒ fragmentation ?

Unité de transfert du réseau

Un datagramme IP a une **taille maximale** de 65535 octets (*taille indiquée sur 16bits*).
Ce datagramme lors de son acheminement doit emprunter des réseaux dont la taille maximale des paquets transportables peut être inférieure.
La **taille maximale** d'une trame d'un réseau est appelée MTU, «*Maximum Transfer Unit*».
Cette MTU dépend de la nature du réseau et peut exprimer la taille de la trame ou de son contenu.

Link Protocol	Typical MTU Limit	Maximum IP Packet
Ethernet	1518	1500
IEEE 802.3	1518	1492
Gigabit Ethernet	9018	9000
IEEE 802.4	8191	8166
IEEE 802.5 (Token Ring)	4508	4464
FDDI	4500	4352
SMDS/*ATM*	9196	9180
Frame relay	4096	4091
SDLC	2048	2046

Fonctionnement

Le datagramme IP est soit :

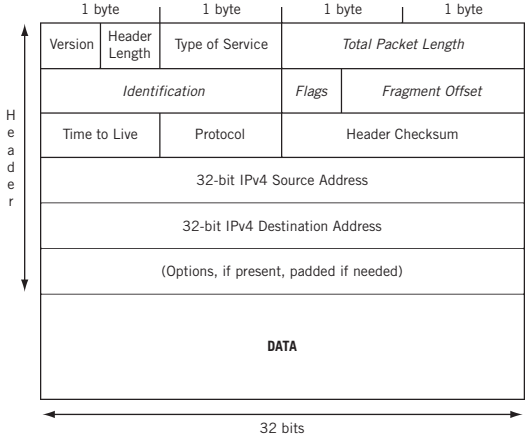
- **encapsulé** dans une trame si il est de taille inférieure ou égale au MTU ;
- **fragmenté** en plusieurs fragments dans le cas contraire, où :
  - chaque fragment doit avoir la **plus grande taille** possible et **multiple de 8 octets**.

Attention

Les **routeurs** d'un réseau **ne défragmentent pas** les fragments.
Ces fragments **n'empruntent pas forcément** tous le même chemin.

Un Fragment = Un datagramme IP

Lors de la fragmentation, le routeur renseigne l'en-tête de chaque fragment :



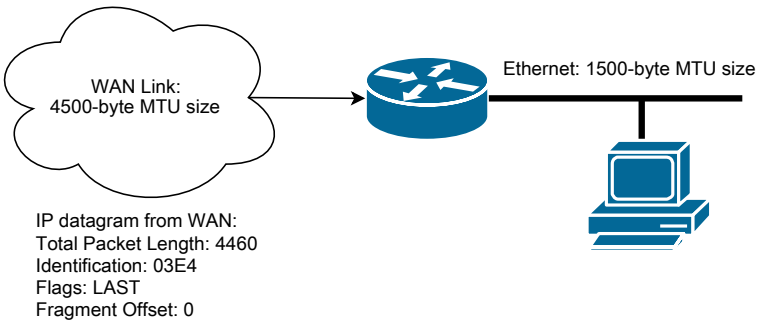
- *fragment offset*, déplacement : champ permettant de connaître la position du début du fragment dans le datagramme initial ;
- *identification* : numéro attribué à chaque fragment afin de permettre leur réassemblage : tous les fragments sont identifiés de la même manière.
- *total packet length* : il est recalculé pour chaque fragment ;

- *flags*, drapeaux : il est composé de trois bits
  - le 1er non utilisé ;
  - le 2nd, DF: *Don't Fragment* : autorise ou non la fragmentation.
    Si un datagramme a ce bit à un et que le routeur doit le fragmenter, alors le datagramme est détruit avec un message d'erreur ICMP.
  - le 3ème, MF: *More Fragments*, indique pour un fragment :
    - `0` : c'est le dernier ;
    - `1` : il reste des fragments après lui ;

Le processus de réassemblage

- ⋆ À la réception du premier fragment, le destinataire final déclenche un **temporisateur** de réassemblage ;
- ⋆ Ce temporisateur est un**délai maximal d'attente** de tous les fragments :
  - ◇ Si tous les fragments **n'ont pas été reçus** après ce délai d'attente, ils sont détruits et le **datagramme est ignoré**.
    le champs TTL de chaque fragment est décrémenté à intervalle régulier.
  - ◇ la **taille complète** du datagramme n'est connue que lors de la **réception du dernier fragment**.

Exemple où la MTU donnée est celle du contenu ou «*payload*»



Packet from LAN:	Frag1	Frag2	Frag3
Total Packet Length:	1500	1500	1500
Identification:	03E4	03E4	03E4
Flags:	MORE	MORE	LAST
Fragment Offset:	0	185	370

Est-ce que les «offset» sont corrects ?

«LAST» vient de l'interprétation de «MF=0».

Soit la trame suivante :

```
0000   00 22 AA 01 21 31 00 D0  F1 10 12 13 08 00 45 00   ."..!1........E.
0010   00 28 02 37 00 00 23 06  F9 23 C1 32 B9 12 C9 1B   .(.7..#..#.2....
0020   59 15 08 10 01 BB 00 00  00 00 00 00 00 00 50 02   Y.............P.
0030   20 00 E9 A1 00 00
```

fragmentée en :

```
0000   00 22 AA 01 21 31 00 D0  F1 10 12 13 08 00 45 00   ."..!1........E.
0010   00 1C 02 37 20 00 23 06  D9 2F C1 32 B9 12 C9 1B   ...7 .#../.2....
0020   59 15 08 10 01 BB 00 00  00 00                     Y.........
```

et :

```
0000   00 22 AA 01 21 31 00 D0  F1 10 12 13 08 00 45 00   ."..!1........E.
0010   00 1C 02 37 20 01 23 06  D9 2E C1 32 B9 12 C9 1B   ...7 .#....2....
0020   59 15 00 00 00 00 50 02  20 00                     Y.....P. .
```

Ce qui donne ?

Ces métriques correspondent à des mesures des performances d'un réseau

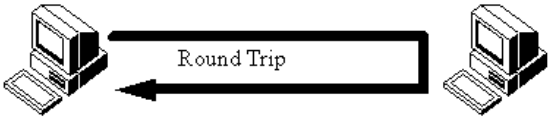
Elles peuvent être mesurées au cours du temps, ou seulement, à certains moments, ou bien encore, uniquement avec une valeur approchée.

La **latence** ou «*latency*»

- ⋆ définie par le temps de passage des données de l'émetteur vers le récepteur ;
- ⋆ pas toujours mesurable facilement (l'heure entre deux interlocuteurs n'est pas forcément synchronisée... et la synchroniser est difficile !)

Le **RTT** ou «*Round Trip Time*»

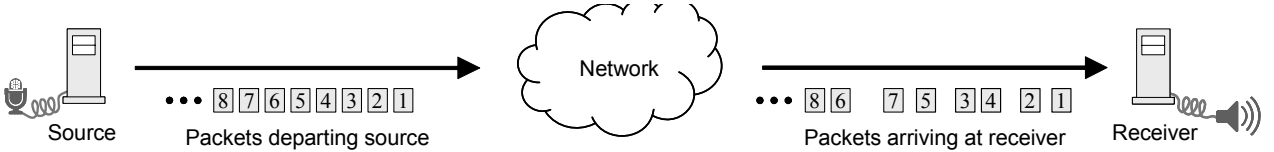
- ▷ mesure le temps pris pour obtenir une réponse après avoir envoyer une demande à un interlocuteur;
- ▷ permet d'obtenir une valeur approchée de la latence : $RTT/2$
  - ◇ la machine A envoie une «estampille» avec sa propre heure dans un paquet ;
  - ◇ la machine B répond avec un paquet contenant cette estampille ;
  - ◇ A mesure la différence de temps avec son heure actuelle.

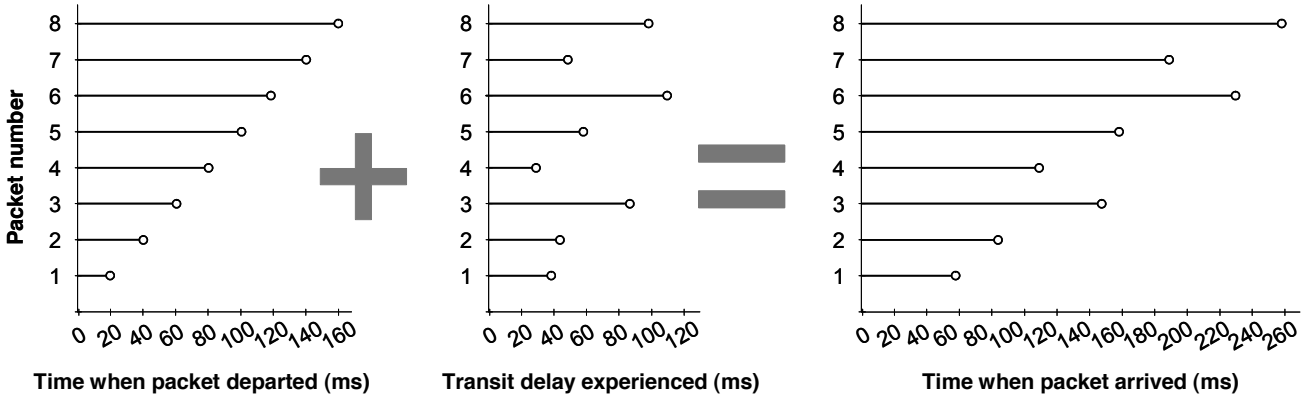


La **gigue** ou «*jitter*»

- ⋆ mesure la variation de la latence au cours du temps :
  $Jitter = latency(n) – latency(n-1)$, où la mesure $n-1$ est prise à un temps t et n à un temps $t + \delta t$
- ⋆ mesure importante pour les communications «temps réel» :
  Mieux vaut une forte latence et une gigue stable, qu'une faible latence et une gigue importante
  - ◇ Si la gigue est nulle : la latence est stable ; si elle varie (positive ou négative) : la latence varie.

Influence du jitter sur l'ordre de remise des datagrammes





La «commande» de mesure de la RTT, le «ping»

```
xterm
$ ping -c 5 164.81.1.4
$ ping6 fe80:1030:5329:6d2c:211:deff:fead:beef -I eth0
```

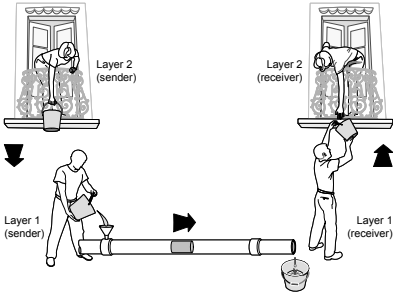
Attention : le «ping» est sensible à l'occupation de la machine cible : plus elle est occupée moins vite elle répond.

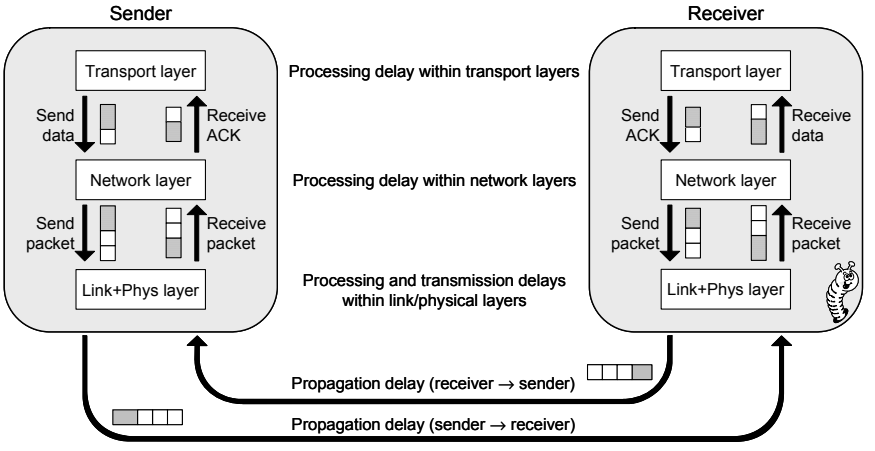
Quelques mesures de RTT

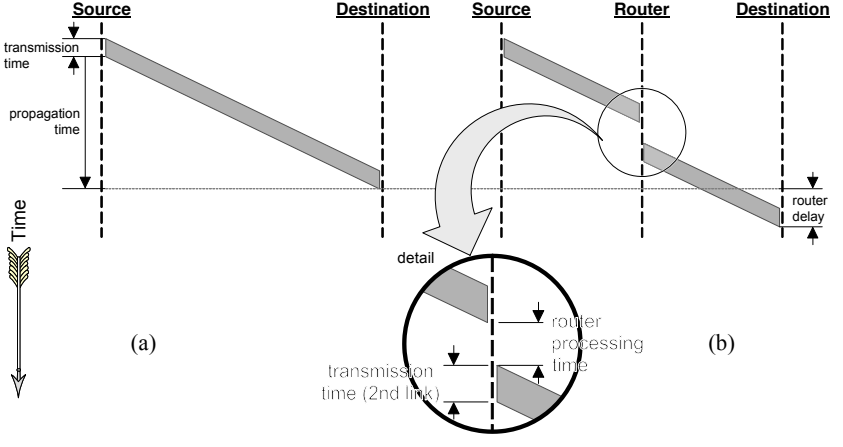
xterm

```
pef@solaris:~$ ping -c 3 164.81.1.4
PING 164.81.1.4 (164.81.1.4) 56(84) bytes of data.
64 bytes from 164.81.1.4: icmp_req=1 ttl=50 time=45.6 ms
64 bytes from 164.81.1.4: icmp_req=2 ttl=50 time=47.5 ms
64 bytes from 164.81.1.4: icmp_req=3 ttl=50 time=45.9 ms

--- 164.81.1.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 45.614/46.351/47.507/0.827 ms
pef@solaris:~$ ping -c 3 www.berkeley.edu
PING www.w3.berkeley.edu (169.229.131.81) 56(84) bytes of data.
64 bytes from webfarm.Berkeley.EDU (169.229.131.81): icmp_req=1 ttl=48 time=192 ms
64 bytes from webfarm.Berkeley.EDU (169.229.131.81): icmp_req=2 ttl=48 time=192 ms
64 bytes from webfarm.Berkeley.EDU (169.229.131.81): icmp_req=3 ttl=48 time=192 ms

--- www.w3.berkeley.edu ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 192.513/192.709/192.874/0.149 ms
pef@solaris:~$ ping -c 3 www.usyd.edu.au
PING solo-rproxy.ucc.usyd.edu.au (129.78.155.111) 56(84) bytes of data.
64 bytes from solo-rproxy.ucc.usyd.edu.au (129.78.155.111): icmp_req=1 ttl=41 time=424 ms
64 bytes from solo-rproxy.ucc.usyd.edu.au (129.78.155.111): icmp_req=2 ttl=41 time=447 ms
64 bytes from solo-rproxy.ucc.usyd.edu.au (129.78.155.111): icmp_req=3 ttl=41 time=347 ms

--- solo-rproxy.ucc.usyd.edu.au ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 347.922/406.958/447.972/42.790 ms
```


RTT : les délais induits par le réseau/par le traitement



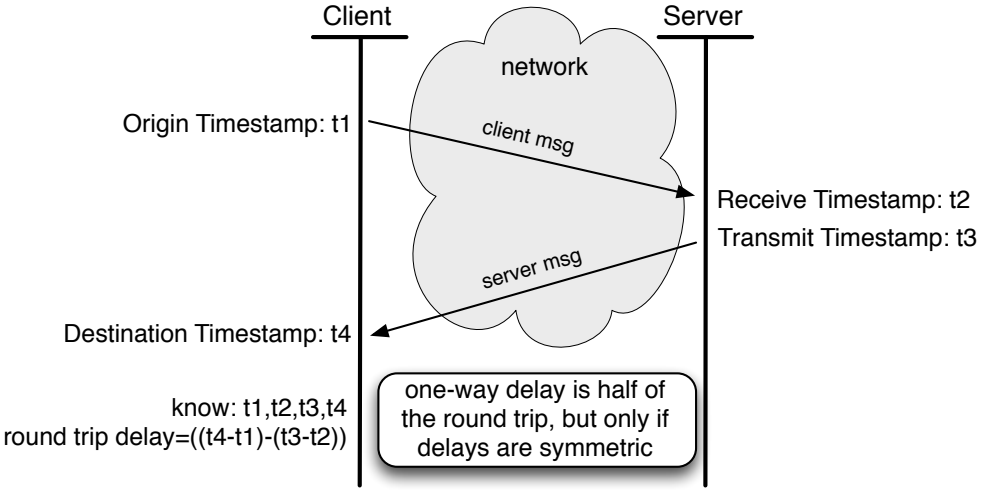




Le calcul du RTT : utilisation du «*Timestamp*»

Une demande d'«étiquette de temps» est réalisé dans le protocole (par exemple TCP) :

- le serveur et le client ont leur propre réglage d'horloge *sûrement* décalé ;
- le client envoie **l'heure courante** dans t_1 ;
- le serveur note t_1 et mesure t_2 l'heure d'arrivée du paquet ;
- le serveur *après traitement du protocole* envoie un message de réponse où il mesure t_3 et indique $t_3 - t_2$;
- le client mesure t_4, **l'heure d'arrivée** du message de réponse ;



- à la fin de l'échange, le **client** connait t_1, $t_3 - t_2$ et t_4 et peut calculer une **valeur approchée** du RTT !

Le **débit** :

- ⋆ mesuré en **bps** ou bits par seconde, exprimé en puissance de 10 (norme ISO) ;
- ⋆ correspond à la **quantité d'information** par unité de temps qui a été remis à son destinataire ;
- ⋆ Le **débit maximal** d'une ligne de transmission correspond à sa **capacité** :
  - ◇ peut être atteint sur des réseaux à **circuit vituels** (ou la perte de données est exclue) ;
  - ◇ peut être **égal** ou **inférieur** au débit « physique » du réseau, car il faut enlever les informations nécessaires au fonctionnement du réseau physique et aux erreurs (collision, attente de la disponibilité du support dans le cas de réseau en mode datagramme).
- ⋆ **L'occupation du canal** : pourcentage de la **quantité d'information** transmise par rapport au débit «physique» du réseau :

  un débit de 70Mbit/s dans un réseau Ethernet 100Mbit/s définit une occupation du canal de 70%
- ⋆ Le **débit utilisateur** : pourcentage de **données utilisateur** transmises par rapport à la quantité d'information transmise (informations d'adressages, de contrôle d'erreur, de contrôle, d'adaptation, *etc*)

La **QoS** ou «*Quality of Service*» :

- ◦ vise à **associer un débit** pour un type communication donné (VoIP, Web, Partage de fichier, *etc*) ;
- ◦ peut être **garantie** ou bien **souhaitée** ;
- ◦ suppose que les communications peuvent être **identifiées** et **classifiées**.

Rapport entre **débit** et **latence**, la notion de **vitesse perçue** :

- ▷ Si un camion transporte par route un stock de blu-rays de Limoges à Paris, la latence peut être grande (1j) mais le débit peut être énorme !
- ▷ Un réseau **faible débit** de **faible latence** est perçu **plus rapide** (Surfer sur le net en ADSL par exemple).

Gestion des erreurs

- ⋆ possibilité d'avoir des **erreurs** :
  - ◇ les **lignes de transmission** ne sont **pas parfaites** : modification des données entraînant une erreur ;
  - ◇ le **réseau interconnecté** n'est **pas parfait** : paquet perdu ou détruit.
- ⋆ choix parmi **différentes méthodes de contrôle d'erreur** entre l'émetteur et le récepteur :
  - ◇ le récepteur doit être **capable de déterminer** si un paquet reçu est **correct ou non** ;
  - ◇ le récepteur doit disposer d'un moyen **d'indiquer à l'émetteur** quels paquets ont été correctement reçus ou non (*accusé de réception, mécanisme d'attente, …*) ;
  - ◇ le récepteur doit pouvoir gérer des paquets n'arrivant **pas dans l'ordre** ou **manquant** (*gestion de buffer de réception, que faire si un paquet manquant n'arrive toujours pas ?*
  - ◇ l'émetteur doit-il renvoyer **tout de suite** un paquet ou attendre une **notification** du récepteur ?

Contrôle de flux

Un émetteur rapide peut saturer un récepteur lent.

Il faut éviter que les paquets ne pouvant pas être traité par le récepteur ne restent et finissent par **congestionner** le réseau global.

- ⋆ **asservissement** entre récepteur et émetteur ;
- ⋆ accord sur la **vitesse de transmission** ;
- ⋆ **intervention du routeur** avec les bits «ECN» (ICMP «*Source Quench*», RFC6633 de mai 2012⟶ *deprecated*).

Choix des chemins entre émetteur et récepteur

a. déterminer un **chemin possible** ;
b. améliorer ce chemin par le choix du **moindre coût** en fonction du trafic (métrique de débit par exemple) ;
c. tenir compte de **souhaits utilisateurs** comme choix du passage ou non par un pays suivant sa législation, le coût de communication, le débit possible… (utilisation de lignes louées à une TELCO, satellite, *etc*).

Circuit virtuel *vs* datagramme

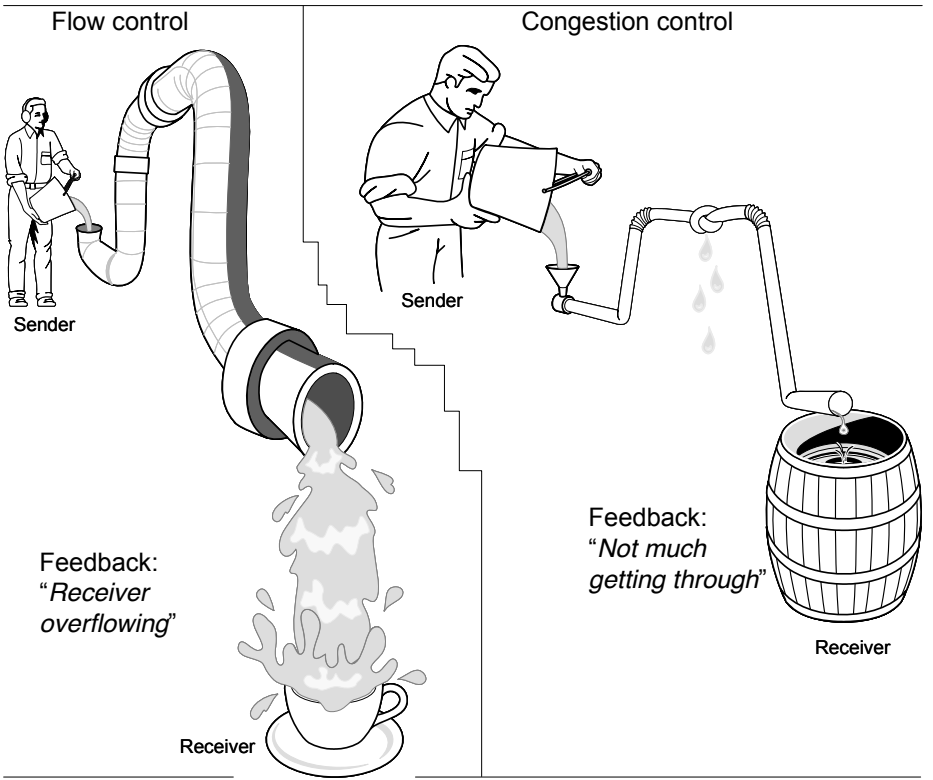
- Dans un réseau **point-à-point** en mode «**circuit virtuel**» :
  - le **débit obtenu** est très proche de celui offert en théorie par le réseau physique ;
  - lors de l'établissement du circuit virtuel, on peut connaitre le **débit prévisible** ;
  - le débit peut être **garanti**.

- Dans un réseau **point-à-point** en mode «**datagramme**» :
  - La capacité de la ligne de transmission est partagée avec les autres utilisateurs et ne peut être garantie :
    On **ne peut pas savoir** *ce que font les autres utilisateurs !*

  - Le **contrôle de flux** augmente l'utilisation du réseau et diminue son débit (envoi d'accusé de réception, retransmission, etc) ;

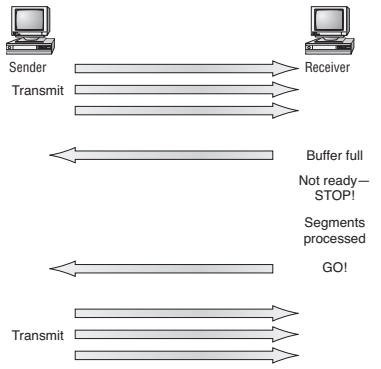
  - Des **pertes de paquets** peuvent se produire (si le réseau est en «*congestion*») ;

  - **Peu ou pas de QoS** :
    - uniquement sur des **parties contrôlées** : réseau de la même entreprise, passage par un FAI, …
    - en général :
      - on ne peut **pas prévoir à l'avance** le **chemin emprunté** par un datagramme et donc son temps d'acheminement ;
      - même si le chemin emprunté reste le même, le **taux d'occupation** de certaines parties de ce chemin varie suivant les usages des autres utilisateurs dont on ignore ce qu'ils font.



- La congestion dépend de l'état du **réseau global**, qui tient compte de toutes les communications entre tous les matériels connectés ;
- le contrôle de flux ne s'applique que sur **une communication** et entre deux interlocuteurs.

«**Éviter qu'un émetteur rapide sature un récepteur lent**» $\Longrightarrow$ le **récepteur contrôle l'émetteur**.



Utilisation de messages «*START/STOP*» :

- ⋆ lorsque le récepteur reçoit une «rafale» ou un «*burst*» de paquets : **bufférisation**, le problème est résolu si l'afflux de paquets tient dans le buffer ;
- ⋆ lorsque le récepteur reçoit **trop de paquets** et que son buffer est plein : risque de perte de paquets
  - ◇ le récepteur transmets à l'émetteur un **indicateur** «*Not ready*» pour stopper son émission ;
  - ◇ le récepteur transmets ensuite un **indicateur** «*Go*» pour autoriser de nouveau l'émetteur.

Exemple : le «*Flow control*» en technologie Ethernet :

- ▷ «***Half Duplex***» : contrôle de flux **implicite** :
  - ◇ CSMA/CD contrôle de flux uniquement du point de vue du support partagé :
    - ⋆ à la détection d'une collision attente d'un slot de temps choisi aléatoirement dans un intervalle agrandi à chaque collision (limité à 2^{10})
    - ⋆ erreur sur l'émetteur au 16$^{\text{ème}}$ essai (le récepteur ne détecte rien… *bien sûr*) ;
    - ⋆ **Sur un switch** : pour freiner un émetteur : créer des collisions, faire croire que le support est occupé ;
- ▷ «***Full Duplex***» : contrôle de flux **explicite** :
  - ◇ utilisation de «trame de pause» :
    - ⋆ envoyé par le récepteur à l'émetteur pour le stopper ;
    - ⋆ contient un paramètre indiquant le temps d'attente avant d'envoyer de nouvelles trames ;
    - ⋆ une nouvelle trame de pause avec un temps d'attente nul pour déclencher l'émetteur.

Du point de vue «Service» offert :

- service de transport **fiable**, en mode «**connecté**» ou «orienté connexion» :
  - étapes d'initialisation, d'échanges et de terminaison ;
  - garantie de remise des données au récepteur ;
  - garantie de respect de l'ordre de ces données ;
- autorise la gestion des données en **flots d'octets** (flux, ou *stream*) ;
- échanges **bidirectionnels simultanés** («*Full duplex*»).

Il offre un «mode connecté» **simulé** *dans un réseau mode datagramme.*

Attention

TCP **ne permet pas** :

- le **multicasting** (multi diffusion) ;
- le **broadcasting** (diffusion générale).

Du point de vue de la mise en œuvre

Une connexion correspond à :

- une **paire de points** ou d'extrémités de connexion ou « socket » :
  - chaque extrémité de connexion correspond à un TSAP : (@IP, n° port) ;
  - chaque connexion est identifiée par : [$\text{TSAP}_{\text{client}}$(@$\text{IP}_{\text{client}}$, n° $\text{port}_{\text{client}}$), $\text{TSAP}_{\text{serveur}}$(@$\text{IP}_{\text{serveur}}$, n° $\text{port}_{\text{serveur}}$)]

  Il n'y a pas d'autre identifiant (pas de numéro de circuit virtuel).
- la création de **deux flots de données inverses** et indépendants : $A \rightarrow B$ et $B \leftarrow A$;
- possibilité de **terminer un flot** dans un sens sans interrompre l'autre ;
- une mise en œuvre en deux étapes :
  - une **ouverture passive** : elle **accepte** une connexion entrante ;
  - une **ouverture active** : elle **demande** l'établissement de la connexion.

Mutliplexage de plusieurs communications grâce à la notion de numéro de port.

xterm

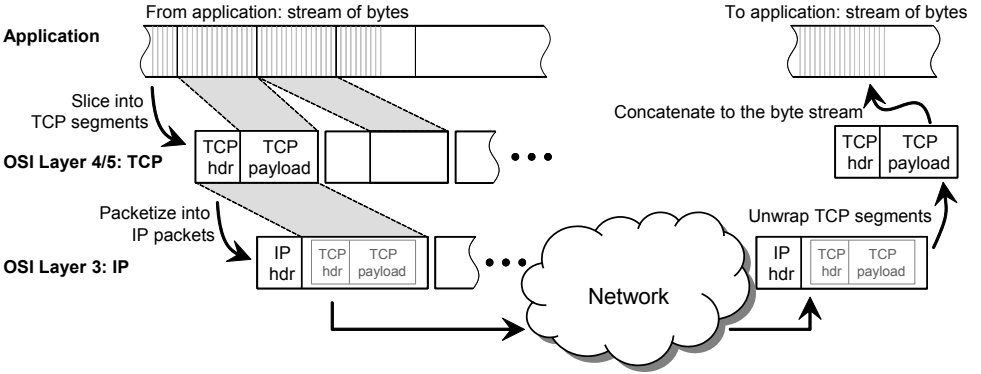
```
pef@solaris:~$ ss -tn
State      Recv-Q  Send-Q            Local Address:Port                 Peer Address:Port
ESTAB      0       0                 198.245.60.92:43506              198.211.120.59:443
FIN-WAIT-2 0       0                 198.245.60.92:35166               195.201.44.21:22067
CLOSE-WAIT 0       0                 198.245.60.92:46174                 164.81.1.97:443
ESTAB      0       0                 198.245.60.92:5222               109.190.127.78:55269
```

Segmentation

Les données transmises à TCP constituent un flot d'octets de **longueur variable**.

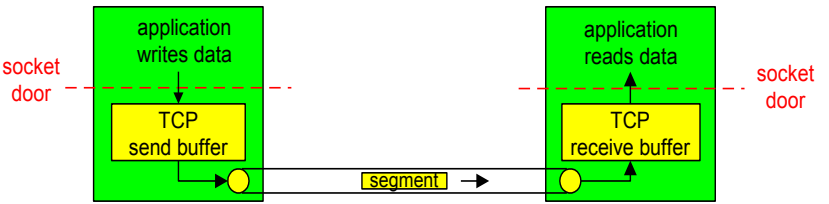
La transmission de ces données est bufférisée (mises dans un tampon) :

- les données **courtes** à envoyer et reçues sont **bufférisées** pour améliorer la communication :
  - à l'envoi : améliorer le taux d'occupation du réseau $\Longrightarrow$ *plus de données à envoyer à la fois* ;
  - à la réception : avertir/réveiller l'application que lorsqu'il y a un volume suffisant de données à traiter ;
- les données à envoyer, sont **fractionnées** en fragment de taille optimale pour TCP, appelés «segment» ;
  Cette fragmentation peut être contrôlée ou indépendante de l'application.
- chaque **segment** est émis dans un datagramme IP :

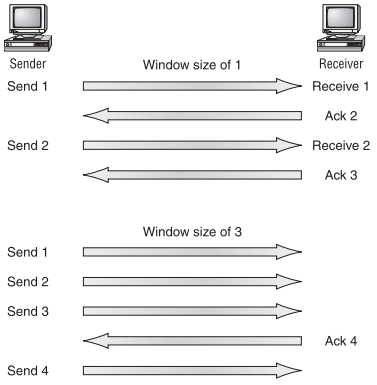


Du point de la programmation socket :

Les données de l'utilisateur passe au travers de la socket et sont segmentées automatiquement.

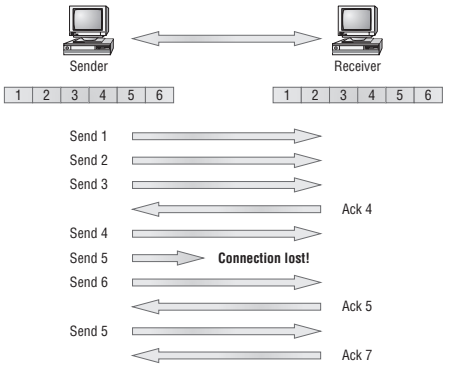


Le mécanisme de fenêtre⟹«*Contrôle de flux*»



- ⋆ **ne pas attendre** un acquittement avant de continuer à émettre ;
- ⋆ **optimiser** le temps d'accès au réseau :
  - ◇ la taille d'un paquet dépend de la technologie du réseau pas du récepteur ;
  - ◇ le réseau est rapide (beaucoup plus rapide que lorsque la taille du paquet a été choisie) ;
  - ◇ l'émetteur a remporté la compétition d'accès au support ;
  - ◇ le réseau lui est alloué ;
  - ◇ le temps d'allocation est supérieur au temps d'émission d'un paquet ;
- ⋆ **ne pas saturer le récepteur** :
  - ◇ le récepteur définit une **fenêtre d'autorisation d'envoi** ;
  - ◇ la taille de cette fenêtre définit le nombre d'octets pouvant être transmis par l'émetteur ;
  - ◇ la taille de cette fenêtre est **choisie par le récepteur**.

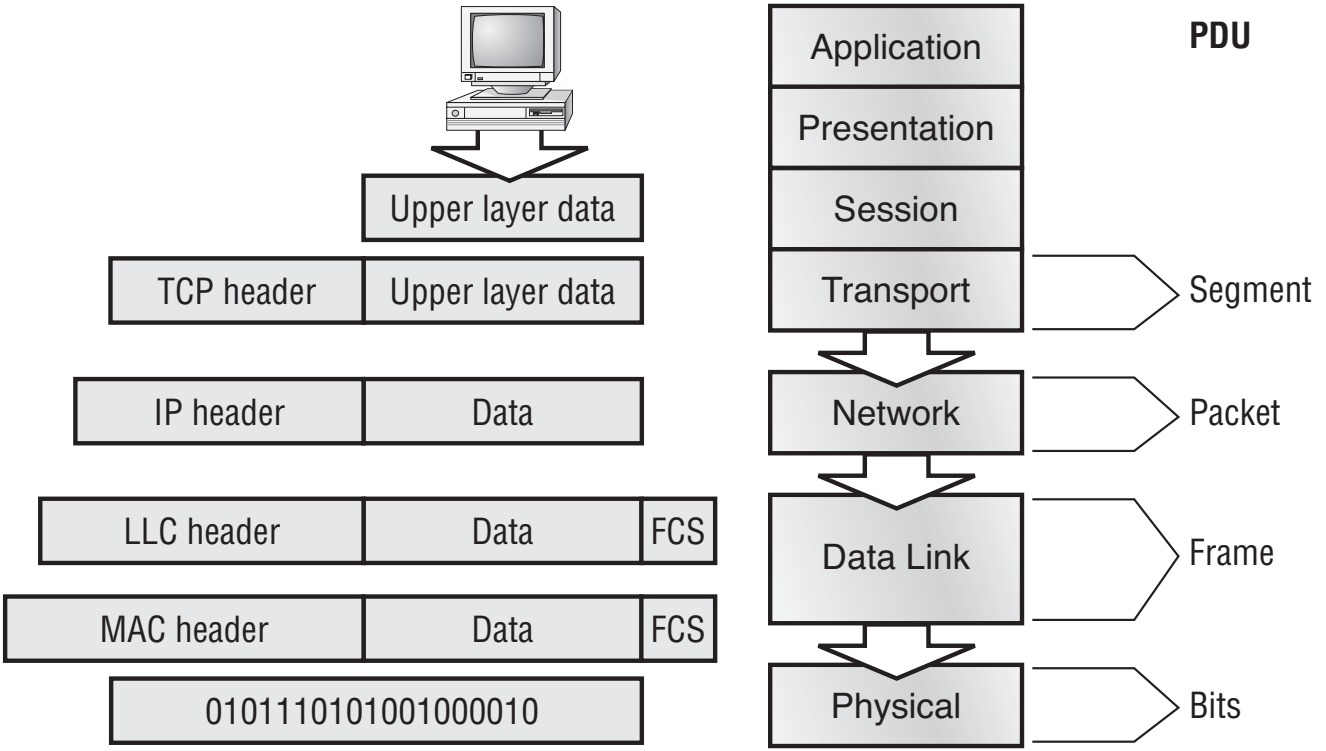
Le mécanisme d'acquittement positif avec retransmission⟹«*Correction des pertes*»



- ⋆ garantie l'absence de perte et de duplication des segments ;
- ⋆ le récepteur envoie un acquittement en retour à l'émetteur lorsqu'il reçoit un segment ;
- ⋆ l'émetteur :
  - ◇ envoie un segment et attend l'acquittement avant d'envoyer le suivant ;
  - ◇ lors de l'envoi, un **compte à rebours** démarre : s'il expire avant la réception de l'acquittement, le segment est renvoyé ;
- ⋆ **Si on combine avec le mécanisme de fenêtre** :
  - ◇ l'émetteur envoie plusieurs segments suivant la taille de la fenêtre ;
  - ◇ le récepteur envoie un acquittement demandant le **segment suivant de tous ceux de numéros consécutifs** qu'il a bien reçu.

Encapsulation du segment TCP

Pour généraliser le traitement des différentes couches, on parle de PDU, «*Protocol Data Units*».

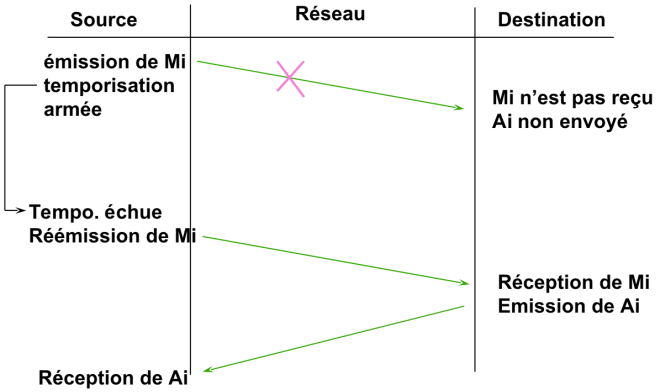


Gestion des erreurs

TCP garantit l'arrivée des données, en cas de perte les deux extrémités sont prévenues :

- ⋆ utilisation d'un mécanisme d'acquittement :
  - ◇ lorsque la source S envoie un message M_i vers une destination D,
  - ◇ S attend un acquittement A_i de D avant d'émettre le message M_{i+1}.

- ⋆ utilisation d'un mécanisme de réémission temporisée, *timer* ou compte à rebours :



  - ◇ si l'acquittement A_i ne parvient pas à S,
  - ◇ S considère au bout d'un certain temps que M_i est perdu et réémet M_i.

Dans la suite des transparents, le temps d'attente avant réémission automatique est appelé RTO, «Retransmission TimeOut», et il est calculé en fonction du RTT d'un segment (temps de transmission du segment+temps de réception de l'acquittement).

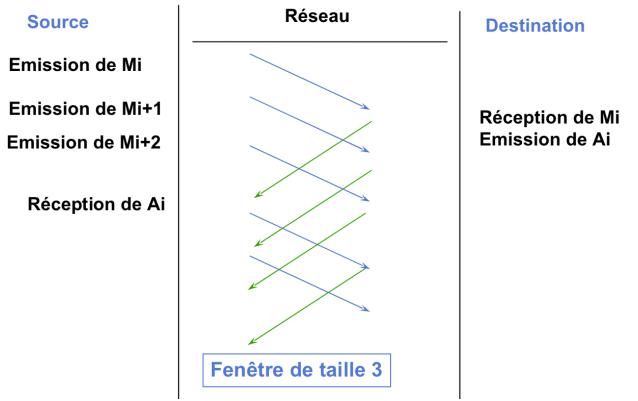
Contrôle de flux

TCP utilise un **mécanisme de fenêtre** pour faire du contrôle de flux :

Pourquoi ?

La technique basée sur l'acquittement simple **pénalise les performances** en faisant baisser le taux d'occupation du réseau :

- ◇ choisir la taille d'un segment adaptée à la MTU du réseau ;
- ◇ attendre la réception d'un acquittement avant d'émettre un nouveau segment ;
- ◇ *c-à-d l'envoi d'un **seul segment** par RTT !*



Comment ?

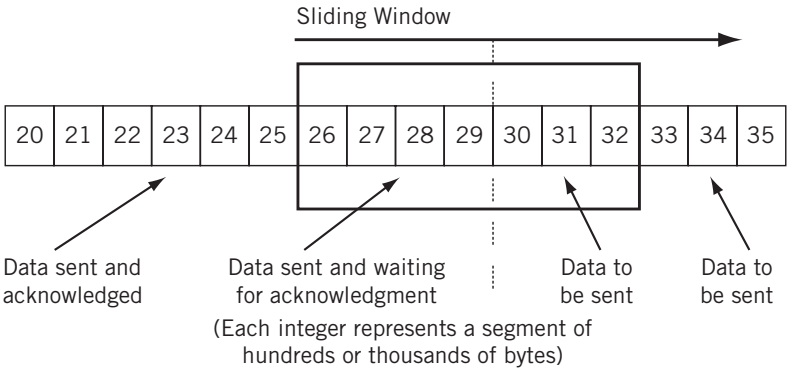
La taille T de la fenêtre :

- ⋆ correspond à une **taille en octets** et non à une taille en nombre de segments ;
- ⋆ est **variable** au cours de la communication ;
- ⋆ correspond à la capacité de la **mémoire tampon de réception** ;
- ⋆ est défini pour **chaque interlocuteur**.

Une fenêtre de taille T autorise l'émission d'au plus n segments (dont la somme des tailles $< T$), sans attendre l'acquittement de chaque segment envoyé, avant de ne plus pouvoir émettre.

Amélioration supplémentaire : acquitter **plusieurs messages** reçus en **un seul acquittement**.
On parle d'acquittement cumulatif ou d'acquittement retardés, «delayed ACKs».

Fenêtre glissante

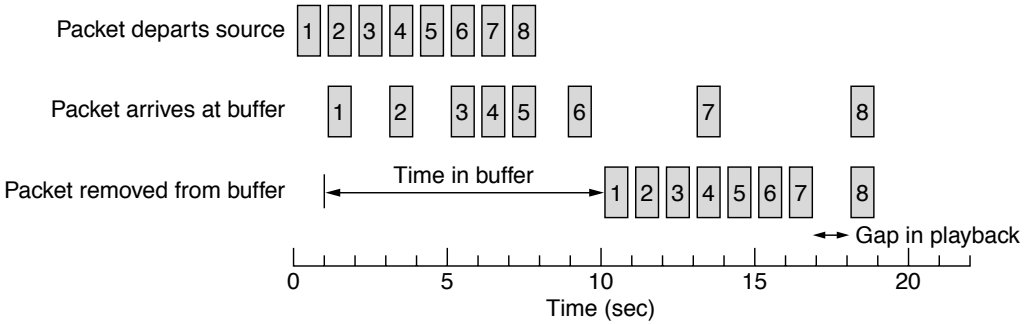


Le **récepteur** contrôle :

- ▷ la **taille de la fenêtre** (> à un segment) : plus petite ou plus grande suivant sa capacité de traitement ;
- ▷ **l'acquittement** des données reçues contigües : il fait *glisser* la fenêtre.

L'émetteur est **autorisé** à envoyer de nouveaux segments lorsque la fenêtre **glisse**.

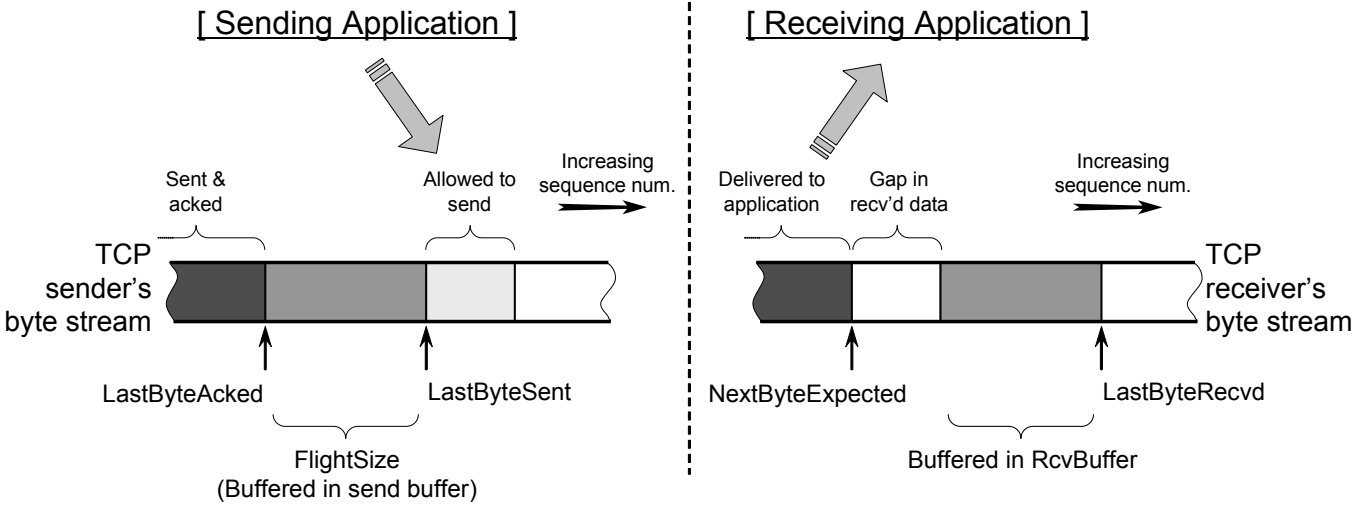
Bufférisation



Elle permet de :

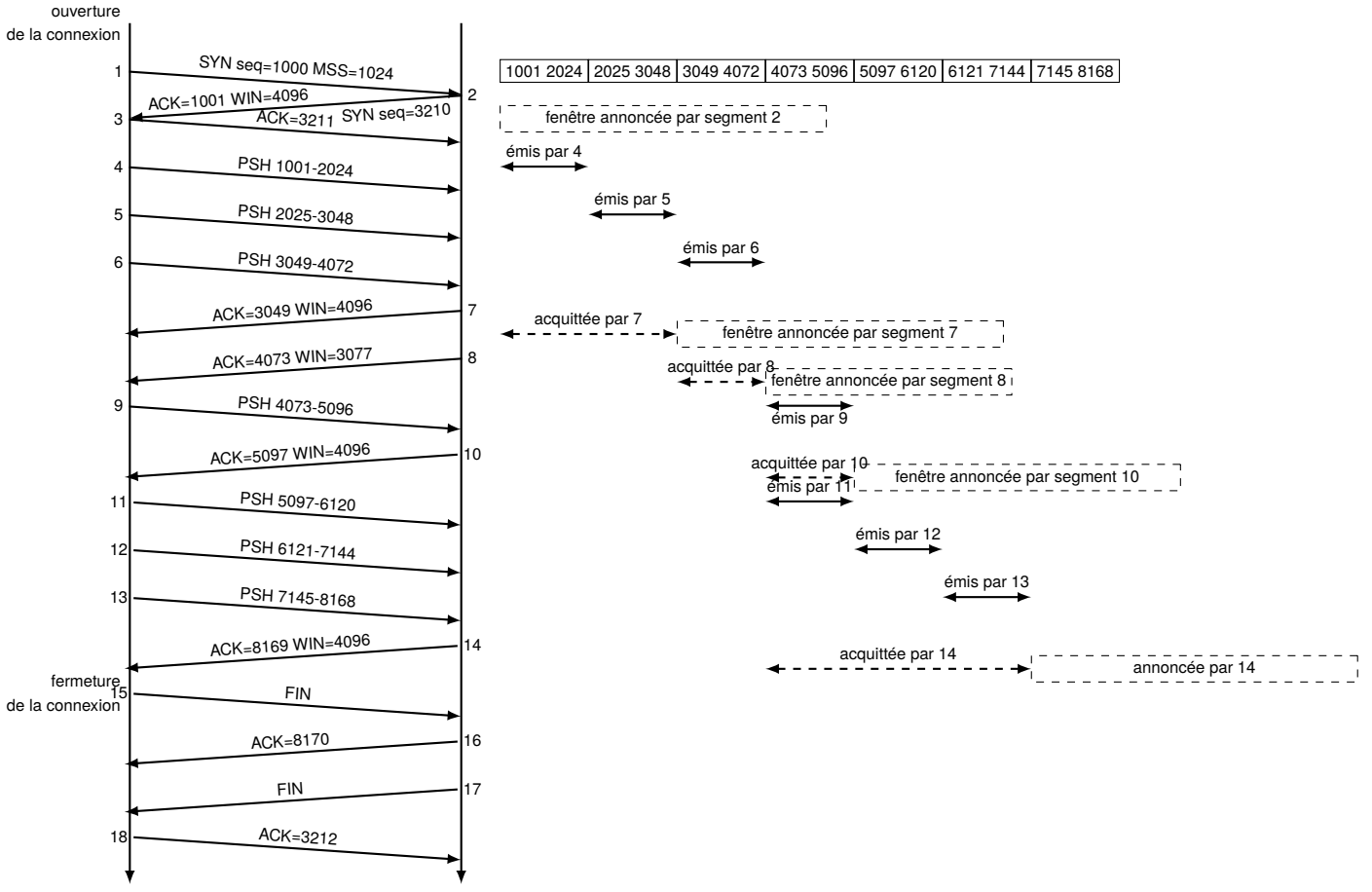
- ◇ «lisser» le débit en sortie ;
- ◇ gérer les trous, *gaps*.

Bufférisation *vs* Application



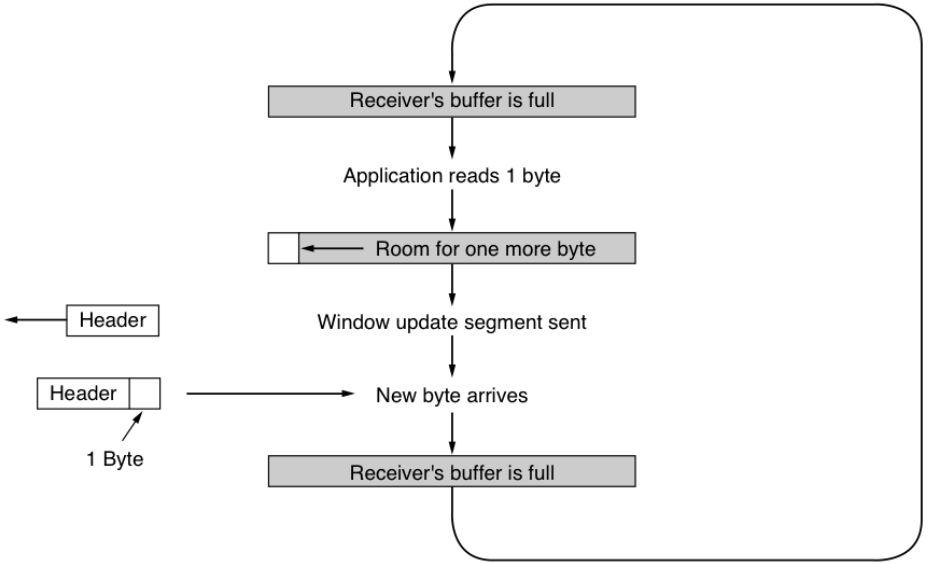
- L'émetteur dispose d'un **buffer d'envoi** dans lequel il stocke les données envoyées par l'application (si le buffer est plein, l'application est bloquée dans son ordre d'envoi) ;
- L'émetteur envoi des données entre le dernier octet acquitté, «LastByteAcked» et un octet, le «LastByteSent» (la taille de l'envoi doit être inférieure à la taille de la fenêtre d'autorisation, indiquée par «Allowed to send») ;
- *On appelle les données transmises des «données en vol», «data in flight», c-à-d qu'elles sont en train de circuler dans le réseau.*
- Le récepteur dispose d'un **buffer de réception** dans lequel il range les données reçues depuis le réseau ;
- Le récepteur range les données qu'il reçoit dans le buffer aux emplacements indiqués par l'«offset» du segment reçu : il peut y avoir des trous, «gap in received data».
- Le récepteur ne transmet à l'application que les données sans trou, indiquées par le «NextByte Expected». S'il n'y a pas suffisamment de données, l'application est bloquée dans son ordre de réception.

TCP : fenêtre glissante



TCP : problème de la fenêtre *stupide*

Silly Window Syndrome



Observation Il faut des algorithmes **empiriques** pour gérer les mécanismes de contrôle de TCP.

Solution de Clarke attendre que l'espace libre sur le récepteur soit suffisamment grand avant de prévenir l 'émetteur...

Transmission sans attente ou non bufférisées

- l'application peut demander que ses données soient **transmises immédiatement** même si le tampon d'envoi n'est pas plein (traversée du tampon d'envoi).

  Elles sont appelées données «poussées», *Push*, par exemple dans le cas des protocoles lignes par lignes (SMTP par exemple).
- l'arrivée de ces données est **notifiée sans attendre** à l'application réceptrice (traversée du tampon de réception).

Comment est-ce possible ?

Un bit «*PUSH*» est réservé dans le segment pour prévenir le récepteur.

Transmission de données urgentes et hors communication

Le protocole TCP permet l'envoi de **contrôle** et de **données** au travers de la même communication.

Ces informations de **contrôle** :

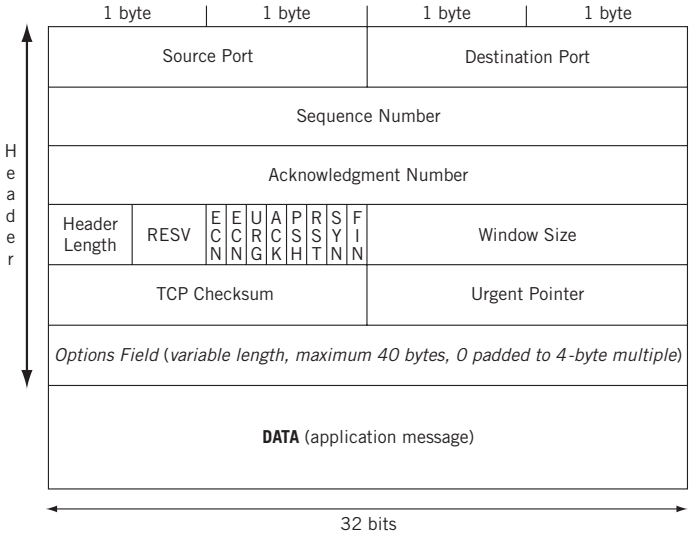
- sont à **prendre en compte immédiatement** par l'application réceptrice *Urgent* ;
- ne correspondent pas à des données «normales» de la communication (des données de contrôle, par exemple, comme celles utilisées pour avertir d'un problème sans nécessité d'établir une nouvelle connexion pour les transmettre) ;

  Elles sont appelées «*Out of Band*», elles génèrent un **signal Unix** sur l'application réceptrice.

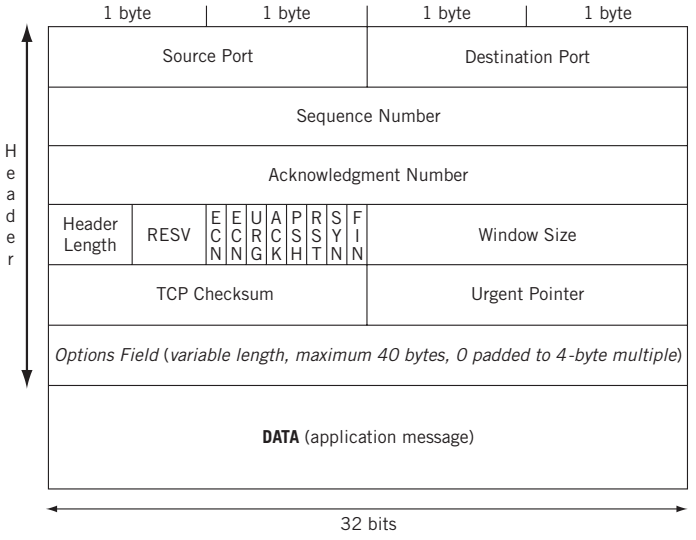
Comment est-ce possible ?

En utilisant le réseau datagramme sous-jacent : les datagrammes arrivent en même temps, qu'ils contiennent des données du flux ou bien des données «**out-of-band**» !

TCP : le format du segment



- **Port source et destination** :
  - *sur 16 bits*
  - identifie le processus associé et définie le TSAP.
- **Sequence number** :
  - *exprimé sur 32 bits*,
  - déplacement ou offset en octet à partir d'une valeur initiale, ISN, «*Initial Sequence Number*» ;
  - *positionne le segment dans le flux à recomposer* ;
- **Acknowledgement number** :
  - numéro de séquence du **prochain segment attendu** ;
  - acquitter les octets précédemment reçus (ceux de séquence $<$ à celle du prochain segment demandé) ;
    acquittement **cumulatif** *: il peut correspondre à plusieurs segments déjà reçus.*
- **Taille conseillée** du segment : 536 octets de données + 20 octets d'en-tête : 556 octets.
  $\Longrightarrow$ *Un lien réseau doit avoir une MTU suffisante pour pouvoir acheminer un segment TCP d'au moins 536 octets.*

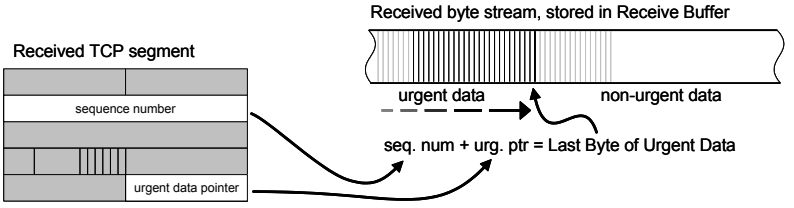


- **header length** ou «**Data Offset**» : *sur 4bits*, indique la taille en multiple de 4 octets.
- RESV : réservé pour un usage ultérieur sur 4bit ;
- ECN, *Explicit Congestion Notification* : *sur 2bits*, avertir le récepteur d'un état de début de congestion du réseau ;
- URG, ACK, PSH, RST, SYN, FIN : 6 bits
- **TCP checksum** : total de contrôle sur 16bits pour assurer la validité de l'en-tête et des données.

Le rôle des différents bits :

- **URG** indique des données urgentes indiquée par le «*Urgent Pointer*» ;
- **ACK** accusé de réception présent dans le «*Acknowledgement Number*» ;
- **PSH** ce segment doit être pris en compte sans bufférisation ;
- **RST** réinitialiser la connexion (problème de désynchronisation) ;
- **SYN** synchronisation des numéros de séquence pour initialiser une connexion ;
- **FIN** l'émetteur à fini d'émettre (permet de clore un sens de communication).

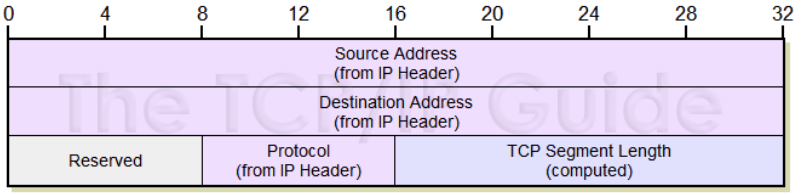
- **Window size** : *sur 16bits*,
  - utilisé pour le contrôle de flux par fenêtre glissante.
  - il indique le nombre d'octets que l'émetteur de ce segment est prêt à recevoir, ce qui permet à son interlocuteur d'augmenter ou de diminuer le nombre de segments qu'il peut transmettre (varie pendant la communication).
- **Urgent pointer** :
  - permet d'indiquer le rang du premier octet des données normales (les données urgentes sont en tête du segment) ;
  - ce rang est indiqué par rapport au segment courant (et pas du flux).
  - l'utilisation de ce champ est conjointe à celle du bit URG.



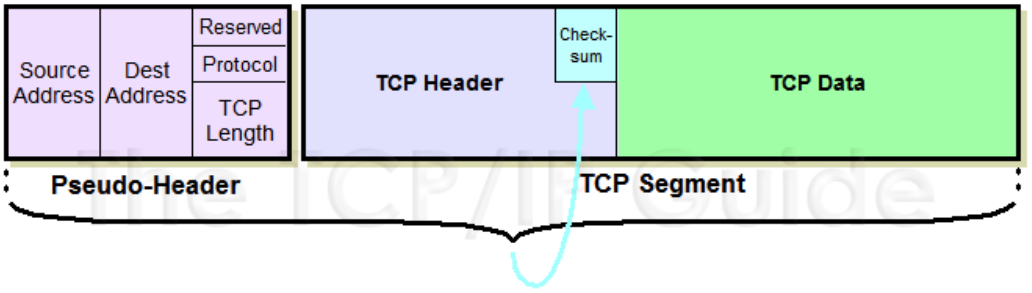
- **options** :
  - la MSS, *Maximum Segment Size* est d'indiquer lors de l'établissement de la connexion, la taille maximale du segment TCP que l'on peut envoyer.
    Une bonne taille est choisie en accord avec le MTU du réseau employé.
  - le WSF, *Window Scaling Factor*, sur 3 octets, définie un coefficient multiplicateur pour la donnée de la taille de fenêtre > 65535 ;
  - le TS, «*Time Stamp*», pour mesurer le RTT ;
  - …

Dans le segment TCP, une **somme de contrôle** est calculée sur :

- ▷ entête du segment TCP ;
- ▷ données du segment ;
- ▷ **pseudo** entête du datagramme IP :



Méthode



a. on additionne en complément à 1 tous les mots de 16 bits (complété par un octet nul dans le cas où le nombre d'octets total est impair)
b. on prend le complément à 1 de cette somme.

Lors de la réception le calcul est réalisé sur le segment entier (total de contrôle compris) et l'on doit obtenir zéro si le segment est correct.

Technique du "Piggybacking" ou mode en superposition

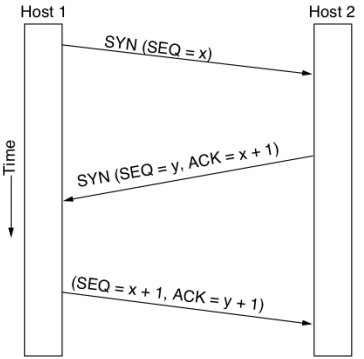
L'acquittement d'un paquet paquet de $A \rightarrow B$, peut être transmis dans un paquet allant de $B \rightarrow A$.

La communication pouvant être bidirectionnel simultanée, elle peut nécessiter la transmission d'information de $A \rightarrow B$ et de $B \rightarrow A$ simultanément.

L'établissement de la connexion, ou *handshake*

- ⋆ se fait en 3 étapes ;
- ⋆ correspond à une demande d'ouverture de la part de chaque interlocuteur ;
- ⋆ permet de synchroniser l'ISN, «Initial Sequence Number», la valeur initiale du numéro de séquence :
  - ◇ pour le `connect()` SYN=1 & ACK=0 ;
  - ◇ pour le `accept()` SYN=1 & ACK=1.

  Un SYN unique indique une demande de connexion.



Avec l'outil «tcpdump» :

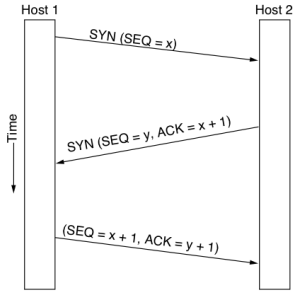
- ⋆ `[S]` : indique un SYN ;
- ⋆ `[S.]` : indique un SYN/ACK ;
- ⋆ `[.]` : indique un ACK.
- ⋆ connexion de :
  `192.168.42.83`
  port `56149`
  vers
  `164.81.1.45` port `25`.

xterm

```
pef@solaris:~$ sudo tcpdump -c 3 -n -S tcp and port 25
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:07:26.954500 IP 192.168.42.83.56149 > 164.81.1.45.25: Flags [S], seq
1249996154,
win 14600, options [mss 1460,sackOK,TS val 138100882 ecr 0], length 0

22:07:26.999157 IP 164.81.1.45.25 > 192.168.42.83.56149: Flags [S.], seq
4144328101, ack 1249996155,
win 5792, options [mss 1460,sackOK,TS val 1968359736 ecr 138100882],
length 0

22:07:26.999227 IP 192.168.42.83.56149 > 164.81.1.45.25: Flags [.], ack
4144328102,
win 14600, options [nop,nop,TS val 138100893 ecr 1968359736], length 0
3 packets captured
0 packets dropped by kernel
```

TCP : la connexion classique *vs* simultanée

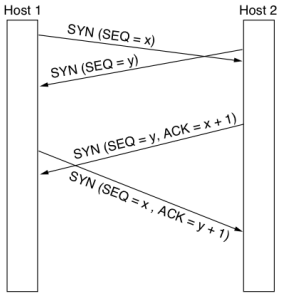
Connexion classique




```
N°   TCP A                                          TCP B
 1.  CLOSED                                        LISTEN
 2.  SYN-SENT    --> <SEQ=100><CTL=SYN>             --> SYN-RECEIVED
 3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>   <-- SYN-RECEIVED
 4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>     --> ESTABLISHED
 5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> -->   ESTABLISHED
```

CTL, «contrôle», indique les bits du segment TCP sélectionnées.

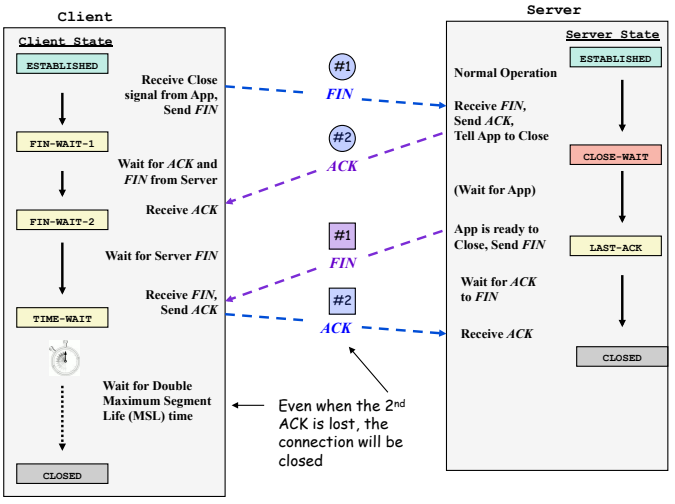
Connexion simultanée




```
     TCP A                                                TCP B
1.   CLOSED                                             CLOSED
2.   SYN-SENT    --> <SEQ=100><CTL=SYN>                ...
3.   SYN-RECEIVED  <-- <SEQ=300><CTL=SYN>                <--   SYN-SENT
4.                 ... <SEQ=100><CTL=SYN>                -->   SYN-RECEIVED
5.   SYN-RECEIVED  --> <SEQ=100><ACK=301><CTL=SYN,ACK> ...
6.   ESTABLISHED   <-- <SEQ=300><ACK=101><CTL=SYN,ACK> <--   SYN-RECEIVED
7.              ... <SEQ=101><ACK=301><CTL=ACK>       -->  ESTABLISHED
```

La connexion simultanée est difficile à mettre en œuvre, et elle est rarement implémentée dans les piles TCP/IP.

Déconnexion




```
   TCP A                                                TCP B
1. ESTABLISHED                                          ESTABLISHED
2. (Close)
   FIN-WAIT-1 --> <SEQ=100><ACK=300><CTL=FIN,ACK>    -->   CLOSE-WAIT
3. FIN-WAIT-2  <-- <SEQ=300><ACK=101><CTL=ACK>       <--   CLOSE-WAIT
4.                                                      (Close)
   TIME-WAIT   <-- <SEQ=300><ACK=101><CTL=FIN,ACK>   <--   LAST-ACK
5. TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>       -->   CLOSED
6. (2 Maximum Segment Lifetime)
 CLOSED
```

Le MSL, «Maximum Segment Lifetime» :

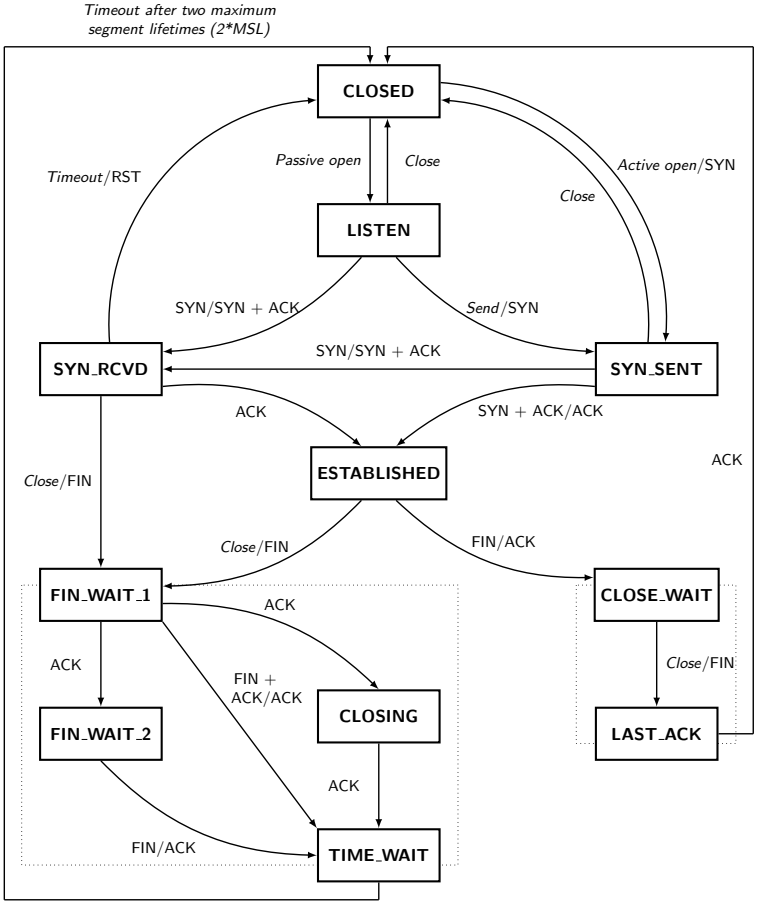
- ▷ *est choisie en fonction de l'expérimentation,*
- ▷ *permet d'éviter de prendre en compte des segments d'une ancienne connexion dans une nouvelle connexion (segment retardés dans le réseau).*
- ▷ *en général choisie à* $30s$.

Déconnexion rapide

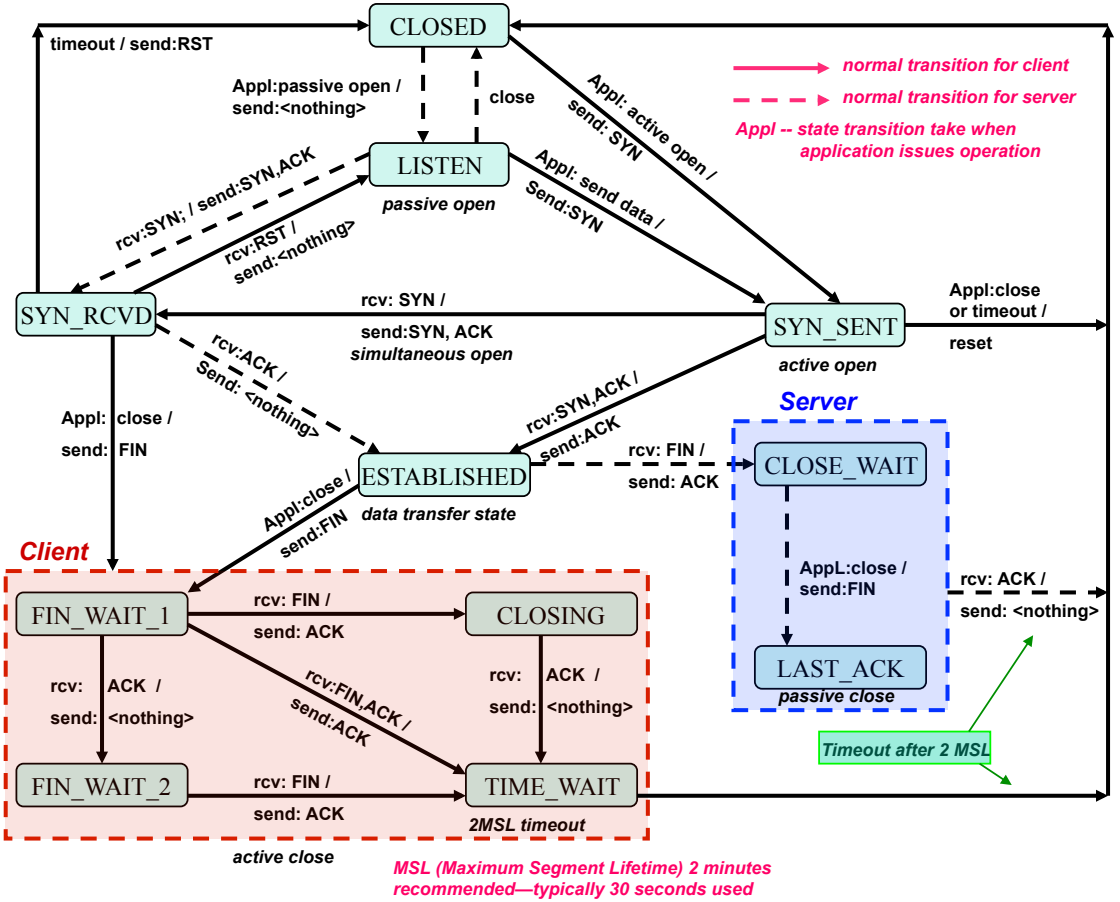
On utilise une déconnexion «abrupte» avec le `RST` :

- ◇ le serveur envoie un `FIN` au client ;
- ◇ le client envoie un `ACK`, puis un `RST` : on évite le temps d'attente de fin de connexion.

TCP : l'automate de fonctionnement



TCP : l'automate de communication expliqué



Gestion de la réception d'un ancien datagramme avec SYN

```
      TCP A                                                  TCP B
1.  CLOSED                                                 LISTEN
2.  SYN-SENT    --> <SEQ=100><CTL=SYN>              ...    pas encore reçu
3.  (duplication)      ... <SEQ=90><CTL=SYN>          -->    SYN-RECEIVED
4.  SYN-SENT    <-- <SEQ=300><ACK=91><CTL=SYN,ACK>  <--    SYN-RECEIVED
5.  SYN-SENT    --> <SEQ=91><CTL=RST>               -->    LISTEN
6.                          ... <SEQ=100><CTL=SYN>    -->    SYN-RECEIVED
7.  SYN-SENT    <-- <SEQ=400><ACK=101><CTL=SYN,ACK> <--    SYN-RECEIVED
8.  ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>     -->    ESTABLISHED
```

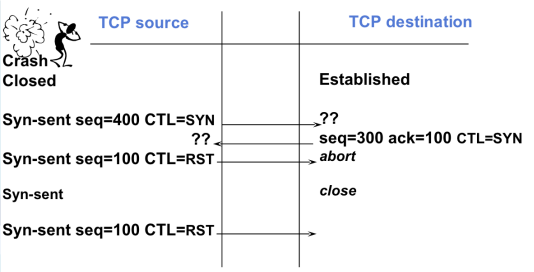
Gestion du RST

Le bit RST demande la **réinitialisation** de la communication.

Il est émis en rapport à une connexion :

- **non existante** : un RST est envoyé lors de la **réception de tout segment** relative à cette connexion (sauf si ce segment est déjà un RST).
  Les paquets SYN envoyés à une connexion inexistante sont rejetés par ce moyen.
  Cela permet de gérer le cas ou la machine a rebooté alors que le correspondant continue a communiquer.

- **existante** : lors de la réception d'un **segment qui ne correspond pas** à celui attendu par la connexion : segment entrant qui n'est pas synchronisé et/ou acquitte quelque chose qui n'a pas été envoyé (ACK mauvais).



- ⋆ **MSS**, Maximum Segment Size : la taille maximale des données contenues dans un segment (sans tenir compte de l'entête du segment).
  Cette option est transmise lors de l'établissement de la connexion Il peut y avoir une différence entre la valeur utilisée pour les deux sens de communication.
- ⋆ **Window Scale Factor**, qui permet de définir un coefficient multiplicateur pour la fenêtre glissante.
  Cette option permet de définir des tailles de fenêtre supérieure à 16bits : on décale à gauche du coef. indiqué (on multiplie par 2^{coef}) :
  Cette option est très utile pour les connexions très rapide (Gbits).
  Pour la désactiver sous GNU/Linux :

```
xterm
$ sudo sysctl -w net.ipv4.tcp_window_scaling=0
```

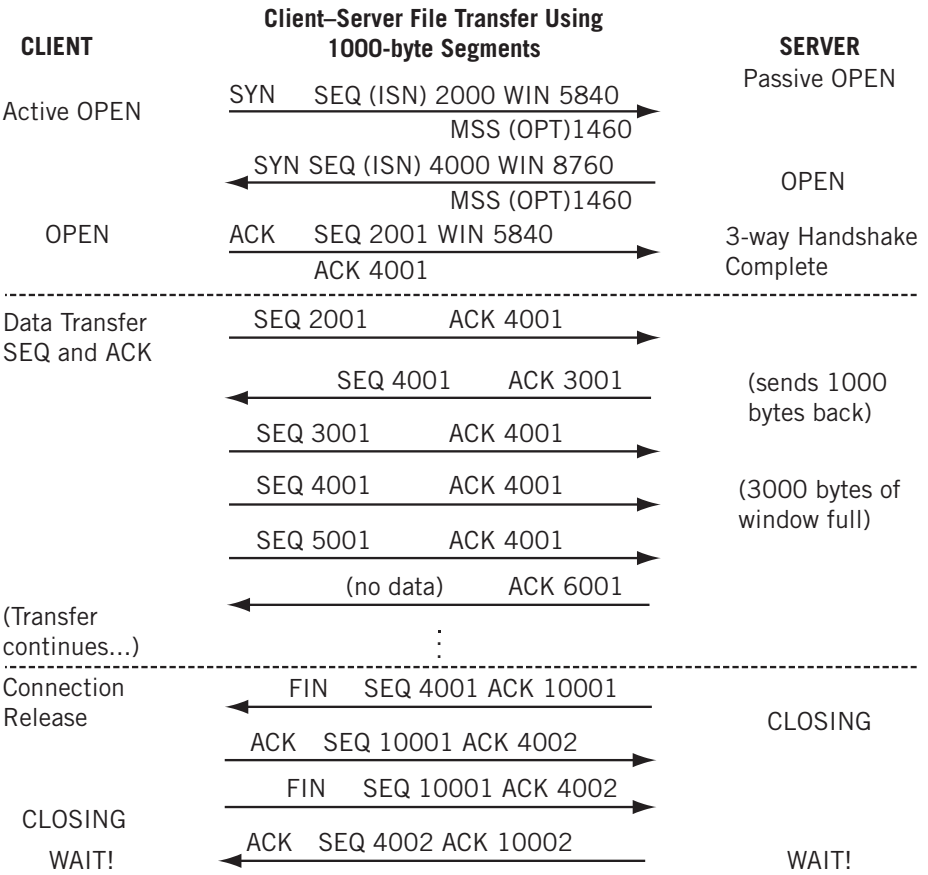
- ⋆ **Selective Acknowledgment Permitted**, ou SACK, qui permet d'acquitter des segments non contigües (RFC 2018) *Cette option est négociée à l'établissement de la connexion Dans le champs options peut être indiqué une liste de (adresse début, adresse fin) de données reçues mais non encore acquittées*
- ⋆ **Timestamp** : permet de calculer le RTT d'un segment (RFC1323) :
  - ◇ il existe deux champs contenant un timestamp : «Echo», «Echo reply» :
  - ◇ l'émetteur envoie un segment avec un timestamp par rapport à sa propre horloge ;
  - ◇ le récepteur lui renvoie ce timestamp dans le segment réponse : l'émetteur connait le RTT ;
  - ◇ le fonctionnement est symétrique : une seule option permet au deux interlocuteurs de faire leur mesure respective.

Autres améliorations

Keep-Alive, correspond à l'envoi de plusieurs segments de taille nulle ou déjà acquitté.
Cela permet pour les connexions inactives où aucun échange n'a lieu.

- ◇ «garder en vie» la connexion en évitant qu'un routeur réalisant du filtrage, «*firewall*», supprime la connexion du fait de son inactivité ;
- ◇ de tester la présence de l'interlocuteur : sans réponse, on mettra fin à la connexion.

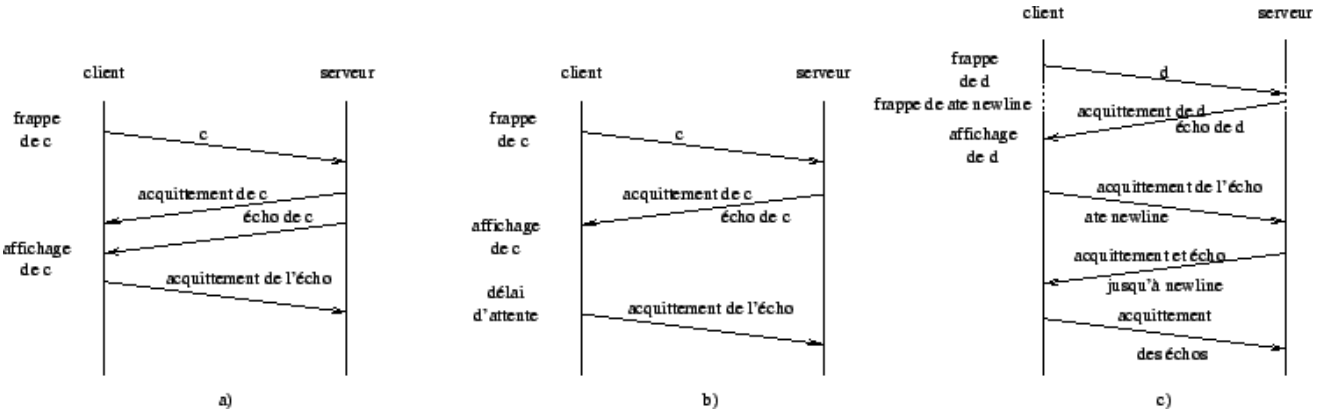
TCP : une communication complète



Le transfert de données de TCP est de deux types :

- **interactif** dans lequel chaque segment transporte très peu d'octets, voire un seul ;
- **en masse** où chaque segment transporte un maximum d'octets.

Exemple du protocole telnet, remplacé par le protocole SSH



L'utilisateur dans une session telnet envoie la commande `date` :

a. chaque caractère est transmis puis acquitté :c'est son acquittement qui est affiché à l'utilisateur, «echo»

b. en réalité TCP transmet l'écho en même temps que l'acquittement.

c. on utilise **l'algorithme de Nagle** (RFC 896) utilisant une bufférisation légère : si l'utilisateur tape vite on mémorise avant d'envoyer jusqu'à un retour à la ligne `\n` (utile pour ne pas saturer le réseau avec des segments ne contenant qu'un seul octet).

Il est :

- **différent de TCP** : pas de connexion, pas d'accusé de réception, pas de tri des datagrammes à la réception, pas de contrôle de flux, pas de garantie contre les pertes ;
- **identifié** par le numéro de protocole 17 (11 en hexa) dans le datagramme IP ;
- associé au **bit DF**, «*Don't Fragment*», du datagramme IP (en général) ;
- «***stateless***», contrairement à TCP : il n'y a pas d'**état conservé** par les hôtes l'utilisant pour communiquer (pas de TCB, «*transmission control block*» contenant les TSAPs, les numéros de séquence&acquittement, les buffers de réception/envoi, les fenêtres d'envoi/congestion *etc.*) ;
- capable de faire du «*multicast*» (envoyer simultanément à plusieurs destinataires) ;
- capable d'être **multiplexé** grâce au numéro de port :
  - *Well-kown* ports : de 0 à 1023, sont associés à des protocoles «système» ;
  - *registered* ports : de 1024 à 49151 peuvent être enregistrés auprès de l'ICANN ;
  - *dynamic* : 49152 à 65535 ;
  - *ephemeral* vs *persistent* : celui d'un client par rapport à celui d'un serveur ;

  La liste des associations ports/services est indiquée dans le fichier `/etc/services`*, sous GNU/Linux et dans* `C:\%SystemRoot%\system32\ drivers\etc\SERVICES` *sous Windows.*

Ce protocole est utilisé pour les applications :

- de type «*question/réponse*», où l'établissement d'une connexion serait trop coûteuse : la requête et la réponse peuvent être contenues dans un seul datagramme.
- «*temps réels*» : celles dans lesquelles le plus important est d'avoir les données **à temps**, comme la transmission du son et de l'image.

Pour obtenir la liste des ports en attente de connexion pour TCP et de datagramme pour UDP, sous GNU/Linux, à l'aide de la commande `netstat` :

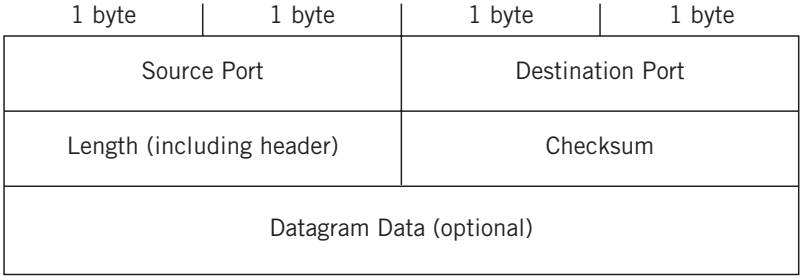
xterm

```
$ sudo netstat -lp --inet
Connexions Internet actives (seulement serveurs)
Proto Recv-Q Send-Q Adresse locale     Adresse distante   Etat     PID/Program name
tcp        0      0 *:https            *:*                LISTEN    2403/apache2
tcp        0      0 *:17500            *:*                LISTEN    1979/dropbox
tcp        0      0 localhost:mysql    *:*                LISTEN    3535/mysqld
tcp        0      0 *:www              *:*                LISTEN    2403/apache2
tcp        0      0 *:domain           *:*                LISTEN    1677/dnsmasq
tcp        0      0 *:ssh              *:*                LISTEN    1706/sshd
tcp        0      0 localhost:ipp      *:*                LISTEN    1561/cupsd
udp        0      0 *:52906            *:*                          470/avahi-daemon
udp        0      0 *:domain           *:*                          1677/dnsmasq
udp        0      0 *:bootpc           *:*                          393/dhclient
udp        0      0 *:bootpc           *:*                          21932/dhclient
udp        0      0 *:17500            *:*                          1979/dropbox
udp        0      0 *:mdns             *:*                          470/avahi-daemon
udp        0      0 *:radius           *:*                          2328/freeradius
```

Des protocoles utilisant UDP :

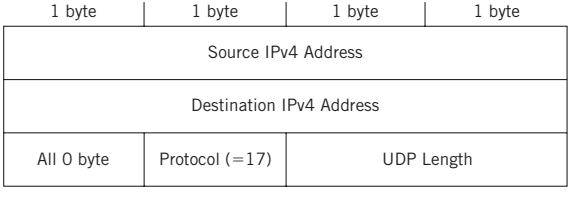
- NTP permet de mettre les ordinateurs à l'heure par internet à 500 millisecondes près ;
- DNS permet de retrouver une adresse IP en fonction d'un nom symbolique (comme un annuaire) ;
- VoIP permet de communiquer vocalement par Internet ;
- IPTv pour la télé sur Internet (quelle idée !) ;
- …

Le format du datagramme UDP



- la **taille du datagramme UDP** inclue celle de l'entête : elle varie de 8, juste l'entête, à 65 353 (il dépend de la taille du buffer d'envoi et de réception souvent limité à 8000 octets) ;
- il est **encapsulé dans un datagramme IP** de taille minimale à supporter par les routeurs de 576 octets, soit 556 octets sans l'entête IP (certains protocoles le limitait à 512 octets comme le protocole DNS, *Domain Name Server*, dans la RFC1035, mais cette limitation a été levée dans la RFC2671) ;

Calcul du Checksum



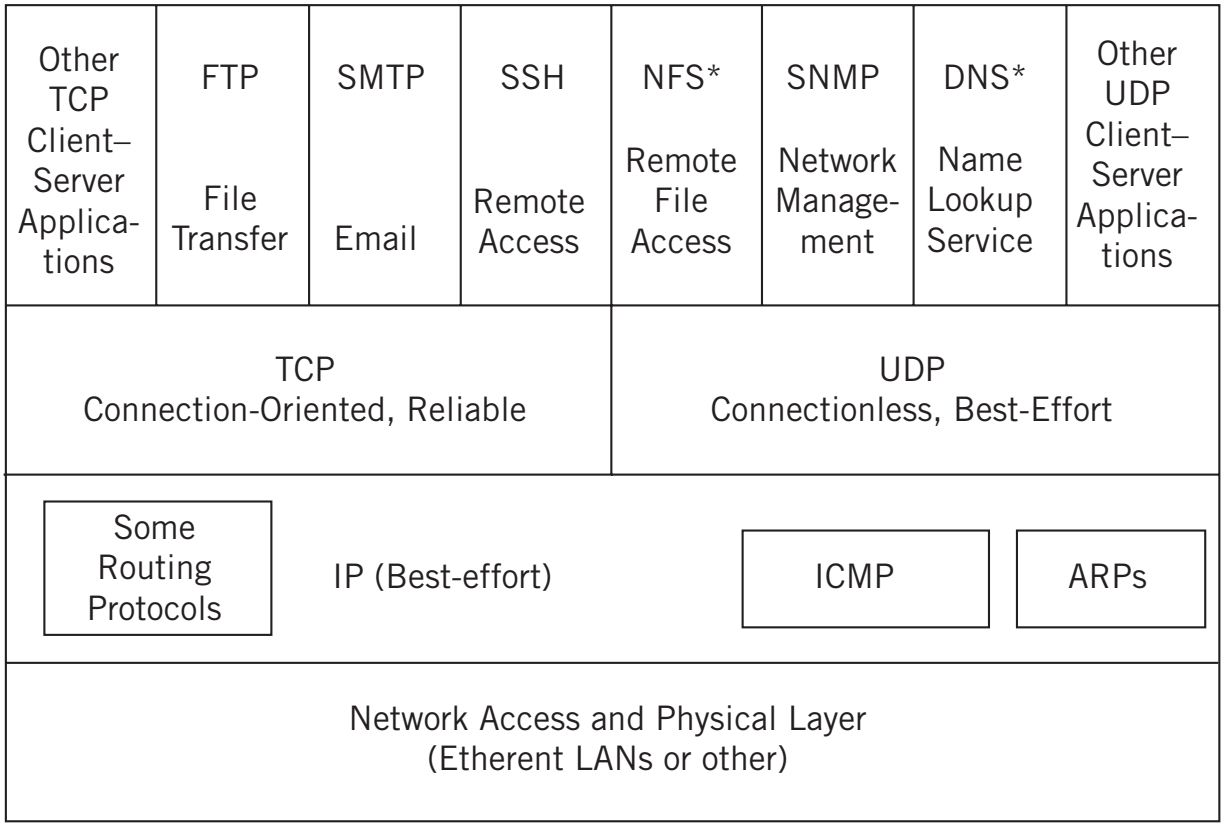
- il est calculé avec une **pseudo-entête** :
- il n'est pas obligatoire **mais conseillé** (obligatoire dans IPv6) :
  - si le client envoie avec un checksum, le serveur répond avec un checksum ;
  - s'il est mauvais le datagramme est éliminé *silently*$\Longrightarrow$ **pas de message d'erreur**.

UDP : les différents cas de réponses

Certains envois de datagramme UDP peuvent générer une réponse d'erreur de type ICMP :

Action	Condition	Outcome
UDP request sent to server	Server available	Sender gets UDP reply from server
UDP request sent to server	Port is closed on server	Sender gets ICMP "Port unreachable" message
UDP request sent to server	Server host does not exist	Sender gets ICMP "Host unreachable" message
UDP request sent to server	Port is blocked by firewall/router	Sender gets ICMP "Port unreachable — Administrative prohibited"message
UDP request sent to server	Port is blocked by silent firewall/router	(timeout)
UDP request sent to server	Reply is lost on way back	(timeout)

UDP vs TCP



*In some instances, NFS and DNS use TCP.

Le protocole HTTP

La naissance de la *toile* ou du «*Web*»

C'est le «*phénomène*» qui a fait explosé l'utilisation d'internet.

Il date de 1989, inventé par Tim Berners-Lee, physicien au CERN.

But
- ▷ permettre un **accés facile** à des documents de différentes natures
  - ◇ utiliser le concept **d'URL**
  - ◇ utiliser le format de descritption de contenu **MIME**
- ▷ permettre la création de **document composite** (texte+image+son)
  - ◇ définir un format de document mélant ces éléments de nature différente
- ▷ permettre d'établir des **interdépendances** entre des documents :
  - ◇ relier des documents entre eux par un lien définit sur le contenu
- ▷ faciliter la **localisation** des documents
  - ◇ utilisation de document de « départ » **connu** pour l'accés à d'autres documents **moins connus**

Moyen
- ▷ définition d'un **format de document structuré** (HTML "Hyper Text Markup Language")
- ▷ définition d'un **protocole simple** pour l'échange de données basés sur le modéle client/serveur (envoi de requête, réception d'un document) (HTTP "Hyper Text Transfer Protocole")
- ▷ utilisation du format universelle pour la **désignation** d'un document (URL "Uniform Resource Locator")
- ▷ réutilisation du format MIME pour la **description** des documents échangés

Outils
- ▷ le **serveur Web** permettant la mise à disposition de documents (en général au format HTML)
- ▷ le **navigateur Web**, ou "browser", permettant la récupération et la consultation de ces documents

Serveurs spécialisés le **moteur de recherche** qui parcours, indexe les documents accessibles par le Web et permet de rechercher ces documents en fonction de leur contenu (pages HTML), où de leur nom.

Localisation et accés à l'information (RFC 738 et 808)

Le problème de l'accés aux données est un **double problème**, il faut indiquer :

- ▷ l'endroit où se trouve la resource ;
- ▷ le moyen pour la récupérer avec éventuellement des **autorisations d'accès**.

Format universel

```
service :// adresse_machine [:n° port] / chemin_accés
```

En général le nom du service correspond à celui du protocole :
Exemple : `http://www.sciences.unilim.fr` `ftp://ftp.unilim.fr`, `news://news.unilim.fr`

Peuvent être ajouté :

- ☐ une identité : `ftp://toto@alphainfo.unilim.fr` ;
- ☐ une identité et un mot de passe : `ftp://toto:top_secret@ftp.unilim.fr`
- ☐ un chemin d'accés à un répertoire ou à un fichier ou à un groupe :
  `ftp://ftp.unilim.fr/pub/mac`
  `http://www.sciences.unilim.fr/index.html`
  `news://news.unilim.fr/fr.rec.*`
- ☐ un **numéro de port** de connexion pour utiliser un numéro de port différent de celui par défaut du service (serveur utilisateur ne pouvant utilisé un port réservé par l'administrateur, serveur supplémentaire, port non filtré par un firewall...)

Dans le cas d'une localisation avec un chemin d'accés vers un répertoire ou un fichier, il est nécessaire de tenir compte des **droits d'accés** à ces resources.

L'accés à un fichier peut être **bloqué**, ou le contenu d'un répertoire **interdit en lecture**.

Mais il peut être utile de modifier l'URL au niveau du chemin d'accés pour pouvoir accéder à une resource qui aurait été déplacée (changement de répertoire ou de nom...).

Un système de requêtes/réponses

La base de la consultation de documents sur le Web est le mécanisme de requête/réponse :

Pour le client web :

- ▷ il reçoit une URL,
- ▷ il analyse l'URL pour en déduire l'adresse de la machine désignée par l'URL et le nom du document à demander
- ▷ il se connecte sur la machine hébergeant la resource de la manière indiquée dans l'URL (ftp, http...) – après une connexion réussie, il réclame le document
- ▷ en général, la connexion est interrompue.

Chrome, Opera, Brave... sont des navigateurs web.

Pour le serveur Web :

- ▷ il traite différentes natures de document, mais uniquement pour un accés suivant le protocole HTTP ;
- ▷ il attend la connexion d'un client, en général, sur les port 80 et 443 pour la version sécurisée par TLS ;
- ▷ il reçoit une requête ;
- ▷ il analyse la requête afin de déterminer le document demandé ;
- ▷ il vérifie les autorisations d'accés pour le document (sous Unix, le serveur Web se comporte comme unutilisateur ayant des droits d'accés restreint, en particulier sur les autres fichiers que ceux qu'il met en consultation) ;
- ▷ il retourne le document demandé au client ;
- ▷ en général, il termine sa connection avec le client ;

Apache est le serveur web le plus employé et le plus sûr.

Caractéristiques

Protocole texte
- ☐ échange de lignes de commandes au format ASCII 7bits
- ☐ transmission entre le client et le serveur basé TCP.
- ☐ **très simple**, ce qui explique sa popularité et sa facilité de mise en œuvre.

Versions

HTTP/0.9 version de base avec requête/réponse le document est renvoyé directement ;

HTTP/1.0 version normalisée (RFC 945) avec comme amélioration, l'ajout :
- ▷ d'en-tête pour la description des resources échangées (utilisation du format MIME RFC822),
- ▷ des informations supplémentaires envoyées par le client (format de données supporté ou désiré, description de la version et de la marque du navigateur...)

HTTP/1.1 ajout de **connexions persistantes** entre le client et le serveur en vue de l'échange de plusieurs resources par l'intermédiaire de la même connexion (transfert des différents éléments d'un même document composite).

HTTP/2.0 version transitoire avec transfert dans **une même connexion TCP** de différentes ressources **fractionnées** ;

HTTP/3.0 protocole Quic **basé sur UDP** mélangeant négociation sécurité (authentification/chiffrement) visant à diminuer le nombre d'échanges nécessaires (rtt, «*round trip time*») et échange de données **frationnées**.

Dialogue client/serveur

Le protocole HTTP se caractérise par l'échange d'une requête et d'une réponse.

Ces **requêtes** et **réponses** ont un schéma similaire :
- ▷ une ligne initiale : ligne de commande pour la requête ou d'état pour la réponse ;
- ▷ aucune ou plusieurs lignes d'en-tête : `entête: valeur`, `entête: valeur` ;
- ▷ une ligne « vide »
- ▷ un message optionnel (un fichier, des données pour la requête ou des résultats de la requête)

Le protocole HTTP : illustration

Commande «`socat`» pour établir une connexion TCP et on envoie les lignes suivantes :

xterm

```
pef@darkstar:~$ socat stdio tcp:p-fb.net:80
GET /toto HTTP/1.0
Host: p-fb.net

```

ligne vide pour indiquer la fin de la requête

On reçoit de la part du serveur les lignes suivantes :

xterm

```
HTTP/1.1 404 Not Found
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Access-Control-Allow-Origin: *
ETag: "6178833c-634"
x-proxy-cache: MISS
X-GitHub-Request-Id: A898:8123:D1862A:D8A34D:617BCE57
Content-Length: 1588
Accept-Ranges: bytes
Date: Fri, 29 Oct 2021 10:35:03 GMT
Via: 1.1 varnish
Age: 0
Connection: close
X-Served-By: cache-cdg20758-CDG
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1635503703.088888,VS0,VE98
Vary: Accept-Encoding
X-Fastly-Request-ID: 51070abe1910e8323daccf62b4550f1cc43c3e89

<!DOCTYPE html>
<html lang="fr" class="js csstransforms3d">

<head>
...
```

ligne d'état

début de la ressource : ici, une page HTML

La ligne de commande dans le cas de la requête

Ces commandes sont divisés en **trois parties** séparées par des espaces :

- une méthode :

GET	Récupération d'un document
HEAD	Consultation de renseignement sur un document
PUT	Rangement d'un document sur le serveur
POST	« Ajout » à une resource des informations données
DELETE	Suppression d'un document
LINK	Définition d'une connexion entre deux documents
UNLINK	Suppression d'une connexion entre deux documents

- le chemin d'accès de la resource ou requête URI «*Uniform Resource Identifier*» ;
- la version du protocole HTTP utilisé indiquée sous la forme « HTTP/x.x » en majuscule ;

Exemple : `GET /chemin/d/acces/au/fichier/index.html HTTP/1.0`

La ligne d'état dans le cas de la réponse

Elle est décomposée en trois parties séparées par des espaces :

- la version du protocole HTTP
- un code d'état pour donner le résultat de la requête suivi d'une phrase expliquant la raison du code d'état ;

Le code d'état est un entier sur trois chiffres, dont le premier chiffre donne la catégorie générale :

1xx	message d'information seulement
2xx	indication du succés
3xx	redirection du client vers une autre URL
4xx	indication d'une erreur au niveau du client
5xx	indication d'une erreur au niveau du serveur

Les lignes d'en-tête

Ces lignes donnent des informations à propos de la **requête** ou de la **réponse**.

Elles sont au format général introduit dans la RFC 822, c-à-d « Nom-En-tête: valeur » :

- les noms d'en-têtes peuvent être minuscules ou majuscules ;
- il peut y avoir des espaces et des tabulations entre le « : » et la valeur ;
- une ligne d'en-tête commençant par un espace ou une tabulation est associé à la ligne précédente :

```
EN_TÊTE: un_nom_tres_long, un_autre_nom_tres_long
```

est équivalent à :

```
EN_TÊTE:  un_nom_tres_long,
          un_autre_nom_tres_long
```

Il existe de nombreuses en-têtes (HTTP/1.0 : 6 en-têtes, HTTP/1.1 : 46 en-têtes).

Au niveau d'une requête :

- `Host` **:** qui est obligatoire pour indiquer le nom symbolique du serveur que l'on veut utiliser ;
  Dans le cas de l'hébergement de plusieurs sites Web sur une même adresse IP, il permet de distinguer le site auquel on veut avoir accès en fonction de l'alias de la machine que l'on a utilisé : `193.50.85.1` *correspond à* `www.un_site.un_domaine.fr` *et* `www.un_autre_site.un_domaine.fr`
- `User-Agent` **:** permet d'identifier le logiciel qui fait la requête ;
  Exemple : `User-agent: Mozilla/0.9`

Au niveau d'une réponse :

- `Server` **:** permet d'identifier le logiciel serveur
- `Last-modified` **:** permet de donner la date d'actualisation de la resource (intéressant pour un cache...)

Le message optionnel transmis après les lignes d'en-tête

En général, il correspond à la **resource** elle-même dans le cas d'une **réponse**.
Dans le cas d'une «requête», ce message peut contenir une resource à **mettre à jour** (commande PUT par exemple) ou bien des informations **saisies** par l'utilisateur (champs de formulaire).
Ce message dispose de sa propre en-tête, exprimée sous forme de lignes d'en-tête :

- `Content-Type` pour donner la nature de cet élément au format MIME ;
- `Content-Transfer-Encoding` pour indiquer quel format est utilisé pour le transférer ;
- `Content-Length` pour donner la taille de l'élément.
  Dans le cas de paramètres joints à une requête, c'est un champ obligatoire !
  Il permet au serveur de lire **exactement** le nombre d'octets concernant l'élément, avant de passer à l'élaboration de la réponse.

Cas particuliers de certaines commandes ou méthodes

- méthode `HEAD` : seule l'en-tête de la réponse est envoyée (juste la ligne d'état et les en-têtes correspondant au serveur mais pas le message optionnel).
  Cette méthode permet de savoir si une resource a été modifiée et doit être de nouveau récupérée.

- méthode `POST` : elle est employée pour envoyer des données de l'utilisateur.
  Elle utilise pour décrire le message joint à la requête, les en-têtes `Content-Length` et `Content-Type`.
  Ces informations sont fournies **en entrée** d'un programme et la réponse du serveur correspond à la **sortie** de ce programme :
  - l'URI employée est souvent le nom du programme à déclencher ;
  - la réponse est souvent construite de manière dynamique par l'exécution de ce programme.

On parle de CGI, «Common Gateway Interface».

Mode spécial de transfert du message optionnel

Le mode « en morceaux », `Content-Transfer-Encoding: chunked`, permet de transmettre un message optionnel crée par morceaux et dont on ne connait pas la taille totale a priori.
(on donne à chaque fois en hexadécimal sur deux chiffres suivi d'un point-virgule la taille du morceau suivi du morceau).

Les connections persistantes

Dans le cas de la récupération d'un document **composite**, il est courant de devoir récupérer **plusieurs ressources** sur un **même serveur** et il est alors intéressant de pouvoir utiliser la **même connexion** pour échanger toutes ces resources :

- ▷ cette modification a été introduite dans la version 1.1 du protocole HTTP, pour lequel c'est le comportement par défaut.
- ▷ le client effectue **plusieurs requêtes** à la suite, et lit les réponses dans le même ordre avec lequel il a envoyé ses requêtes.

Si un client ne veut pas utiliser ce comportement, il doit ajouter la ligne d'en-tête `Connection: close` dans sa requête, auquel cas le serveur ne lira qu'une requête et fermera sa connection après avoir donné sa réponse.

Les connections lentes

Un serveur peut rendre une réponse **temporaire**, indiquée par la ligne d'état `100 Continue` avant de transmettre une réponse complète. Cela permet de **retarder** la transaction entre le serveur et le client.

Les connections avec cache

Le serveur doit pouvoir gérer les lignes d'en-têtes suivantes :

- ▷ `Date` pour permettre d'estampiller les données transmises ;
- ▷ `If-modified-since` pour ne donner la resource que si elle a été modifiée.
  Dans le cas où elle n'a pas été modifiée le serveur répond par `304 Not Modified`.

MIME, «*Multipurpose Internet Mail Extensions*» RFC 1521

But :

- redéfinition du format des messages tout en restant compatible avec le format définie dans la RFC 822.
- envoi de texte accentués (français, allemand, tchèque...)
- envoi de texte dans des alphabets non latins (hébreu, cyrillique, grec...)
- envoi de texte dans des langues sans alphabet (chinois, japonais, coréen...)
- envoi de données non textuelles (audio, vidéo...)

Attention, la capacité lors de la réception des messages à traiter le contenu dépend entièrement de la configuration de la machine du destinataire !

Moyens :

- ajout de lignes pour contrôler le traitement du message à l'arrivée ;
- découpage du message en plusieures parties avec des marqueurs de délimitation ;
- champs supplémentaires et obligatoires au niveau de l'en-tête :
  - MIME-Version
  - Content-type
  - Content-transfer-encoding
  - Content-Id (optionnel)

Le champ `content-type` est divisé en deux champs :

- type du contenu : text, image, video, application, multipart
- format du contenu : text/plain, text/html, image/gif, image/jpeg, video/mpeg, multipart/mixed, application/octet-stream

La commande «curl» va récupérer les données au format JSON et ces données vont être formatées par le module «json.tool» :

xterm

```
pef@darkstar:/Users/pef $ curl -sH 'Accept: application/json' http://api.icndb.com/jokes/random | python3 -m
json.tool
{   "type": "success",
    "value": {
        "categories": [
            "nerdy"
        ],
        "id": 543,
        "joke": "Chuck Norris's programs can pass the Turing Test by staring at the interrogator."
    }
}
        }
    ],
}
```

xterm

```
pef@darkstar-8:/Users/pef $ curl -vI http://www.unilim.fr/
*   Trying 164.81.1.97...
* TCP_NODELAY set
* Connected to www.unilim.fr (164.81.1.97) port 80 (#0)
> HEAD / HTTP/1.1
> Host: www.unilim.fr
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
HTTP/1.1 301 Moved Permanently
< Server: nginx
Server: nginx
< Date: Mon, 11 Sep 2017 10:56:46 GMT
Date: Mon, 11 Sep 2017 10:56:46 GMT
< Content-Type: text/html
Content-Type: text/html
< Connection: keep-alive
Connection: keep-alive
< Location: https://www.unilim.fr/
Location: https://www.unilim.fr/

<
* Connection #0 to host www.unilim.fr left intact
```


Connaissance du client par le serveur HTTP

- Le protocole HTTP permet **d'accéder** aux ressources.
- La gestion des liens est assuré au niveau du format **HTML** (hypertexte).
- Il n'existe **pas de liens** entre les ressources d'un même document composite vis-à-vis du protocole HTTP (au plus il y a une partie d'URL commune).

Un serveur n'a pas connaissance (ou très réduite) des utilisateurs récupérant les ressources qu'il met à disposition.

Dans le cas de la **récupération successive**, par le client, de plusieurs documents au format HTML, le serveur ne garde pas trace de cette succession et surtout ne l'associe pas à la navigation d'un **même client**.

Dans le cas de l'utilisation d'un **proxy** (un serveur intermédiaire servant à centraliser toutes les requêtes des différents utilisateurs), tous les utilisateurs de ce proxy sont ignorés. Le serveur n'a conscience que du proxy.

On parle de protocole «*stateless*» (sans état).

Besoins d'un dialogue client/serveur personnalisé

- obtention d'une ressource suivant les désirs de l'utilisateur (choisir la langue du document HTML, le nombre de liens par page dans le cas d'une recherche sur un moteur de recherche...)
- envoi d'information de la part de l'utilisateur (passer une commande, s'inscrire pour recevoir de la publicité...)
- simulation d'un dialogue entretenu durant une navigation (session de connexion sur un serveur...)

Mécanisme de personnalisation

Il existe **différents mécanismes** permettant de **personnaliser** une transaction entre un client et un serveur, mais toutes ces méthodes consiste à transmettre depuis le serveur, une information au client.

Cette information sera ensuite **retransmise** au serveur lors du chargement d'un nouveau document et permettra d'identifier de **manière unique** le client vis-à-vis du serveur :

– utilisation de « formulaire » HTML, avec éventuellement des champs cachés ;
– utilisation de «*cookies*» (Attention : un cookie ne doit être envoyé qu'au serveur qui l'a émis)

Un cookie est un ensemble de données confié à un tiers, dont l'interprétation lui est cachée. – construction d'URL contenant un identifiant.

Durée de la personnalisation

Elle dépend du mécanisme employé :

- dans le cas d'un **cookie**, ce cookie peut être mémorisé au sein du navigateur Web de manière **courte** (dès que le navigateur quitte ou que le serveur en demande la destruction) ou **longue** (indépendante dela connexion au site et jusqu'à expiration du cookie) ;
- dans le cas d'une **URL**, l'URL peut être **mémorisée** au sein du navigateur (sous forme de signet) et peut éventuellement permettre de reprendre une navigation personnalisée ;
- dans le cas de **formulaire**, les valeurs des champs de formulaires ne sont pas mémorisés. Il existe néanmoins des mécanismes permettant de les sauvegarder en tant que cookie.

Notion de session

L'identification du client auprès du serveur va permettre la définition d'une session :

– **ouverture** de la session :
  - identification par nom et mot de passe dans le cas d'un site sécurisé
  - requête de la page d'accueil du site
– **fermeture** de la session :
  - déconnexion souhaitée par l'utilisateur par défaut,
  - fermeture du navigateur

Extension du protocole HTTP (RFC 209)

La gestion d'un cookie se fait par ajout de lignes d'en-têtes au niveau du protocole HTTP.

Dans la réponse envoyée depuis un serveur :

```
Content-type: text/html
Set-Cookie: gateau=genoise; path=/; expires Mon, 11-Dec-2021 13:46:00 GMT
```

Ces lignes demandent au navigateur le stockage d'un cookie dont :

- ▷ le nom est « gateau » associé à la valeur « genoise » ;
- ▷ la date d'expiration est le 11 décembre 2021 à 13h46 heure de Greenwich.
- ▷ le chemin d'accés est « / » soit le site dans sa globalité.

Lors d'une nouvelle requête, le navigateur web transmet les lignes d'en-têtes suivantes :

```
Content-type: text/html
Cookie: gateau=genoise
```

Définition du cookie

- ☐ le **nom** du cookie si la valeur associée au cookie est vide, le cookie est détruit ;
- ☐ la **valeur** du cookie, exemple : nom=valeur ;
- ☐ la **date d'expiration** du cookie, optionnel. Si elle n'est pas spécifié le cookie expire à la fermeture de la session (fermeture du navigateur ou fermeture explicite de la session) ;
- ☐ le **chemin** pour lequel le cookie est **valide**, restreint l'accés au cookie par rapport au chemin d'accés de la page ;
- ☐ le **domaine** pour lequel le cookie est **valide**, permet de définir les domaines pouvant avoir accés au cookie. Il peut y en avoir plusieurs dans le cas d'un site réparti sur plusieurs machines ;
- ☐ le besoin de disposer d'une **connexion sécurisée** pour échanger le cookie.

Méthode «*POST*» : les champs/valeurs sont transmis après la requête

xterm

```
$ curl -v -X POST -d 'user=toto' -d 'passwd=supersecret' http://www.unilim.fr
*   Trying 164.81.1.97:80...
* TCP_NODELAY set
* Connected to www.unilim.fr (164.81.1.97) port 80 (#0)
> POST / HTTP/1.1
> Host: www.unilim.fr
> User-Agent: curl/7.68.0
> Accept: */*
> Content-Length: 21
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 21 out of 21 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 301 Moved Permanently
< Server: nginx
< Date: Mon, 22 Nov 2021 12:38:35 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< Location: https://www.unilim.fr/
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
<
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host www.unilim.fr left intact
```

ici, sont transmis les paramètres `user` *et* `passwd`

Les données transmises par l'utilisateur sont «cachées» *dans la transaction HTTP.*
Elles sont encodées pour éviter les caractères spéciaux.

Méthode «*POST*» : les champs/valeurs sont transmis après l'URL

xterm

```
$ curl -v 'http://www.unilim.fr?user=toto&passwd=supersecret'
*   Trying 164.81.1.97...
* TCP_NODELAY set
* Connected to www.unilim.fr (164.81.1.97) port 80 (#0)
> GET /?user=toto&passwd=supersecret HTTP/1.1
> Host: www.unilim.fr
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Server: nginx
< Date: Mon, 22 Nov 2021 21:40:03 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< Location: https://www.unilim.fr/?user=toto&passwd=supersecret
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
<
<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host www.unilim.fr left intact
* Closing connection 0
```

Les données transmises par l'utilisateur sont ajoutées dans l'URL ; elles sont visibles dans la barre d'adresse du navigateur et leur encodage n'est pas garanti.