

IP et ARP



ARP

✦ Rôle essentiel d'ARP en réseau

✦ On a une adresse IP

- - > soit directement (ping)
- - > soit via table de routage (message qui transite)

✦ Nécessité d'avoir l'@MAC pour créer la trame

Utilisation **ARP**

correspondance @ IP - > @MAC dans table MAC

!! ARP prend du temps -> pb avec ICMP



Le WEB

— ■ — ■ — ■ — ■ — ■ — ■ — ■ — ■ — ■ — ■ —

HTTP

HTTPS

— ■ — ■ — ■ — ■ — ■ — ■ —

Protocole HTTP

✦ HTTP : HyperText Transfert Protocol (RFC 1945 et 2616)

- ✦ Protocole de rapatriement des documents
- ✦ Protocole de soumission de formulaires
- ✦ 2 versions un peu différentes : HTTP 1.0 et 1.1
- ✦ **Fonctionnement au-dessus d'une connexion en TCP**
 - Utilisation d'une URI (Uniform Resource Identifier)
 - `http://<nom_machine>:<port>/<path>?<query>`
- ✦ Actuellement version : HTTP 2.0 -> RFC 7540
 - HTTP/3.0 en cours de déploiement RFC 9114

Méthode HTTP (1)

✦ Quelques méthodes

- http 1.0 {
- ✦ **méthode GET**
 - récupération d'un document
 - ✦ **méthode POST**
 - envoi de données au programme situé à l'URL spécifié
 - ✦ **méthode HEAD**
 - récupération des informations sur le document (lignes d'en tête)
 - ✦ PUT (document à enregistrer)
 - ✦ DELETE (permet d'effacer un fichier)
 - ✦ OPTIONS (différentes options utilisables par le serveur)
 - ✦ CONNECT (connexion à un proxy)
 - ✦ TRACE (récupérer le message reçu par le serveur)

Requête HTTP

✧ Une requête http comprend:

- ✧ une ligne de requête contenant

- la méthode ou commande
- l'URL
- la version du protocole utilisé (http/1.1)

- ✧ les champs d'en-tête de la requête

- des informations supplémentaires
info : valeur

- ✧ Une ligne blanche

- ✧ le corps de la requête (si nécessaire)

- ✧ Exemple :

- GET /index.html HTTP/1.1\r\nHost: www.isima.fr\r\nAccept : text/html, application/xhtml+xml\r\nAccept-Encoding : gzip, deflate\r\nConnection : keep-alive\r\n\r\n

Réponse HTTP

✦ Une réponse http comprend:

- ✦ une ligne de statut contenant
 - la version du protocole utilisé
 - le code de statut
 - la signification du code
- ✦ les champs d'en-tête de la réponse
 - des informations supplémentaires
info : valeur
- ✦ Une ligne blanche
- ✦ le corps de la réponse

HTTP/1.1 200 OK\r\n

Date: Tue, 22 Apr 2014 09:48:55 GMT\r\n

Server: Apache/2.4.4 (Win32) PHP/5.4.14\r\n

Last-Modified: Wed, Feb 2014 15:57:14 \r\n

Content-Length: 125\r\n

Keep-Alive: timeout=5, max=99\r\n

Connection: Keep-Alive\r\n

Content-Type: text/html\r\n

\r\n

<html>\r\n

<head><title>page essai</title>\r\n

</head>\r\n

<body>\r\n

<H3>Vive le reseau</H3>\r\n</html>

Quelques en-têtes (1)

✧ Générique :

- ✧ Content-length : longueur des datas
- ✧ Content-type : type MIME des datas
- ✧ Connection : keep-alive ou close (optimisation à partir de http 1.1)

✧ Requête

- ✧ **Host : hôte virtuel demandé (obligatoire à partir de http 1.1)**
- ✧ Accept : Type MIME autorisé
- ✧ Accept-encoding : Méthode de compression autorisée
- ✧ Cookie : cookie mémorisé par le client
- ✧ User-agent : nom et version du logiciel client
- ✧ If-modified-since : renvoie le document seulement si il a été modifié
- ✧ etc

Content-length et content-type obligatoire avec requête POST

Quelques en-têtes (2)

✦ Réponse :

- ✦ Date : date de la réponse
- ✦ Server : nom et version du logiciel
- ✦ Last-modified : date dernière modification
- ✦ Content-language : langue utilisée pour le document
- ✦ Expires : date à laquelle le document expire
- ✦ Etag : numéro de version du document
- ✦ Set-cookie : pour mettre un cookie
- ✦ Etc...

Codes de réponse

✧ 100 à 199

- ✧ informationnel

✧ 200 à 299

- ✧ Succès de la requête
 - 200 : OK
 - 201 : Created
 - 202 : Accepted ...

✧ 300 à 399

- ✧ Redirection
 - 301 : moved permanently
 - 302 : found ...

✧ 400 à 499

- ✧ Erreur due au client

- 400 : bad request
- 401 : unauthorized
- 402 : payment required
- 403 : forbidden
- 404 : not found

✧ 500 à 599

- ✧ Erreur due au serveur

- 500 : internal error
- 501 not implemented
- 502 : bad gateway ...

Commande HTTP (2)

✦ Pour récupérer un document

- ✦ méthode **GET** : les paramètres sont passés dans la ligne de l'URL

(le " " est remplacé par +, caractères spéciaux par %code ascii)

- *ex: GET /~toto/q1.php?name1=val1&name2=val2 HTTP/1.1*

Host : www.isima.fr

(<http://www.isima.fr/~toto/q1.php?name1=val1&name2=val2>)

- ✦ méthode **POST** permet de mettre les arguments dans le corps de la requête (→ invisible)

- *ex : POST /~toto/q2.php HTTP/1.1* *(<http://www.isima.fr/~toto/q2.php>)*

Host : www.isima.fr

Content-type : application/x-www-form-urlencoded

Content-length : xx

** une ligne blanche **

name1 = val1 & name2 = val2

Cache HTTP

✦ 2 caches possibles :

- ✦ sur le navigateur
- ✦ **sur les proxys** (équipement réseaux)

✦ En-têtes concernés

- ✦ Last-modified
- ✦ Expires
- ✦ Date
- ✦ If-modified-since
- ✦ Cache-control
- ✦ Etag avec if-Match ou If-None-Match

On ne renvoie pas la réponse → gain de données envoyées

Suivi de session (1)

✧ Motivations

- ✧ la notion de session est importante dans une application conversationnelle
- ✧ Hors, **HTTP est un protocole sans état**
 - une connexion à chaque demande ou presque
le serveur ne maintient pas d'informations liées aux requêtes d'un même client
- ✧ Comment implanter la notion de session sur plusieurs requêtes HTTP ?

Suivi de session (2)

-
- ✦ Le serveur génère un identificateur de session et associe un état à une session

le client renvoie l'identificateur de session à chaque requête

✦ Plusieurs solutions possibles:

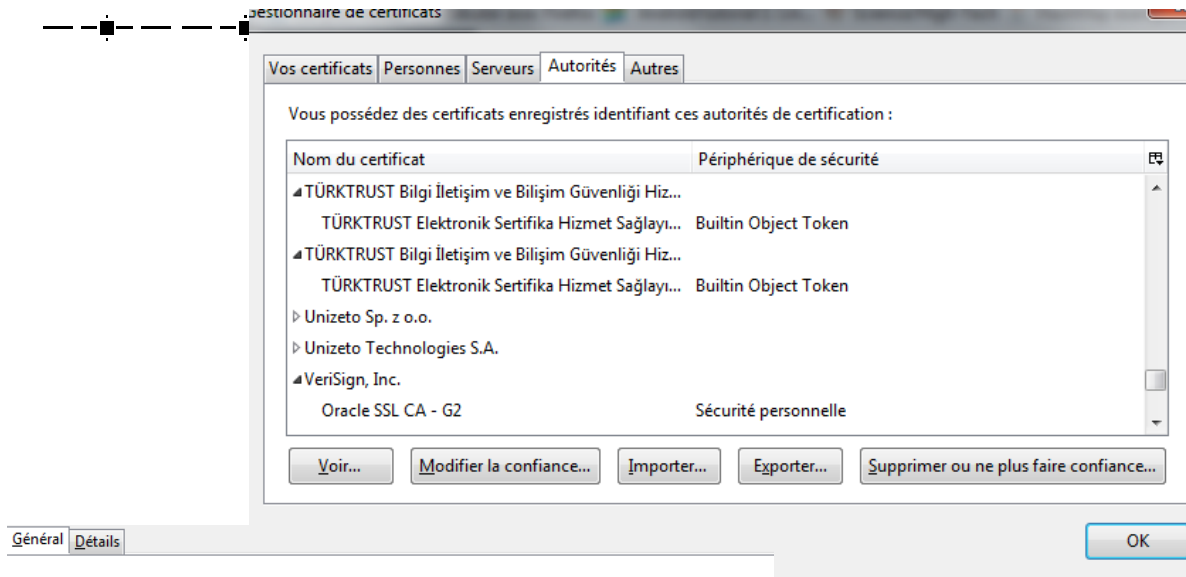
- ✦ Input HIDDEN dans les formulaires
 - on renvoie à chaque fois un identifiant
- ✦ la Ré-écriture dans chaque URL
 - on remet l'identifiant dans les réponses
- ✦ Utilisation d'un cookie
 - Set-cookie venant d'un serveur
 - Cookie à partir du client

Https (1)

✦ HTTPsecured : HTTP+SSL

- ✦ Vérification du site grâce à un certificat
- ✦ Utilisation d'un chiffrement asymétrique puis symétrique
- ✦ Certificat de type X.509
 - Signé par une autorité de certification
 - Auto-signé
- ✦ Confidentialité et intégrité des données
- ✦ **Utilisation du port 443**
- ✦ Utilisation pour toutes les transactions bancaires, financières, achats,...

Https (2)



Ce certificat a été vérifié pour les utilisations suivantes :

Autorité de certification SSL

Émis pour

Nom commun (CN)	Oracle SSL CA - G2
Organisation (O)	Oracle Corporation
Unité d'organisation (OU)	Symantec Trust Network
Numéro de série	31:35:BB:7E:6D:E2:66:3A:B0:E2:33:E6:B0:E1:A5:C

Émis par

Nom commun (CN)	VeriSign Class 3 Public Primary Certification Authority - G5
Organisation (O)	VeriSign, Inc.
Unité d'organisation (OU)	VeriSign Trust Network

Période de validité

Début le	06/01/2015
Expire le	06/01/2025

Empreintes numériques

Empreinte numérique SHA-256	E4:AF:2F:AE:41:18:7D:58:F2:09:B0:1B:1D:87:53:C2:DC:CB:3F:60:1C:E8:62:73:E3:7E:87:38:C2:A5:CC:B5
Empreinte numérique SHA1	03:86:D9:39:CF:7B:A7:88:55:27:34:18:5B:EA:D0:B1:3E:87:39:14



Cette connexion n'est pas certifiée

Vous avez demandé à Firefox de se connecter de manière sécurisée à **linuxfr.org**, mais nous ne pouvons pas confirmer que votre connexion est sécurisée.

Normalement, lorsque vous essayez de vous connecter de manière sécurisée, les sites présentent une identification certifiée pour prouver que vous vous trouvez à la bonne adresse. Cependant, l'identité de ce site ne peut pas être vérifiée.

Que dois-je faire ?

Si vous vous connectez habituellement à ce site sans problème, cette erreur peut signifier que quelqu'un essaie d'usurper l'identité de ce site et vous ne devriez pas continuer.

Sortir d'ici !

- ▶ Détails techniques
- ▶ Je comprends les risques

https (3)

✦ Failles :

◆ Man In the Middle

- Attaque presque invisible
- Passage de https en http

◆ Sécurité interne à l'entreprise

- On sécurise la communication
- On ne sécurise pas le stockage des données
- Est-ce que tous les certificats sont valables ?
- Pb sur la gestion de la validité...

Défaut de http

✧ Problèmes :

◆ Latence d'une page

- Latence du réseau , mais disparaît avec le haut-débit
- Augmentation de la rapidité en ne fermant pas la connexion (http 1.1)
- Utilisation de connexions parallèles pour faire plusieurs requêtes simultanées
 - ◆ Mais maximum 8 threads par serveur
- Utilisation du pipelining dans les requêtes

Amélioration de la latence, **Mise en place de HTTP /2**

- basé sur le protocole SPDY
 - Développé par google mais abandonné au profit de http/2

HTTP/2 (1)

✦ Compatible avec HTTP1.0 et 1.1

- ◆ Même méthode utilisée, même codes de réponse
- ◆ Même URI

✦ But : être transparent pour l'utilisateur et "plus rapide"

✦ Segmentation des données, et multiplexage de celles-ci

✦ Envoi des données en binaire (au lieu de ASCII)

✦ Tous les navigateurs et serveurs web peuvent utiliser HTTP2

HTTP/2 (2)

✦ Actuellement , passage de http1.1 en http/2

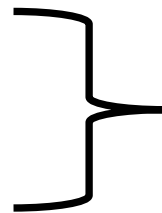
◆ Dans le cas non sécurisé

◆ Lors de la 1^{er} requête, rajout d'un champ Upgrade

- GET / HTTP/1.1 Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c (http2 clear)
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>

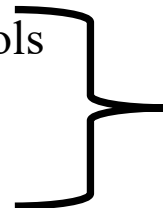
◆ 2 possibilités :

- HTTP/1.1 200 OK
Content-Length: 352
Content-Type: text/html



http2 non supporté

- HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c



http2 supporté, suite en http2

HTTP/2(3)

✧ En mode sécurisé

- ✧ **Interdiction** d'utiliser un **upgrade : h2** pour passer de http1.1 à http2s
- ✧ Utilisation d'un protocole au-dessus de TLS
 - Utilisation de l'upgrade h2
 - **ALPN : Application-Layer Protocole Negotiation**
 - ♦ Lors de l'échange TLS, prise en compte de l'extension ALPN
 - ♦ Inclut dans le java 9
 - ♦ Utilisable via openssl

Les navigateurs ont mis en place seulement HTTP/2 pour connexion sécurisée utilisant min TLS1.2 actuellement

HTTP/2 (4)

✦ Améliorations

◆ Compression en-tête

- Utilisation de la compression (protocole HPACK)
- Ne pas renvoyer ce qui a déjà été envoyé (cookie, ...)

◆ Utilisation de push préventif

- Envoie de données qui pourront servir plus tard

Entre 15 à 40 % de gain sur une communication !!!

HTTP/3.0 (1)

✧ Objectif

- ✧ Etre encore plus rapide, et passer outre les « défauts » de TCP
 - Connexion avec triple poignée de main
 - Utilisation seq /ack pour contrôler les flux
 - Problèmes dans le multiplexage avec ce contrôle de flux
 - Rajout de trames si connexions en mode sécurisé
- ✧ Accélérer les transmissions

✧ Utilisation de UDP en couche transport

- ✧ Evite les phases de connexion

✧ Mise en place du **Protocole QUIC** au-dessus UDP

- ✧ Gère le multisession
- ✧ Met en place des paramètres pour calculer la perte des données



Les Sockets

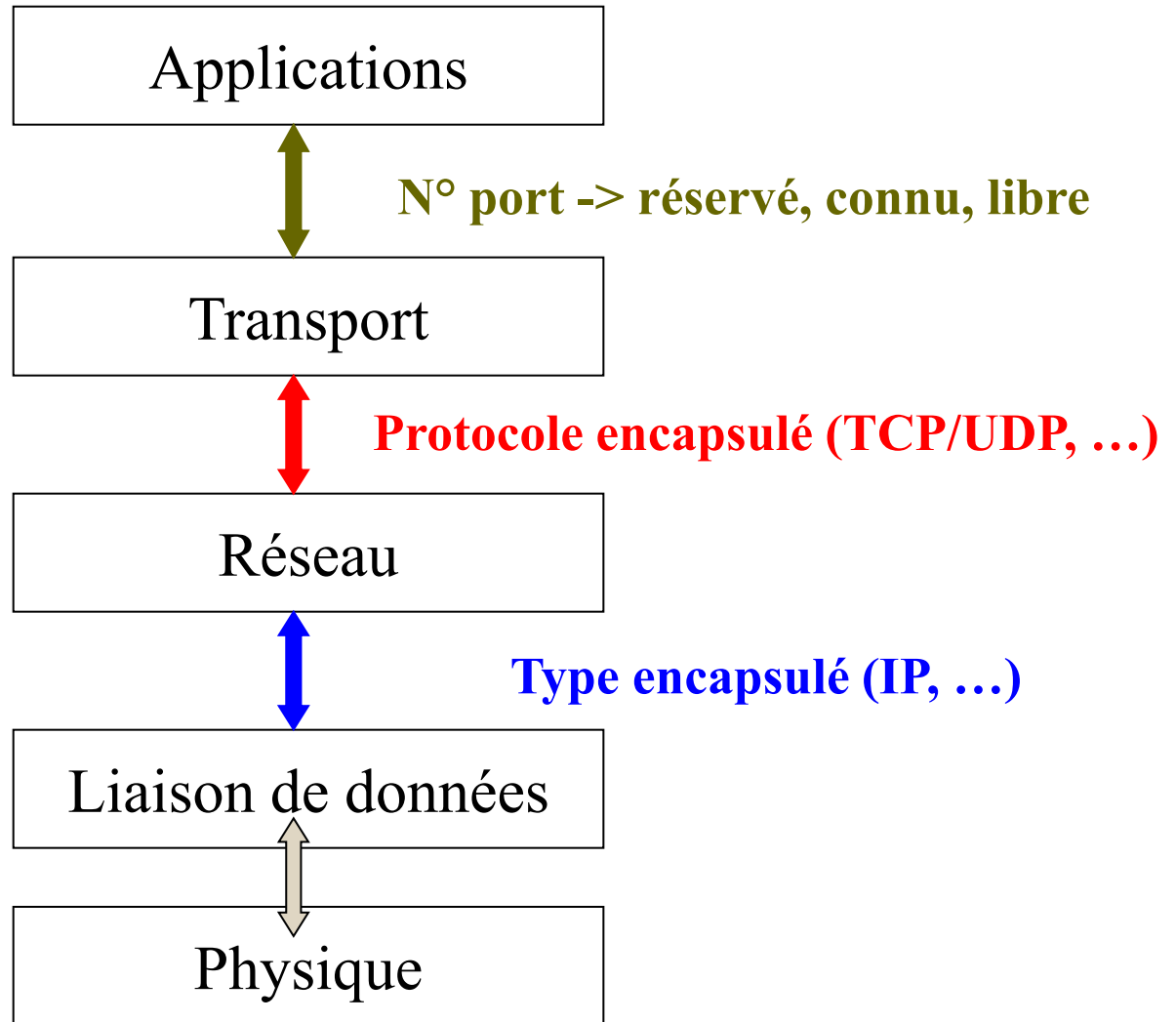
-
- Généralités
 - Programmation pour IPv4
 - En langage C
 - En JAVA
-

Généralités sur les sockets(1)

- ✦ Mécanisme d'interface de programmation
 - ◆ permet aux processus d'échanger des données
 - ◆ n'implique pas forcément une communication par le réseau (ex socket unix)

- ✦ Une connexion est entièrement définie sur chaque machine par :
 - ◆ le type de protocole (TCP, UDP,...)
 - ◆ l'adresse IP
 - ◆ **le numéro de port associé au processus**
 - (statiquement pour le serveur, dynamiquement pour le client)

Généralités (2)



Mode connecté/ non connecté

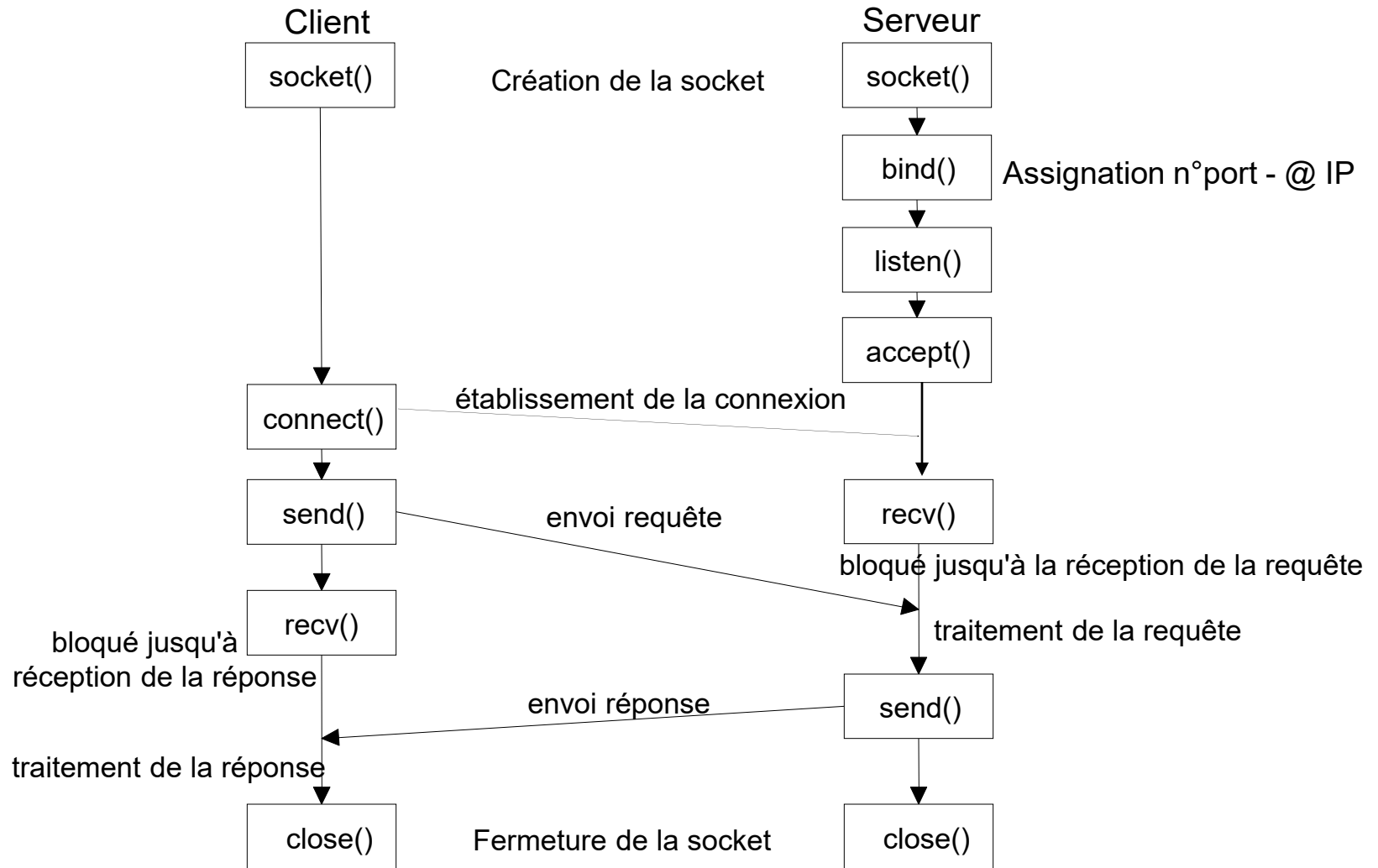
✦ Mode connecté (TCP)

- Problèmes de communication gérés automatiquement
- Gestion de la connexion coûteuse en message
- Primitives simples d'émission et de réception
- Pas de délimitation des messages dans le tampon

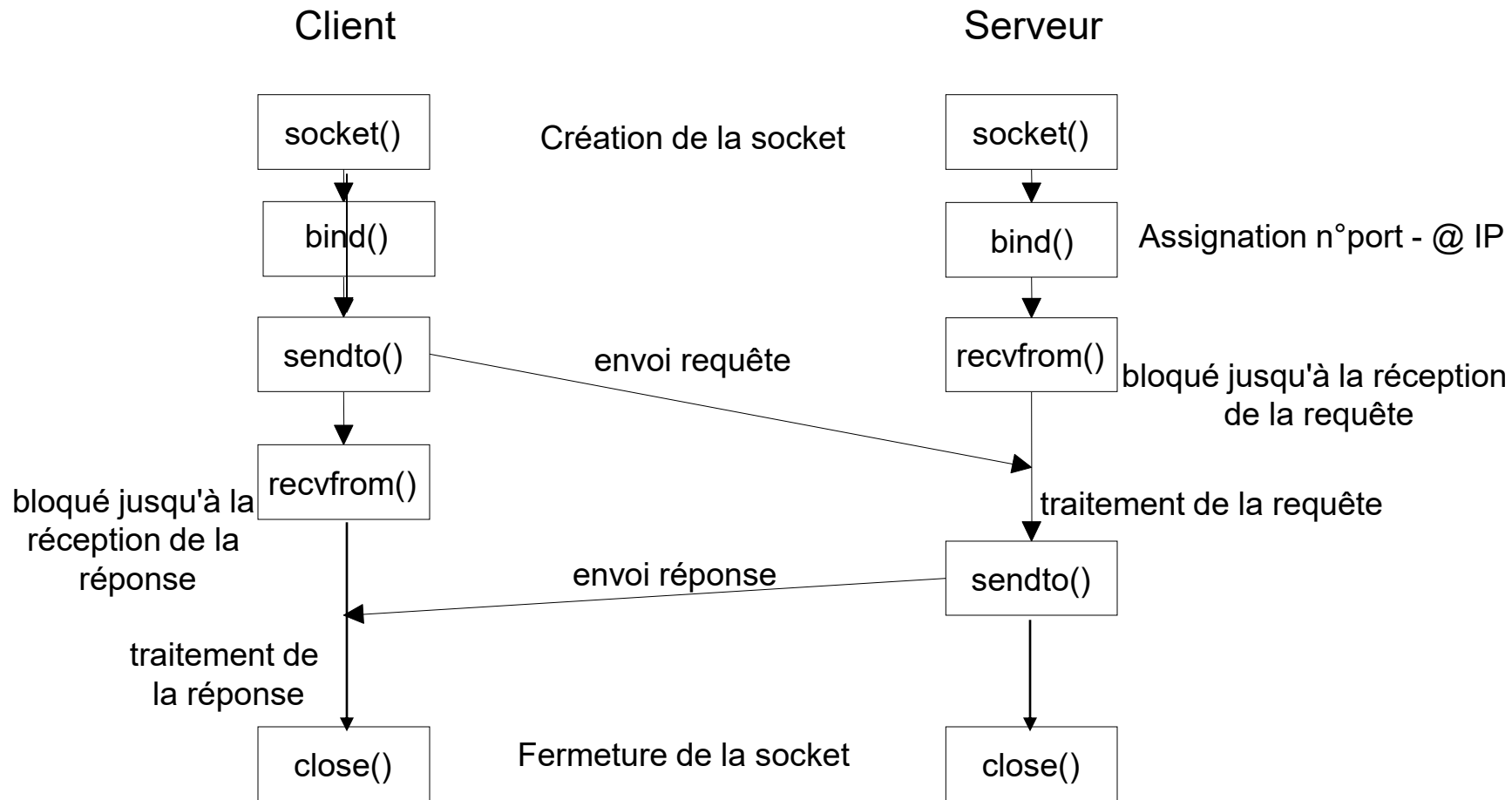
✦ Mode non connecté (UDP)

- Consomme moins de ressources
- Permet la diffusion
- Aucune gestion des erreurs,
➡ c'est la couche applicative qui doit gérer ce problème

Mode Connecté



Mode non connecté



Généralités (3)

✧ Une connexion

- ✧ @IP source, @IP destination, port source, port destination

✧ Une socket est un fichier virtuel sous Unix avec les opérations d'ouverture, fermeture, écriture, lecture ... (concept légèrement différent sous windows)

✧ Il existe différents types de sockets:

- ✧ Stream socket (connection oriented) → SOCK_STREAM
mode connecté = TCP
- ✧ Datagram sockets (connectionless) → SOCK_DGRAM
mode non connecté = UDP
- ✧ Raw sockets → SOCK_RAW
accès direct au réseau (IP, ICMP)

Programmation en C (1)

✧ Définition d'une socket

✧ **int s = socket (domaine, type, protocole)**

- *Domaine*

- ♦ AF_UNIX : communication interne à la machine
structure sockaddr_un dans <sys/un.h>
- ♦ *AF_INET : communication sur internet en utilisant IP*

- *Type*

- ♦ *SOCK_STREAM : mode connecté (TCP)*
- ♦ *SOCK_DGRAM : mode non connecté (UDP)*
- ♦ SOCK_RAW : utilisation directe niveau 3 (IP, ICMP,...)

- *Protocole*

- ♦ Actuellement non utilisé, valeur = 0.

Programmation en C (2)

✧ Attachement d'une socket

✧ **int error = bind (int sock, struct sockaddr *p, int lg)**

- *error* : entier qui contient le compte-rendu de l'instruction
 - ✧ 0 : opération correctement déroulée
 - ✧ -1 : une erreur est survenue (en général, problème dans p)
- *sock* : descripteur de la socket
- *p* : pointeur vers la zone contenant l'adresse de la station
- *lg* : longueur de la zone p



*Fonction définit avec sockaddr,
utilisation réelle de sockaddr_in (AF_INET) ou
sockaddr_un (AF_UNIX)*

Programmation en C (3)

✦ Adressage (inclure fichier <sys/socket.h> et <netinet/in.h>)

◆ **struct sockaddr_in {**
 short sin_family; ← domaine
 u_short sin_port; ← n° port
 struct in_addr sin_addr; } ← @IP système

◆ *sin_port* = numéro de port
 - soit laissé libre
 - soit affecté : htons (n°port) (conversion short -> réseau)

◆ *Struct in_addr sin_addr*

- Si serveur `sin_addr.s_addr = INADDR_ANY`
- Si client $\left\{ \begin{array}{l} \text{struct hostent *hp;} \\ \text{hp = gethostbyname(" ");} \\ \text{bcopy(hp->h_addr, \&x.sin_addr, hp->h_length);} \end{array} \right.$

Programmation en C (4)

✧ Primitives du serveur

◆ **int error = listen (int sock, int taille)**

- *error* : entier qui contient le compte-rendu de l'instruction
 - ◆ 0 : opération correctement déroulée , -1 erreur
- *taille* : nombre de requêtes maximum autorisées



Permet de créer une file d'attente pour recevoir les demandes de connexion qui n'ont pas encore été prises en compte

◆ **int scom = accept (int sock, struct sockaddr *p, int *lg)**

- *struct sockaddr* : stockage de l'adresse appelant
 - *lg* : longueur de l'adresse appelant
- *scom* : nouvelle socket permettant la communication



Primitive bloquante

Programmation en C (5)

✧ Ouverture d'une connexion, côté client

✦ **int error = connect (int sock, struct sockaddr *p, int lg)**

- *error* : 0 : opération correctement déroulée , -1 erreur
- *struct sockaddr* : adresse et port de la machine distante (serveur)
- *lg* : taille de l'adresse de la machine distante

✧ Fermeture d'une connexion

✦ **int close (int sock)**

→ le plus utilisé

✦ **int shutdown (int sock, int how)**

- *how* : 0 : réception désactivée,
1 : émission désactivée,
2 : socket désactivé

Programmation en C (6)

✧ Emission de données en mode connecté

◆ **int send (int sock, char *buf, int lg, int option)**

- *option* : 0 rien, MSG_OOB pour les messages urgents

◆ **int write (int sock, char *buf, int lg)**

- utilisable seulement en mode connecté, permet d'écrire dans un descripteur de fichier

✧ Emission de données en mode non connecté

◆ **int sendto (int sock, char *buf, int lg, 0, struct sockaddr *p, int lg_addr)**

- *p* : contient l'adresse du destinataire
- *lg_addr* : longueur de l'adresse destinataire

Programmation en C (7)

✧ Réception de données en mode connecté

◆ **int recv (int sock, char *buf, int lg, int option)**

- *option* : 0 rien, MSG_OOB pour les messages urgents, MSG_PEEK lecture des données sans les retirer de la file d'attente

◆ **int read (int sock, char *buf, int lg)**

- utilisable seulement en mode connecté, renvoie le nombre d'octets réellement lus.

✧ Réception de données en mode non connecté

◆ **int recvfrom (int sock, char *buf, int lg, 0, struct sockaddr *p, int lg_addr)**

- *p* : contient l'adresse de la source
- *lg_addr* : longueur de l'adresse source

Programmation en C (7)

✧ Quelques fonctions (obsolète)

◆ **int getsockname(int sock, struct sockaddr *p, int lg)**

- permet de récupérer l'adresse locale d'une socket, et son numéro de port
- ntohs(p->sin_port) pour afficher le numéro du port
- inet_ntoa(p->sin_addr.s_addr) pour afficher le nom de la machine

(Attention : peut provoquer des segmentation fault !!! → getnameinfo())

✧ Fonctions bloquantes/ non-bloquantes

◆ **fcntl(int sock, F_SETFL, O_NONBLOCK)**

◆ **ioctl(int sock, FIONBIO, 0/1)** → (0 : bloquant, 1, non bloquant)

- permet de rendre une socket bloquante ou non bloquante en lecture/écriture
- si pas de données en lecture, renvoie err=-1 et errno=EWOULDBLOCK

Programmation en C (8)

✧ Programmation sous windows

- ✧ **Identique sauf obligation d'initialiser la librairie qui gère les sockets :**

```
#include <winsock2.h>
```

```
#include <iostream>
```

```
#pragma comment(lib, "ws2_32.lib")
```

```
WSADATA info;
```

```
if (WSAStartup(MAKEWORD(2,0), &info) == SOCKET_ERROR)
```

```
{ cout << "erreur dans l'initialisation " << endl;
```

```
    WSACleanup();
```

```
    exit(1);
```

```
}
```

```
int socket ; struct sockaddr_in adr; struct hostent *entree; ...
```


Programmation en java (1)

✧ Création d'une socket en mode connecté et connexion

◆ Côté client

- *Socket sock = new Socket (nom_serveur, n°port)*
 - ◆ permet la connexion directe en TCP
 - ◆ plus de recherche de nom

◆ Côté serveur

- *ServerSocket sock = new ServerSocket (n°port)*
Socket scom = sock.accept()
 - ◆ la communication s'établit avec la socket scom
 - ◆ Plus besoin d'attachement, c'est automatique

Programmation en java (2)

✧ Lecture/ écriture sur des sockets en mode connecté

plusieurs possibilités

✧ **cas possible pour une socket soc**

- *lecture*

```
Reader reader = new InputStreamReader(soc.getInputStream())  
BufferedReader texte = new BufferedReader(reader);  
line = texte.readLine();
```

- *écriture*

```
Printstream sortie = new PrintStream(sock.getOutputStream());  
sortie.println(line);
```

Programmation en java (3)

✦ Création d'une socket en mode non connecté

◆ Côté client

- *DatagramSocket sock = new DatagramSocket ()*

◆ Côté serveur

- *DatagramSocket sock = new DatagramSocket (n°port)*

◆ Emission/réception

- *DatagramPacket msg = new DatagramPacket(buffer, taille, serveur, n°port)*
sock.sent(msg);
- *DatagramPacket recu = new DatagramPacket(buffer, taille);*
sock.receive(recu);
 - recu.getAddress()* -> adresse de l'expéditeur
 - recu.getPort()* -> N° port de l'expéditeur
 - recu.getData()* -> les données.

Programmation en java (4)

✧ Utilisation de la classe InetAddress

◆ **Récupération de l'adresse IP de la machine**

- *InetAddress a = InetAddress.getLocalHost();*
- *InetAddress a = InetAddress.getByName("....");*
 - *getHostName()* : nom de la machine
 - *getHostAddress()* : numéro IP de la machine
 - *toString()* : affiche les deux informations précédentes.

Plusieurs clients possibles

✧ Actuellement, le serveur ne gère qu'un client à la fois

→ Les autres sont mis en attente

- ✧ Utilisation possible de thread

- ✧ Utilisation de nouveaux processus (fork)
(après le accept)

✧ Primitive recv est bloquante, et Entrée clavier aussi

- ✧ Sous Unix, tout est fichier

 - Utilisation de la primitive **select()** possible ou **thread**

- ✧ Sous Windows

 - Utilisation de **thread**

Exemples

✦ Exemples dans le répertoire

http://perso.isima.fr/~palauren/prog_socket/

- [Socket_c.docx](#) ->> exemple C pour IPv4, IPv6, Csocket
- [Socket_java.docx](#) ->> exemple en Java pour Ipv4, Ipv6
- [Socket_python.docx](#) ->> exemple en python pour IPv4, Ipv6

(fichiers aussi au format pdf)