POLYTECH LYON

Année universitaire	**2017/2018**		
Département	**Informatique**	**Année**	**3A**
Matière	**Architecture des Application réparties (ARAR)**		
Enseignant	**C. Gertosio**		
Intitulé TD/TP :	**Support du cours**		

TABLE DES MATIERES

INTRODUCTION **4**

CHAPITRE 1. MODELE GENERAL D'ARCHITECTURE INFORMATIQUE CLIENT/SERVEUR. **5**

1. L'ORGANISATION DU SERVEUR **5**

2. LA COMMUNICATION DANS LE MODELE CLIENT/SERVEUR **6**

3. LES ARCHITECTURES MULTI TIERS **6**

4. LES DIFFERENTS TYPES DE SERVEURS **7**

CHAPITRE 2. LE SOCLE DE COMMUNICATIONS: LES SOCKETS PF_INET **8**

1. DEFINITION DE LA COMMUNICATION AU SENS DES SOCKETS **8**

2 LES SOCKETS ET JAVA **8**

3 GESTION DE LA COMMUNICATION AVEC UTILISATION DU PROTOCOLE UDP. **8**

3.1 Principe d'une application utilisant UDP **8**

3.2 Construction d'un datagramme DTG. **9**

3.3 Envoi et Réception d'un datagramme **10**

4 GESTION DE LA COMMUNICATION AVEC UTILISATION DU PROTOCOLE TCP. **10**

4.1 Principe d'une application utilisant TCP. **10**

4.2 Les échanges de données sur la connexion. **11**

4.3 La fermeture de la connexion. **11**

CHAPITRE 3. LE PROTOCOLE TFTP (RFC 1350) **12**

1 LA CONNEXION TFTP **12**

1.1 TFTP packets **12**

1.2 La fin de la connexion TFTP **14**

1.3 Exemples de transferts TFTP **14**

CHAPITRE 4. LE PROTOCOLE HTTP ET LES SERVEURS WEB 17

1 LES PROTOCOLES HTTP 1.0 ET 1.1 (RFC 2068) 17

1.1 Le principe http 17

1.2 La version Http 1.1 (RFC 2068 mis à jour par RFC 2616) 18
1.2.1 Etats supposés par http1.1 pour les machines client et serveur 18
1.2.2 Les méthodes http 18
1.2.3 Format générique des messages HTTP 19
1.2.4 Les headers 23

2 GESTION DES CONNEXIONS ENTRE CLIENT HTTP ET SERVEUR HTTP 27

2.1 Gestion des connexions persistantes. 27

CHAPITRE 5. LES SERVEURS BASES SUR LES DOCUMENTS, LES SERVEURS MULTIMEDIA 28

1 SYSTEMES DISTRIBUES BASES SUR LES DOCUMENTS (WEB) 28

1.1 Les processus clients et serveurs Web 28

1.2 Les serveurs dupliqués (cluster server) 29

1.3 Amélioration des performances de http 29
1.3.1 L'interface CGI (Common Gateway Interface). 31
1.3.2 Le server-side script 32
1.3.3 Les applets et servlets 32

2 LES APPLICATIONS MULTIMEDIA (AUDIO) DE L'INTERNET 33

2.1 Le principe de base 33

2.2 Les serveurs audio, streaming 34

CONCLUSION 36

BIBLIOGRAPHIE 37

INTRODUCTION

Les logiciels de communications de niveau bas (physiques, liaisons de données et réseaux) ont pour but d'organiser entre les utilisateurs, un échange de données plus ou moins fiable. La connaissance de ces logiciels constitue le cours de réseau dispensé dans les modules Réseaux & Transmission et Réseaux d'ordinateurs.
Le présent polycop constitue une suite dans la mesure où il a pour objectif d'étudier les mécanismes mis en œuvre dans les logiciels qui assistent directement l'utilisateur lors de la mise en place des applications distribuées. Autrement dit, considérant que les données peuvent être échangées entre les systèmes informatiques, ce cours se focalise sur les échanges réalisés par les applications de type client-serveur pour le transfert de fichiers et le WEB.

Chapitre 1. Modèle général d'architecture Informatique client/serveur.

Une application en réseau de type client/serveur suppose:

- ✓ Qu'au moins 2 tâches (processus) soient impliquées,
- ✓ Que chaque processus soit clairement identifié comme client ou serveur. Le processus client fait une demande de service. Le serveur fournit ce service. Ce mécanisme, appelé **Request-Reply behavior**, constitue la base du modèle client/serveur.

Un processus **serveur**, chargé de fournir le service, existe préalablement à toute demande de service. **Il est perpétuellement en attente de demandes de services.**
Un **processus client** invoque des services (exécution de procédures ou de méthodes) sur le serveur. La durée de vie du processus client est quelconque alors que le processus serveur doit être lancé par un administrateur du service avant toute requête. Les modes de travail sur le modèle client/serveur sont variés. Ils dépendent des applications. Des mécanismes généraux ont toutefois été mis au point pour aider les concepteurs d'applications.

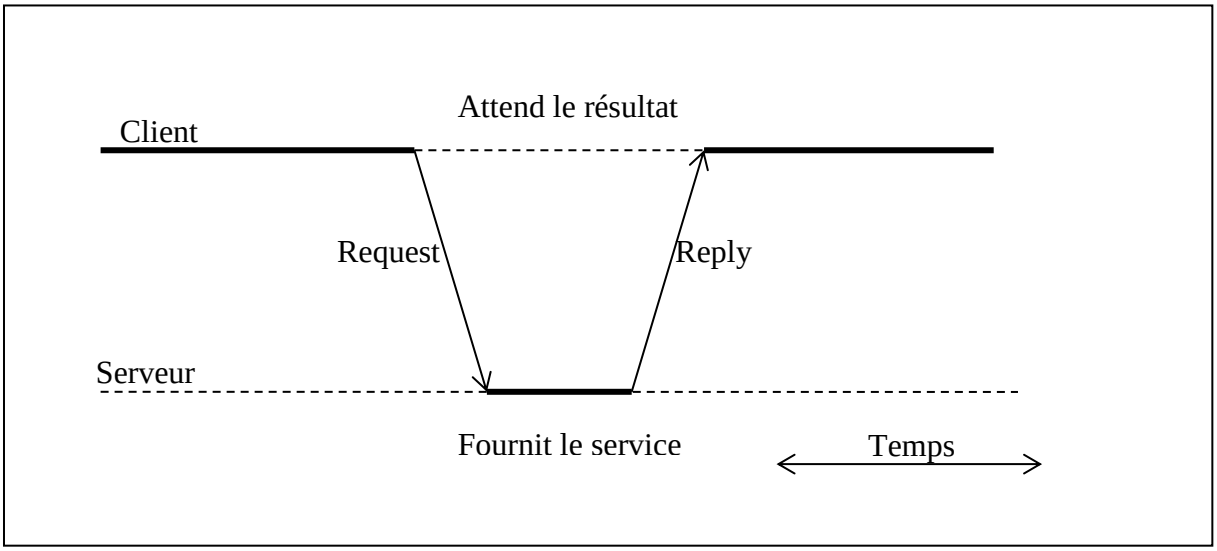


Figure 1 Relation entre client et serveur

1. L'ORGANISATION DU SERVEUR

Le nombre de clients potentiels d'un serveur est a priori quelconque. Le serveur peut être composé d'un pool de processus. Chaque processus pouvant être programmé pour traiter les d'1 ou plusieurs clients. **Le mode d'exécution assuré par le serveur au niveau réseau est une caractéristique du service**.
Le serveur peut fonctionner sans connexion de niveau réseau (avec UDP) et

- **Ne traiter qu'une requête à la fois,**
  - Il sert séquentiellement des clients différents sur un mode question/réponse unique exemple: donner l'heure au client.
  - 1 seul processus et 1 même port.

- **Traiter plusieurs requêtes à la fois en parallèle**

- Servir les clients en parallèle sur 1 mode question/réponse avec le client,
- Le serveur **dialogue** avec chaque client sur un port différent. Plusieurs questions et plusieurs réponses peuvent se suivre entre un même client et le serveur.
- 1 port d'écoute serveur puis affectation d'un port spécifique à chaque nouveau client avec création d'un nouveau port.

Le serveur peut **fonctionner avec connexion de niveau réseau (avec TCP)** et

- **Etre réservé à un seul client** :
  - le client se connecte sur le serveur qui ne traite plus que les requêtes de client jusqu'à ce que la connexion soit fermée par le client ou le serveur.
  - 1 seul processus et 1 seul port.
  - Lenteurs,
- **Etre concurrent et parallèle:**
  - traiter chaque client sur une nouvelle connexion
  - Chaque client dialogue avec le serveur sur un port spécifique dans le cadre d'**une connexion**.
  - Plusieurs processus et plusieurs ports: chaque processus fonctionne sur 1 port et un contexte mémorise l'état d'avancement des travaux effectués par le client.

On a coutume de distinguer les serveurs (services) sans mémoire des serveurs (services) à mémoire.

Le serveur ou service sans mémoire

C'est le cas le plus simple d'application client/serveur. Le serveur attend une demande de service en provenance d'un client. Il traite la demande, renvoie le résultat d'exécution positif ou négatif puis passe au client suivant. Il ne mémorise rien du service rendu au client. Un serveur d'impression peut être vu comme un serveur sans mémoire.

Serveur ou Service à mémoire

Lorsqu'un service a **une mémoire**, il est capable de garder la trace (résultat) des dernières invocations faites par le client. Ceci suppose que le serveur maintienne **dans le temps** le lien avec le client. Ce lien est matérialisé soit par le fait que chaque client utilise un port différent du serveur pour communiquer soit par une connexion réseau entre le serveur et chacun des clients.

2. LA COMMUNICATION DANS LE MODELE CLIENT/SERVEUR

Trois grands types de communication client/serveur sont envisageables:

- Communication de bas niveau via les sockets TCP/IP. Développements spécifiques d'applications communicantes. Fastidieux mais performant.
- Communication avec un serveur de BD supportant SQL (protocole basé sur TCP). *C'est le monde des serveurs de BD que nous ne traiterons pas dans ce cours*.
- Communication via des appels de méthodes de plus haut niveau. Ce mode correspond au modèle distribué: appels à des RPC ou utilisation d'objets distants (Remote Objects).

3. LES ARCHITECTURES MULTI TIERS

Lorsque l'on considère un modèle client/serveur, on n'exclue pas que le serveur peut se comporter en client vis-à-vis d'un autre serveur (architecture 3-tiers).

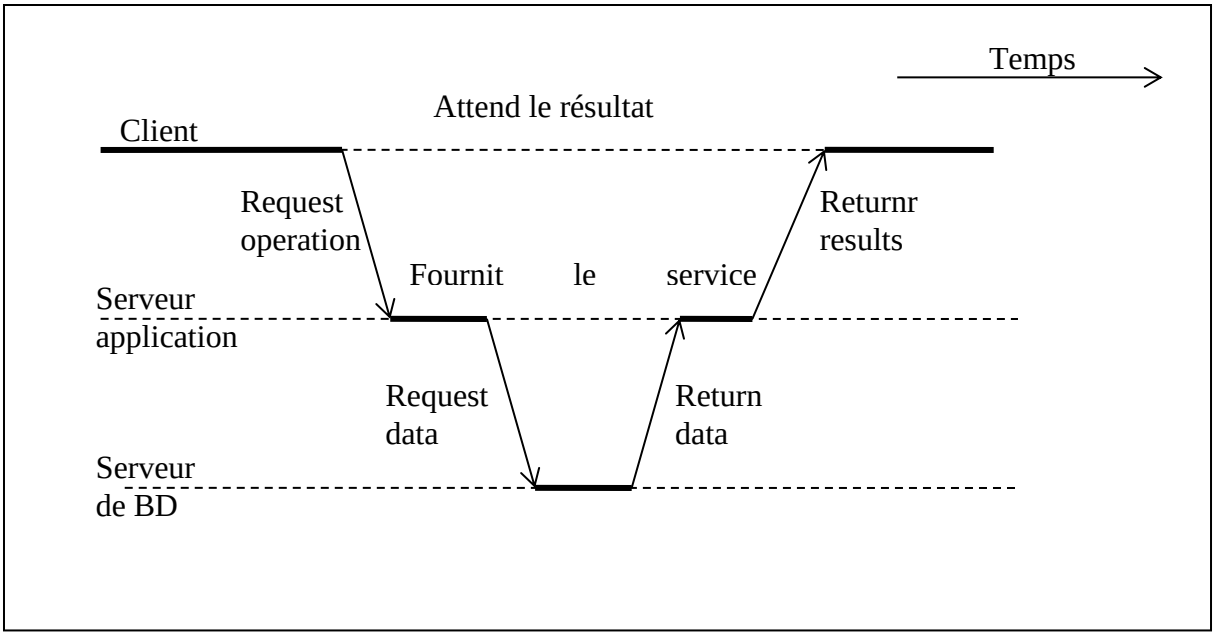


Figure 2: Exemple d'un serveur d'application agissant comme un client

La figure 2 illustre le cas d'un serveur d'application qui accède à un serveur de BD pour obtenir des données qui lui permettront de fournir le service au client. L'exemple de la figure 2 est appelée **architecture multi tiers** ou **distribution verticale.**.
Cette distribution n'est pas la seule possible. Dans certains cas, on a besoin de déployer plusieurs serveurs d'application identiques qui rendent tous le même type de service au client. Dans ce cas, on parle de **distribution horizontale**. Le serveur est alors répliqué sur plusieurs machines accessibles par les clients. Un exemple est traité dans le chapitre des serveurs Web. Cela pose le problème de la mise à jour des informations disponibles sur cet ensemble de serveurs identiques.

4. LES DIFFERENTS TYPES DE SERVEURS

De manière pratique, les serveurs sont spécialisés. On peut les classer en grandes familles:

- **Le serveur de procédures** qui permet d'exécuter des procédures distantes. Ce serveur est souvent intégré dans un service plus complet : NFS ou environnement distribué de type DCE.
- **Le serveur d'objets** qui exécute la logique applicative, protocole d'objets distribués qui est utilisé par les applications
- **Le moniteur transactionnel** dont le but est de coordonner l'exécution de transactions distribuées sur plusieurs machines; par exemple réserver avion + hôtel et il existe un serveur pour les places d'avions et un autre pour les chambres d'hôtel. Dans ce cas l'application devra s'assurer que la transaction se fait totalement ou pas du tout.
- **Le serveur Web** qui a le rôle de serveur de fichiers ou de middleware.
- **Le serveur de sécurité** qui gère l'authentification des utilisateurs et le chiffrement des flux de communication. La norme LDAP (Lightweight Directory Access Protocol) mise en œuvre par Netscape s'impose comme un standard d'annuaire distribué. LDAP au centre des system PKI.

Chapitre 2. Le Socle de communications: les sockets PF_Inet

Les sockets Unix, situés à la frontière entre l'espace utilisateur et le noyau Unix, permettent aux programmes d'un même calculateur, d'échanger des données. Avec le même principe, les sockets du domaine INET permettent à deux applications situées sur deux machines du réseau Internet d'échanger des données**. Les sockets INET sont à la frontière entre l'espace utilisateur des applications et l'ensemble des logiciels de communication de l'Internet niveau 4 compris.** Autrement dit, les sockets Inet constituent une interface de programmation permettant aux applications d'utiliser les services de TCP et UDP pour communiquer avec les applications distantes. Dans ce chapitre nous allons étudier les mécanismes généraux des applications distribuées qui utilisent les sockets pour communiquer.
On suppose dans ce chapitre que les protocoles TCP et UDP sont connus (polycop RESO)

1. DEFINITION DE LA COMMUNICATION AU SENS DES SOCKETS

La communication par sockets est établie pour le compte d'une application distribuée, entre 2 machines distantes, toutes deux reliées au même réseau Internet.
Le système par sockets considère que la communication n'est pas symétrique entre les 2 machines, (au moins dans sa première phase). En effet, il faut attribuer à chaque extrémité le rôle de client ou de serveur.
Le client correspond à l'extrémité qui émet la demande d'ouverture, il est **initiateur**. Le serveur correspond à l'extrémité qui attend les demandes d'ouvertures, il est **accepteur**.
Pour une communication, chaque machine gère, en local, 3 valeurs : le protocole, l'adresse IP appelée InetAddress, et le numéro de port. Au sens des sockets, une communication est entièrement définie par 5 paramètres:

- Le protocole TCP ou UDP utilisé,
- Inet Address de la première machine,
- Le numéro de port associé au processus (application) s'exécutant sur la première machine,
- Inet Address de la deuxième machine,
- le numéro de port associé au processus (application) s'exécutant sur la deuxième machine.

2 LES SOCKETS ET JAVA

Le système de sockets est également implanté dans le monde JAVA. Le paquetage **Java.net** possède trois classes pour représenter les sockets selon les protocoles UDP et TCP : **DatagramSocket pour UDP et ServerSocket et Socket pour TCP.** Une documentation détaillée vous a été distribuée pour les TP.

3 GESTION DE LA COMMUNICATION AVEC UTILISATION DU PROTOCOLE UDP.

3.1 Principe d'une application utilisant UDP

On crée une communication en utilisant, à chaque extrémité, le protocole UDP qui fonctionne sans connexion préalable entre les applications. Toutefois, les échanges de données ne pourront se faire correctement sans un minimum d'organisation : définition d'un dialogue entre client et serveur. En général, lorsque l'on crée une application utilisant UDP pour communiquer, on prévoit un premier échange de datagrammes initié par le client. Celui-ci

envoie un datagramme du style « client YYY», le serveur s'il a été démarré, et s'il reconnait ce message peut répondre Hello serveur XXX écoute ». La figure 3 montre un exemple d'échange UDP entre client et serveur.

1. Le serveur ouvre « son» port d'coute fixé (et connu des clients) et se met en attente du datagramme initial (DTG),
2. Le client ouvre un port local et envoie le premier DTG (généralement prédéfini) puis il se met en attente du DTG réponse serveur.
3. Le serveur reçoit le DTG et renvoie au client sa réponse (DTG visant à s'identifier) et il se met en attente d'un nouveau DTG client.
4. Le client et le serveur échangent ainsi des DTG selon les besoins de l'application.
5. A la fin de l'échange de données; le client doit fermer le socket local pour libérer le port. Le serveur fermera également le port sue lequel il a dialogué avec le client: soit parce-que l'échange applicatif est terminé soit grâce à une temporisation d'inactivité.

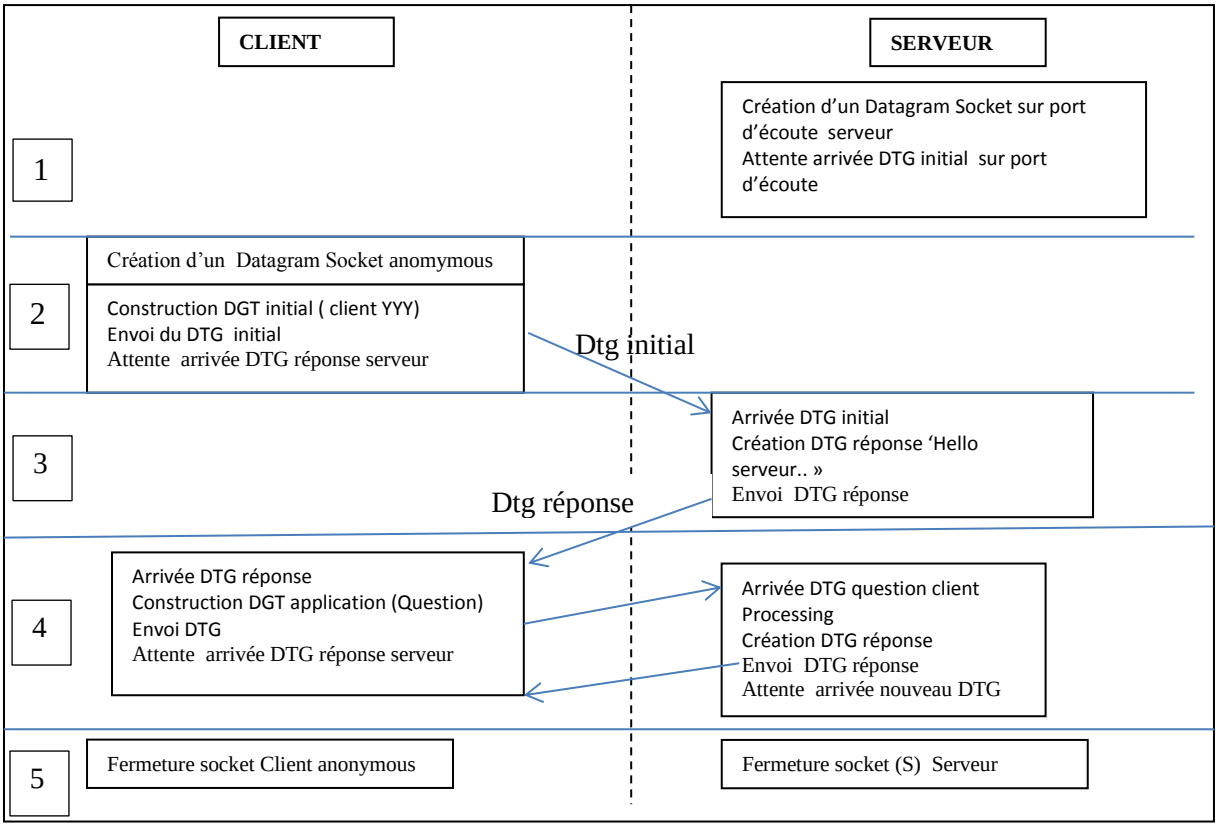


Figure 3 Exemple de dialogue en mode client/serveur avec UDP

3.2 Construction d'un datagramme DTG.

Un datagramme est une suite d'octets conforme au protocole UDP. Il faut le construire en précisant essentiellement 3 paramètres: les données et leur longueur, l'adresse Inet de la machine destination ainsi que le port de destination. Un constructeur Java permet de créer ce datagramme: classe DatagramPacket. La taille maximale d'un DTG est 65535 octets. Par défaut, la taille maximale est de 8192 octets en Java. C'est au moment de la construction du DatagramPacket en vue de son envoi que l'on doit préciser l'adresse Ip et le port de destination. Le constructeur le plus courant est du type :

Public DatagramPacket (Z_envoi, lg_Zenvoi, Inet adr_dest, port_dest)

Z_envoi est un tableau d'octets contenant le message à envoyer, lg Z_envoi contient la longueur de Z_envoi, Inet_adr_dest est l'adresse destination et Port_dest désigne le port destination du datagramme.

3.3 Envoi et Réception d'un datagramme

Un objet DatagramSocket, appelons-le Soc, possède 3 méthodes : send(dp) ; receive(dp) et close ().

- La méthode **send(dp)** permet d'envoyer le datagramme dp.
- La méthode **Receive (dp)** permet de recevoir un DTG à la fois et de le placer dans un objet dp de type DatagramPacket préexistant. Cette méthode bloque l'application jusqu'à ce qu'un Datagramme arrive. En ce qui concerne le DTG reçu, on peut connaitre sa provenance en utilisant les méthodes GetAddress() et GetPort() (cf. documentation Java TP disponible en TP). Le datagramme dp doit être assez grand pour recevoir la totalité des données envoyées. Si ce n'est pas le cas, le DTG reçu est tronqué.
- La méthode **close()** ferme le socket et libère le port utilisé.

4 GESTION DE LA COMMUNICATION AVEC UTILISATION DU PROTOCOLE TCP.

Dans ce cas, il faut **créer une connexion** entre les deux extrémités par le biais de TCP avant d'échanger les données.
Selon que l'on spécifie une extrémité client ou serveur fonctionnant avec TCP, on utilise 2 classes différentes. La classe **java.net.ServerSocket** est utilisée pour programmer la partie « serveur de l'application ». La classe **java.net.Socket** est utilisée pour programmer la partie « client de l'application».

4.1 Principe d'une application utilisant TCP.

Le travail d'un serveur est de veiller en attente d'une demande de connexion entrante TCP arrivant d'un client. Chaque serveur attend sur un port particulier qui est forcément connu du client. La figure 4 montre la chronologie entre client et serveur fonctionnant sous TCP:

1. Créer un objet ServerSocket SS sur le port d'écoute du serveur,
2. Attendre les demandes de connexion client en utilisant la méthode accept(). La méthode accept() est bloquante jusqu'à ce qu'une demande de connexion soit faite par le client. Accept() retourne **un objet socket() conn_cli représentant la connexion client-serveur**.
3. Le client et le serveur échangent les données de l'application à l'aide des flux de données (cf. § suivant).
4. Enfin, client ou serveur ou les deux peuvent décider de clôturer la connexion. La méthode close de l'objet Socket() ne ferme que la connexion du client avec le serveur.

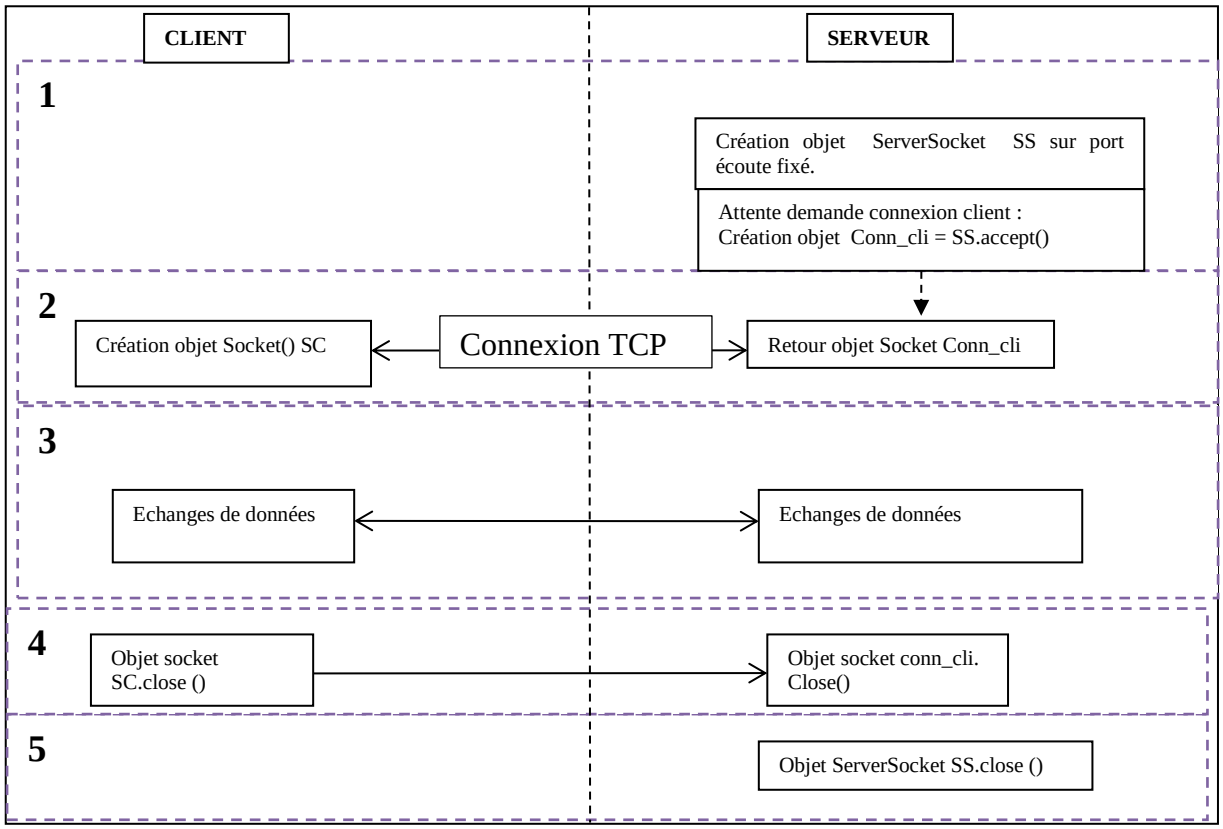


Figure 4 Chronologie d'une application fonctionnant avec TCP

4.2 Les échanges de données sur la connexion.

A partir du moment où la connexion est réalisée, l'application peut utiliser les méthodes getInputStream() et getOutputStream() de l'objet Socket pour réaliser ses échanges de données. La méthode getInputStream() retourne un objet InputStream (flux d'entrée) qui permet de lire les données sur la connexion c'est à dire sur l'objet socket. Les données peuvent être stockées en attente d'envoi (bufferisées), converties (voir les filtres java). La gestion de l'envoi des données est totalement assurée par le système de sockets.
La méthode getOutputStream() retourne un objet OutputStream (flux de sortie) qui permet d'écrire les données sur la connexion c'est à dire sur l'objet socket. Pour plus de précisions sur documentation Java TP disponible en TP)

4.3 La fermeture de la connexion.

C'est la méthode close() de l'objet Socket qui ferme la connexion coté client comme coté serveur.. La fermeture de la connexion par une extrémité provoque une exception sur l'autre extrémité. La méthode close() de l'objet ServerSocket() libère le port fixé.

Chapitre 3. Le protocole TFTP (RFC 1350)

Le protocole TFTP est un protocole très simple utilisé pour transférer les fichiers entre deux machines utilisant Internet et plus particulièrement le protocole UDP. Il prend en charge des fichiers de type:

- **Netascii:** chaque caractère est codé sur 8 bits. C'est un code ascii 8bits
- **Octet:** Ceci remplace le mode binaire. Ce sont 8 bits bruts. Aucun contrôle n'est réalisé sur le contenu de l'octet
- **Mail:** obsolète.

TFTP est structuré sur un mode client/serveur. Cela signifie que pour chaque transfert, il doit exister une application serveur qui attend une requête et une application client qui fait la requête au serveur. Un serveur TFTP attend les requêtes sur le port 69 (well known port). Bien entendu, rien ne s'oppose à ce que le serveur attende sur un autre port (voir TP). Un client peut faire une requête :

- de lecture d'un fichier sur le serveur,
- d'écriture d'un fichier sur le serveur.

Le serveur TFTP peut accepter ou refuser le transfert. Cet échange initial fait office de «connexion TFTP ». Si le serveur accepte la requête d'échange, le transfert du fichier se fait par bloc de 512 octets de données. Un bloc de données de taille inférieure à 512 octets signale la fin du transfert. Les blocs de données sont échangés dans des DATA packets. Chaque DATA packet reçu doit être acquitté par un paquet accusé de réception (ACKNOWLEDGMENT packet ou ACK packet).

1 LA CONNEXION TFTP

Un transfert est établi par un client par

1. **l'envoi d'une requête au serveur**: WRQ packet pour écrire un fichier sur le serveur ou RRQ packet pour lire un fichier du serveur.
2. **La réponse du serveur:** un ACK packet si WRQ, le premier DATA packet si RRQ.

Un ACK packet contient un numéro de DATA packet acquitté. Les blocs sont numérotés à partir de 1. Le numéro de bloc du Ack packet acquittant le RRQ est 0.
Afin de créer une «connexion TFTP» sur un environnement réseau de type non connecté (UDP), chaque extrémité choisit un TID (Transfer IDentification) qui sera valide durant le temps de la connexion. **Le TID peut être choisi aléatoirement mais une manière simple de procéder pour une extrémité, c'est de choisir son port source comme TID.**
Un client choisit son port source pour envoyer la requête initiale au serveur. Celui-ci attend sur un port d'écoute (TFTP = 69) et acquittera la requête sur un nouveau port qui sera utilisé par le client comme TID du serveur. Le reste du transfert se fera sur ce port. Avec ce principe, chaque transfert coté serveur sera réalisé sur un port différent. **Le port d'écoute initial** servira à recevoir et traiter les requêtes des clients.

1.1 TFTP packets

TFTP supporte 5 types de packets. Chaque paquet a un format spécifique mais tous commencent par le OPcode qui se présente toujours sous forme de **2 bytes**.

Opcode (2 octets)	**Operation**
1	Read Request (RQ)
2	Write Request (WRQ)
3	DATA
4	Acknowlegment (ACK)
5	ERROR

Les paquets RRQ/WRQ

2 bytes	string	1 byte	string	1 byte
Opcode = 1 ou 2	Nom fichier	0	Mode	0

Le nom de fichier est une chaine de caractères terminée par le byte nul. Le mode contient la chaîne «netascii», «octet» (en français !) ou «mail». Vous utiliserez «octet» dans le TP.

Le paquet DATA

2 bytes	2 bytes	n bytes (512 ou 1 à 511 bytes)
Opcode = 3	Num. de bloc	DATA

Les numéros de blocs commencent à 1 et sont incrémentés de 1 à chaque nouveau bloc émis. Un bloc DATA de 512 octets signale que ce bloc n'est pas le dernier du transfert. Une taille inférieure signale que c'est la fin du transfert.

Le paquet ACK

2 bytes	2 bytes
Opcode= 4	Num. de bloc

Tous les DATA packets sont acquittés par un ACK packet. Le numéro de bloc du ACK packet désigne le dernier bloc DATA bien reçu. Le numéro de bloc = 0 est l'accusé de réception positif de la demande WRQ.
Tant qu'un DATA packet n'a pas été acquitté par un ACK packet, l'émetteur doit le renvoyer. Le mécanisme prend fin lorsque soit l'acquittement arrive, soit l'émetteur renonce à l'envoi (par exemple 3 réémissions maximum).
La temporisation d'activité (activity timer) est lancée par une extrémité, lorsqu'elle attend un paquet DATA ou ACK. Lorsque cette temporisation expire, sans avoir reçu le paquet, la règle veut que cette extrémité renvoie son dernier paquet (qui peut être un DATA packet ou un ACK packet). Ainsi un émetteur doit garder le dernier paquet DATA ou ACK, qu'il peut être amené à retransmettre. Si un packet est perdu dans le réseau, cette retransmission systématique force l'émetteur du paquet perdu à le réémettre (cf. figure n° 7).

Le paquet ERROR

2 bytes	2 bytes	String	1 byte
Opcode= 5	Errorcode	ErrMsg	0

Le paquet ERROR est un acquittement pour n'importe quel paquet. Error code et Errmsg donnent le type d'erreur.
Plusieurs erreurs peuvent causer la fin du transfert. Une erreur est signalée par l'envoi d'un paquet d'erreur (ERROR packet). Un ERROR packet n'est pas acquitté et ne sera jamais retransmis. L'émetteur et le récepteur d'un ERROR paquet peuvent terminer l'échange tout de suite après l'envoi de ce paquet. Les erreurs peuvent être causées par 3 types d'événements :

- le serveur est dans l'incapacité à satisfaire la demande : fichier non trouvé, répertoire non accessible (Access violation), pas d'utilisateur,
- la réception d'un paquet que le récepteur ne peut comprendre (Incorrectly formed packet) ou bien qui n'est pas attendu au sens du protocole,
- Accès devenu impossible à une ressource nécessaire au transfert (disque plein, accès impossible..).

Il y a 7 erreurs possibles (reportez-vous à la norme RFC 1350 pour les connaitre). La réception d'un paquet ERROR cause la fin du transfert à l'exception du cas un Datagramme reçu ne porte pas le bon port source. Dans ce cas, un ERROR packet est envoyé en réponse à ce paquet, en utilisant le port source et l'adresse IP de l'émetteur du paquet.

1.2 La fin de la connexion TFTP

La fin du transfert débute lorsqu'un DATA packet de taille inférieure à 516 octets (data bloc contenant entre 0 et 511 octets). Ce DATA packet doit être normalement acquitté par un ACK packet comme les autres DATA packets.
L'émetteur du dernier DATA packet doit attendre le ACK packet pour conclure que le transfert s'est bien passé et clôturer la communication. Si la temporisation expire alors il devra re émettre ce dernier DATA packet.

1.3 Exemples de transferts TFTP

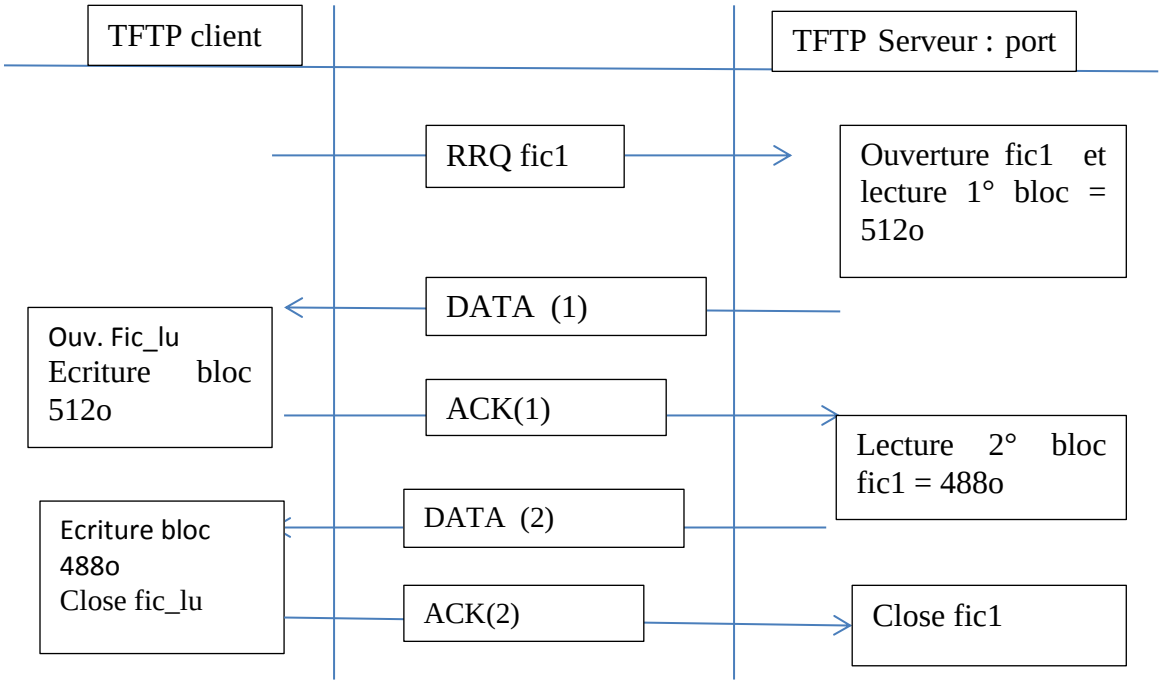


Figure 5 : lecture fichier de 1000 octets sur le serveur

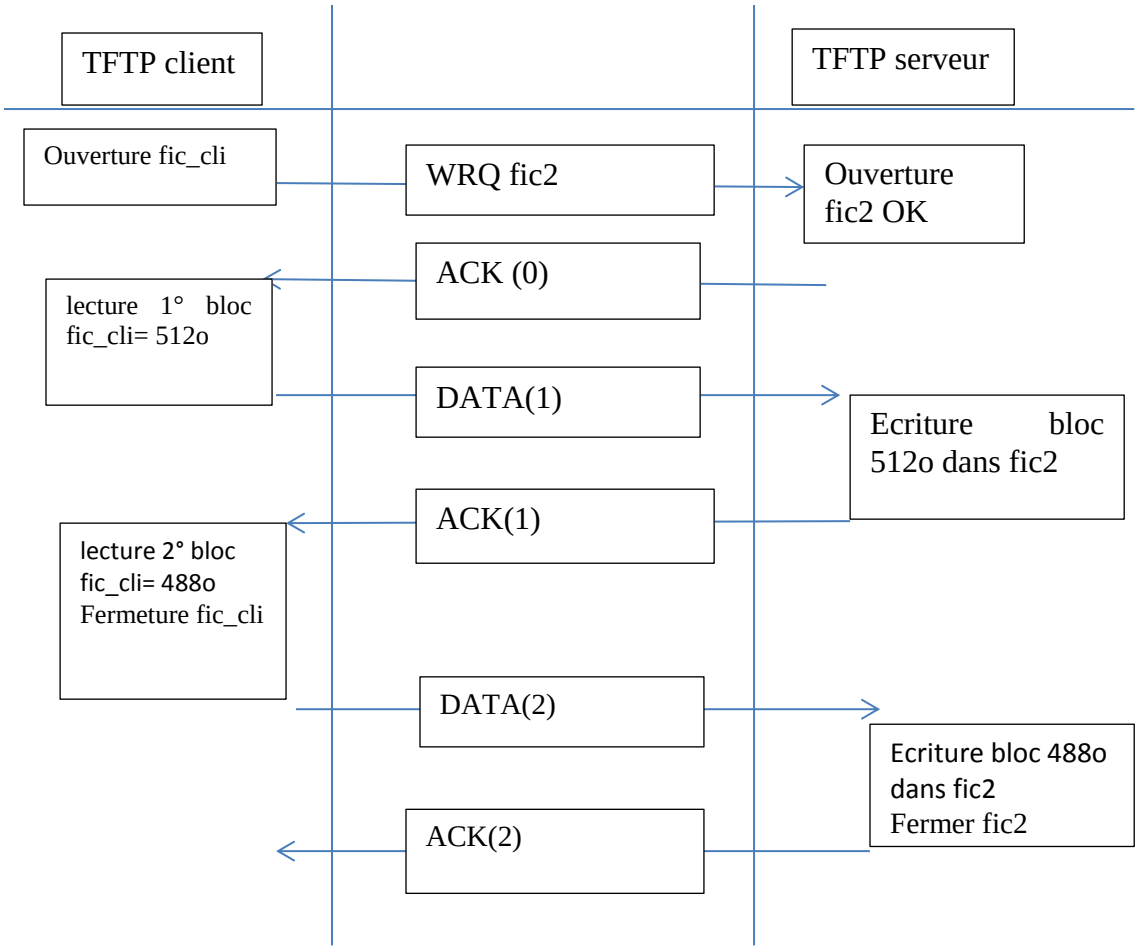


Figure 6: Ecriture d'un fichier de 1000 octets sur le serveur

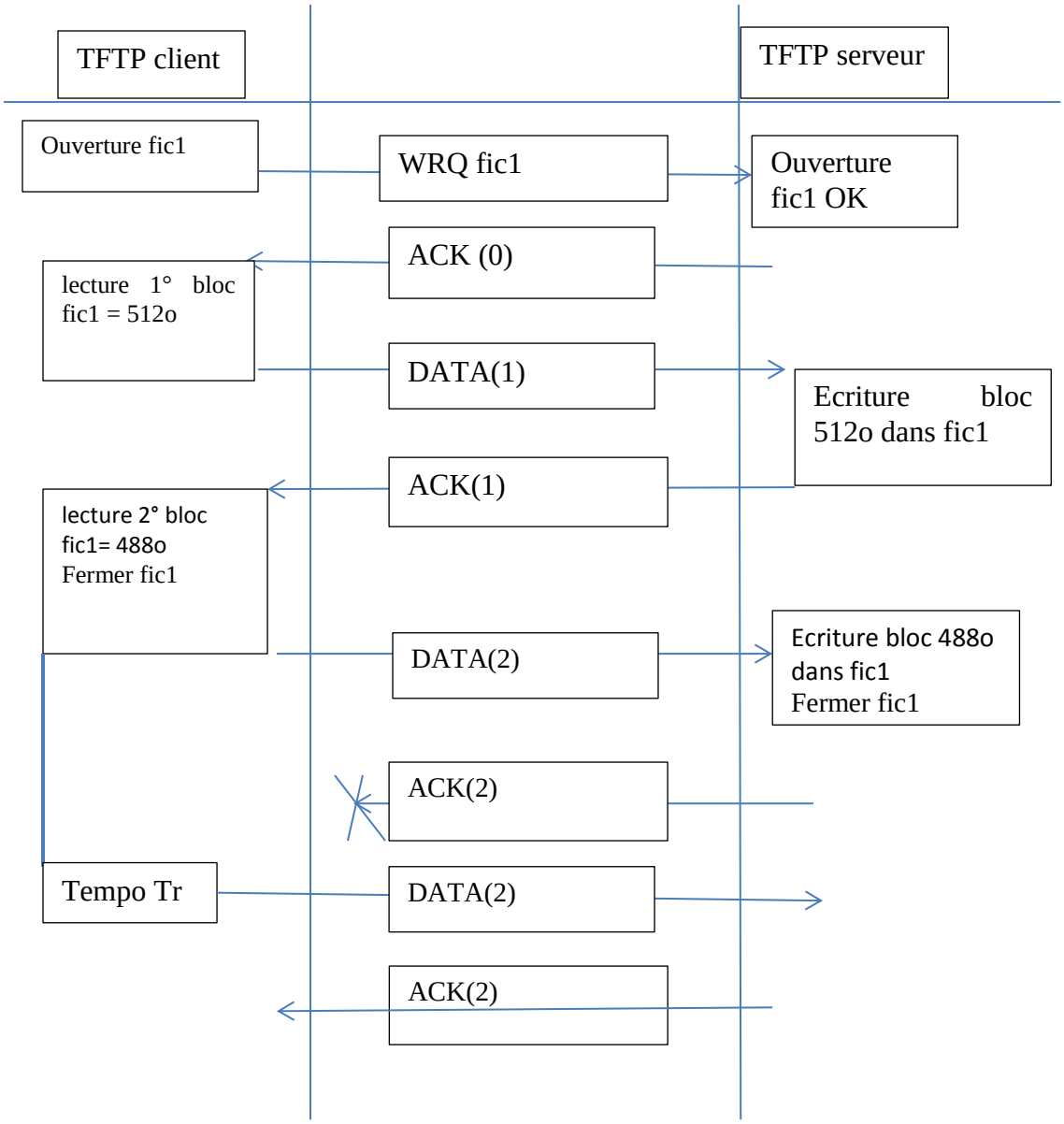


Figure 7: exemple d'écriture de fichier avec retransmission

Chapitre 4. Le protocole http et les serveurs WEB

Le World Wide Web (W3) est le premier réseau hypermedia réparti. Il a été crée en 1989 par **Tim Berners-Lee** qui l'a proposé pour la communication de l'information au CERN.
Ce projet a donné aux utilisateurs d'Internet un outil efficace pour accéder de manière simple à une grande variété de documents.
Le W3 est basé sur le modèle du client-serveur. Un **serveur W3** est un programme serveur qui s'exécute sur un ordinateur dans le seul but de répondre aux requêtes des logiciels clients W3. Un **client W3** est un programme qui permet à un utilisateur de soumettre une requête au serveur W3 et de visualiser le résultat. Un logiciel client WEB est plus connu sous le nom de **navigateur** ou **Browser** (Internet Explorer et Netscape, Mozilla Firefox). Un navigateur est universel, s'il permet d'accéder à plusieurs types de serveurs: www mais aussi FTP ou Gopher. Les règles d'échanges utilisées entre un client et un serveur W3 sont données par le protocole HTTP.

1 LES PROTOCOLES HTTP 1.0 ET 1.1 (RFC 2068)

HTTP (HyperText Transfer protocol) a été défini pour distribuer des documents hypermedia sur un réseau. Il peut être utilisé chaque fois que l'on doit diffuser un document texte, une image. Il définit les messages que les clients peuvent envoyer au serveur et ceux que le serveur retourne en réponse.
Tous les clients et serveur du WWW doivent respecter les spécifications de ce protocole (RFC 2068).

1.1 Le principe http

Http est basé sur le modèle du **client serveur**. Il suppose qu'un dialogue est déjà établi entre la machine serveur qui possède **le document** et la machine client qui veut récupérer **ce document**. Le protocole HTTP est implanté dans la machine client et dans la machine serveur. Il est seulement là pour assurer l'échange du **document** sous la forme **d'une** transaction **de type requête/réponse**. Chaque requête/réponse, faite sur le serveur, étant indépendante de la précédente et de la suivante.
Un document est la plus petite unité fournie par le serveur en réponse à une requête du client. Un document peut être une image, le contenu d'une page, un formulaire**. Un document s'appelle une ressource** et il est accessible au moyen d'une URL (Uniform Resource Locator). Le format d'une URL est le suivant :

méthode://**nom-machine**:port/nom-de-fichier

Le mot méthode indique le protocole qui est utilisé. On peut utiliser les protocoles HTTP, FTP, Telnet. Par exemple,

http://www.yoyodyne.com:1234/pub/files/foobar.html

Désigne le document **foobar.html** dans le répertoire /**pub/file** du serveur 3W www.yoyodyne.com en attente sur le port 1234.

Client **Serveur**
Requête (méthode) ---------->

<---------- Réponse

Quatre étapes sont nécessaires pour récupérer **un document** sur le serveur :
1. Connexion. Le client établit la connexion avec le serveur,
2. Request. L'envoi de la requête vers le serveur : donnez-moi le document 'file.html'
3. Response. Le serveur répond. Sa réponse contient le document ou la raison du refus.
4. Close. Le client ferme la connexion.
Il faut noter que le protocole **http** ne se préoccupe pas de l'établissement de la connexion ni de sa fermeture: ouverture et fermeture de connexion sont à la charge du client.
HTTP nécessite seulement un transport fiabilisé; tout protocole garantissant la sécurité de transmission peut être utilisé pour supporter HTTP. Sur Internet, les communications HTTP s'appuient principalement sur le protocole de connexion TCP/IP. Le port utilisé par défaut est le port TCP 80 mais HTTP peut fonctionner sur un autre port.

1.2 La version Http 1.1 (RFC 2068 mis à jour par RFC 2616)

Comme nous l'avons indiqué au paragraphe précédent, la première version de HTTP 1.0 ne prévoit pas de session permanente entre le client et le serveur: pour chaque requête, le protocole http client attend la réponse puis ferme la connexion. Autrement dit, les 4 étapes sont recommencées pour chaque document. A l'origine, cette méthode convenait bien car la page WEB ne contenait que du texte sous la forme HTML. Au fil des ans, la page WEB moyenne est devenue un conteneur de textes, d'icônes, d'image. L'établissement d'une connexion pour récupérer devient un seul icône alors un mécanisme coûteux.
D'où l'évolution: HTTP 1.1 suppose qu'il communique avec le serveur sur une connexion préexistante: dans ce cas, **une connexion peut être utilisée pour une ou plusieurs questions/réponses. Attention:** le protocole http 1.1 ne gère toujours pas les connexions.

1.2.1 Etats supposés par http1.1 pour les machines client et serveur

Http spécifie qu'une connexion doit être initiée par un client avant transmission de la requête, et refermée par le serveur après délivrance de la réponse. Les deux côtés, client et serveur, doivent être préparés à ce que la connexion soit coupée prématurément, suite à

- Une action de l'utilisateur,
- Une temporisation automatique,
- Une faute logicielle.

Ils doivent apporter une réponse prévisible à cette situation. Ainsi: dans tous les cas, la fermeture d'une connexion, quelle qu'en soit la raison, est assimilable à la conclusion de la requête, quel que soit l'état.

1.2.2 Les méthodes http

Ce terme **méthode** fait penser à des applications orientées objets: bien que Http ait été conçu pour fonctionner sur le WEB, ses propriétés ont été intentionnellement génériques. **Attention la casse est prise en compte! Toujours des majuscules pour les méthodes.** Les principales requêtes de **méthode**s sont:

– GET: requête de lecture de page. La méthode la plus connue, et de loin la plus utilisée, est la méthode GET qui permet à un client HTTP de se procurer une ressource (1 document) sur un serveur.

- HEAD: requête de lecture d'un en-tête de page, sans la page. Cette méthode peut servir pour obtenir la date de dernière modification d'un page, ou simplement à tester la validité d'une URL.
- PUT: requête de stockage de page, C'est l'inverse du GET. Il s'agit d'une opération d'écriture. Cette méthode permet d'écrire un ensemble de pages sur un serveur. Le corps de la requête contient la page. Il peut être encodé au format MIME
- POST: rajout du contenu d'une requête aux données existantes, par exemple une page WEB. La méthode POST ressemble à la méthode PUT.
- DELETE: suppression de la page indiquée. TRACE : demande de retour d'une requête entrante,
- CONNECT: réservé pour usage futur,
- OPTIONS: requête d'informations sur certaines options.

La grande majorité des requêtes émises par HTTP correspond à la méthode GET et HEAD. Les requêtes PUT, POST et DELETE ne peuvent être mises en œuvre qu'avec un mécanisme d'authentification et de permission.

Exemple:

GET*spnomfichier sp*HTTP/1.1

L'argument *nomfichier* indique la ressource (fichier) à récupérer, 1.1 est la version du protocole utilisé.
Les méthodes GET et HEAD doivent être supportées par tous les serveurs. Toutes les autres sont optionnelles: toutefois si elles sont implantées, elles doivent suivre la sémantique donnée par la norme.

1.2.3 Format générique des messages HTTP

Un message HTTP est crée pour chaque **requête** et chaque **réponse**. Le format du message HTTP est **unique**.

http_message = Request| Response

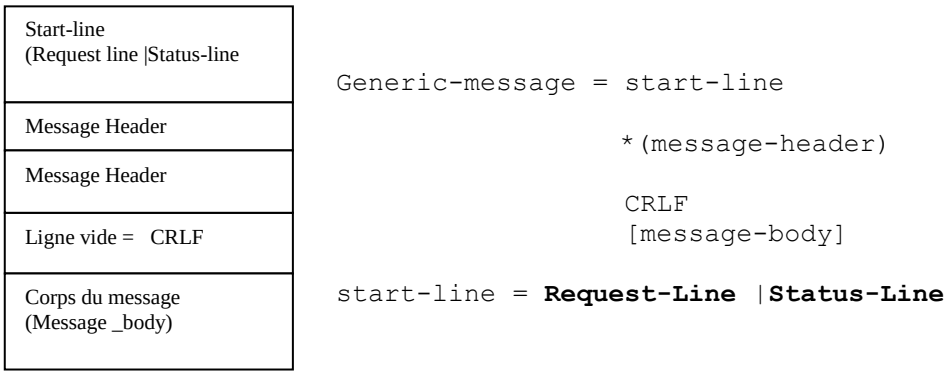


Figure 8 Le format unique d'un message http

Les requêtes et les réponses utilisent le format générique du message http (figure 8)
Ce format prévoit **1** start-line, **zéro ou plusieurs** "message headers" (entête) et **1** ligne vide (CRLF) indiquant la fin des headers. Puis le corps du message (message_body) qui est facultatif. Si message_body est absent, le message http se termine par la ligne vide.

Une Start_line est soit une Request_line soit une Status_line.

Le message-body est utilisé pour transporter the entity-body (s'il existe), associé à la requête ou à la réponse. Dans le cas de la réponse à une requête GET, message _body est le fichier demandé.

Remarque

CR (Carriage Return) représente 1 octet contenant la valeur 13 ou 0dh. Ce caractère, envoyé sur un écran, renvoie le curseur en début de ligne.
LF (Line Feed) représente 1 octet contenant la valeur 10 ou 0Ah. Ce caractère, envoyé à un écran, fait passer le curseur à la ligne suivante.
Une ligne vide sera composée des 2 caractères CR suivi de LF.

Exemple d'invocation de méthode GET avec la réponse

GET http://www.dil.univ-mrs.fr/~jfp/test.html HTTP/1.0 ***CRLF***

HTTP/1.1 200 OK CRLF
Date: Thu, 04 Nov 2004 11:30:07 GMT CRLF
Server: Apache/1.3.12 (Unix) CRLF
Last-Modified: Thu, 04 Nov 2004 11:30:16 GMT CRLF
Content-Length: 17 CRLF
Connection: close CRLF
Content-Type: text/html CRLF
CRLF
texte du document

Format_Request_Line

Request line = method SP Request_URI SP HTTP/x.x CRLF

Les différents paramètres sont séparés par des espaces (SP= espace). La méthode est toujours en MAJUSCULES: la casse est prise en compte dans la requête.

Request_URI c'est la ressource demandée. Le paramètre request-URI désigne l'identificateur de la ressource Uniform Resource Identificator. La ressource demandée doit être totalement identifiée par ce paramètre Request-URI.
x.x est la version du protocole utilisé 0.9, 1.0, 1.1...
Pas de CR or LF excepté dans la séquence CRLF finale.

Request-URI = AbsoluteURI | Abs_path CRLF

AbsoluteURI correspond au chemin complet d'accès à la ressource. Cette forme est obligatoire lorsque la requête est faite à un proxy (voir § proxys). Par exemple: on spécifie complètement l'emplacement du document demandé:

GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1 CRLF
CRLF

Absolute_path spécifie la localisation de la ressource sur le serveur. C'est la forme la plus commune de localisation d'une ressource. Dans ce cas, la localisation du serveur sur lequel on trouve le document est donné par le header Host

GET /pub/WWW/TheProject.html HTTP/1.1CRLF
Host: www.w3.org CRLF
CRLF

A noter: Absolute-path ne peut pas être vide.
Remarque: si pour certaines requêtes le document est absent il doit être symbolisé par un "/" qui représente la racine du serveur (the server root).

GET / HTTP/1.1 CRLF
Host: www710.univ-lyon1.fr CRLF
CRLF

Exemple requête/réponse:

GET www.yoyo.com/ HTTP/1.1 CRLF
CRLF

HTTP/1.1 200 OK CRLF
CRLF

La Réponse: Status line

Lorsque le serveur a reçu une requête, il l'interprète et répond par un message HTTP contenant une Réponse (http_Response message).
Format de la status line

Status-Line = HTTP/x.x SP Status_Code SP Reason-Phrase CRLF

Le Status-Code est utilisé par l'automate du protocole et la Reason-Phrase est destiné à l'utilisateur humain.
Le Status-Code est un entier à 3 chiffres qui donne le résultat d'exécution de la requête. La Reason_phrase est sensée donner une courte description du Status-Code. Le premier digit précise la classe des réponses. Cinq classes:
1xx: informative: la requête a été reçue et le processus continue (Informational-Request received, continuing process).
2xx: Success. Succès l'action a été reçue avec succès, comprise et acceptée (The action was successfully received, understood, and accepted).
3xx: Redirection. Redirection nécessaire: D'avantage d'actions doivent être entreprises afin d'accomplir la requête (Further action must be taken in order to complete the request).
4xx: Client Error. Erreur du client: la requête contient une mauvaise syntaxe ou ne peut être accomplie (The request contains bad syntax or cannot be fulfilled).
5xx: Server Error. Erreur du serveur: le serveur a échoué pour accomplir une requête apparemment valide (The server failed to fullfill an apparently valid request).

Voici quelques-uns des Status_codes et Reason_phrases parmi les plus connus:

"100" Continue	"404" Not Found
"101" Switching Protocols	"405" Method Not Allowed
"200" OK	"406" Not Acceptable
"201" : Created	"407" Proxy Authentication Required
"202" Accepted	"408" Request Time-out
"203" Non-Authoritative	"409" Conflict information
"204" No Content	"410" Gone
"205" Reset Content	"411" Length Required
"206" Partial Content	"412" Precondition Failed
300" Multiple Choices	"413" Request Entity Too Large
"301" : Moved Permanently	"414" Request-URI Too Large
"302" : Found	"415" Unsupported Media Type
"303" See Other	"416" Requested range not satisfiable
"304" Not Modified	"417" Expectation Failed
"305" Use Proxy	"500" Internal Server Error
"307" Temporary Redirect	"501" Not Implemented
"400" Bad Request	"502" Bad Gateway
"401" Unauthorized	"503" Service Unavailable
"402" Payment Required	"504" Gateway Time-out
"403" Forbidden	"505" HTTP Version not supported

1.2.4 Les headers

Ils précisent certaines informations transportées par les messages. Le format des message_headers est le suivant:

Message_header = nom_header: SP [valeur] CRLF

En fonction des messages dans lesquels on les utilise, ces Message_headers sont appelés des General_headers, des Request_headers, des Response_headers ou des Entity headers. Nous ne citerons que quelques messages_headers: le détail se trouve bien entendu dans le document RFC.

Format du Message_header

Message _Header = General_header | Request header| Response header| Entity header

Les General_headers

Les General_headers s'appliquent à toutes les requêtes mais pas aux entités transportées. Deux General headers à titre d'exemple :

- **Connection.**

Le header Connection permet à l'expéditeur de spécifier des options spécifiques à la connexion en cours.

Connection: close
Connection: keep-alive

Le client utilise le header connection pour signaler au serveur que la connexion doit être fermée après l'obtention de la réponse à sa requête : valeur close. Les applications clients HTTP/1.0 qui ne supportent pas les connections persistantes doivent inclure the "close" connection option in chaque message.

A partir de Http1.1, le client peut faire plusieurs requêtes au serveur sur une même connexion.
Le header connection: keep-alive est utilisé par le client dans le cas particulier où celui-ci veut réutiliser une connexion qui vient d'être fermée.

- **Date**

Le header date représente la date et l'heure à laquelle le message est crée

Date: Tue, 15 Nov 1994 08:12:31 GMT

Historiquement les applications HTTP sont autorisées à utiliser 3 formats différents:

Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994 ; ANSI C's asctime() format

Les Request_headers

Les Request-headers permettent au client de passer des informations additionnelles sur la requête, sur le client lui-même ou sur le serveur.

Request-header = nom_header : SP [valeur] CRLF

Quelques Request _headers :

Accept: indique ce que le client accepte au niveau des documents. Le client indique son nom, son numéro de version et spécifie le format des documents qu'il accepte.
Accept-Charset : les jeux de caractères acceptés pour réponse
Accept-Encoding; encodages acceptés: les méthodes de compression (gzip image/gif, image/x-xbitmap, image/jpeg ,image/jpeg,)
Accept-Language; les langues naturelles supportées(french).
Host : nomme le serveur Internet (host) et le n° de port de la ressource demandée. Il permet de trouver le bon serveur:car l'adresse IP peut ne pas suffire à le trouver. Cet entête est obligatoire lorsque Request_URI n'est pas un Absolute-path seulement un abs_path.

Host : SP ["nom"_host [":" port]]

L'hôte www.ics.uci.edu est connecté sur le Port 8001.

Host: SP www.ics.uci.edu:8001

If-Modified-Since ; soumet l'exécution de la méthode à la condition que la page ait été modifiée depuis la dernière demande.

If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

If-Unmodified-Since ; soumet l'exécution de la méthode à la condition que la page n'ait pas été modifiée depuis la dernière demande.

If-Unmodified-Since:: Sat, 29 Oct 1994 19:43:31 GMT

User-agent: permet au client de fournir au serveur les caractéristiques de son navigateur et de son système d'exploitation. Par exemple :

GET somedir/page.html HTTP/1.1 CRLF
User-Agent: mozilla/4.0 CRLF
Accept: text/html, image/gif, image/jpeg CRLF
Accept-Language: fr CRLF
CRLF

Authorization est nécessaire dans le cas des pages protégées: identifie le client.

Cookie: utilisé dans une requête retourne au serveur, un cookie que celui-ci a placé au préalable.

Les Response_headers

Ils sont utilisés par le serveur en réponse à une Requête du client. Ils permettent au serveur de passer des informations additionnelles sur la réponse. Ces entêtes donnent des informations sur la ressource et sur les autres accès qui seront faits à la même ressource.
Quelques Response_headers:
Age : donne une évaluation faite par le serveur de la durée de validité d'un page qui vient d'être copiée.
Server: le Serveur contient des informations sur le logiciel utilisé par le serveur d'origine qui traite la requête

Server: Apache/0.8.4
Content-Encoding: méthode gzip ,
Content-language : langue de la page,
Content-Lenght : longueur de la page en octets,
Content-Type: type MIME de la page.
Last-Modified: date et heure de dernière modification (utilisé pour la mise en cache des pages)
Location: indique au client qu'il doit essayer une autre URL: déplacement des pages sur les serveurs, redirection de pages
Accept-Ranges*:* signale au client si le serveur gère les envois de pages par morceaux
Set-Cookies: permet au serveur de placer un cookie chez le client.

Exemple de réponse

HTTP/1.1 200 OK
Date: thu,09 aug 2015 12:00:15GMT
Server: apache/1.3.0
Last-Modified: mon, 18 jul 2015-11-29
Content-Lenght: 6821
Content-Type: text/html
ligne vide
la page de taille 6821 octets.

La gestion des cookies (RFC 2019)

Les cookies constituent un bon exemple d'utilisation des headers.
A la base, le web est un serveur sans mémoire. Il n'y a pas de concept de session ni de mémorisation des états précédents. Cette organisation est très satisfaisante en ce qui concerne les accès aux documents mais actuellement, le succès du web fait **que l'on a besoin de garder la trace des clients.** Citons 3 exemples :

1. Certains sites demandent aux clients de s'enregistrer (et éventuellement de payer) avant de pouvoir travailler avec le serveur).
2. Le commerce électronique permet au client de remplir son panier en flânant sur le site.
3. Les pages d'accueil qui sont personnalisées selon le client. Dans ce cas, il faut que le serveur reconnaisse le client pour lui afficher sa page personnelle.

En première approche, certains ont pensé que l'adresse IP du client pouvait être utilisée par le serveur pour garder la trace du client. Mais très vite, cette solution a été abandonnée (ordinateur partagés par plusieurs client, nomadisme du client…).
C'est alors que Netscape a imaginé une solution: c'est le **cookie** (témoin). Les cookies ont été très critiqués mais ils ont été formalisés dans le RFC 2019.
Un cookie représente un petit fichier (ou une chaîne, string) de 4K octets au plus. Les navigateurs stockent les cookies dans un répertoire réservé à cet effet sur le disque dur du client. Il est possible de désactiver cette option sur le navigateur. Les cookies sont des fichiers ne contenant que de l'information et non des programmes exécutables. S'ils contiennent un virus, il n'y a pas, en principe, de problème puisqu'ils sont traités comme des données. *Toutefois il est toujours possible qu'un pirate utilise un bug du navigateur pour provoquer leur activation. Les navigateurs récents prennent en compte ce problème*

Exemple

Set-Cookie : clientID=497793521; path=/ ;Domain=toms.casino.com; expires=Thu, 01-Apr-2036 21:46:43 GMT CRLF

Set-Cookie: Prod=1-00501;1-00504 ;2-13721, path= / ;Domain=joe-Store.com; Max-Age= 3600 CRLF

Un cookie peut contenir plusieurs champs:

- Le 1° champ *contenu* est toujours de la forme nom=valeur. Il peut représenter n'importe quelle information sur le serveur. C'est le contenu du cookie.
- Le champ *domain* indique l'URL de provenance du cookie Les navigateurs sont censés vérifier que les serveurs ne mentent pas quant à leur domaine. Chaque domaine peut stocker jusqu'à 20 cookies par client.
- Le champ *path* représente le chemin dans la structure du répertoire serveur. / signifie toute l'arborescence.
- Le champ *expiration* représente la validité du cookie. Si aucune date n'est indiquée, le cookie n'est valable que le temps d'ouverture de la session. Il est dit non persistant. Le navigateur le supprime à la fin de sa session. Si le champ est renseigné, le cookie est dit persistant. Le champ Max-Age Max-Age= 3600 attribue une validité d'une heure au cookie. Pour supprimer un cookie du disque dur du client, le serveur l'envoie à nouveau mais avec une validité dépassée.

Le principe du cookie

Lorsqu'un serveur répond, pour la première fois, à la requête d'un client et qu'il désire placer un cookie chez le client, il inclut dans la réponse, le Response Header **Set-Cookie** suivi du fichier cookie. Ceci à condition que le navigateur accepte les cookies. Le navigateur réceptionne la réponse et place le cookie dans le répertoire « **cookies »** du client. (C:\Documents and Settings\user.).

Chaque fois qu'un navigateur envoie une requête à un serveur, il détermine s'il existe un cookie de ce serveur dans son répertoire : c'est-à-dire si le serveur vers lequel il va faire la requête a déjà placé un cookie dans le répertoire. Si c'est le cas, tous les cookies du domaine sont inclus dans la requête (Request Header Cookie). Lorsque le serveur les obtient, il peut les interpréter comme bon lui semble.

Examinons quelques utilisations possibles des cookies donnés en exemple. Le premier cookie a été placé par le serveur *toms-casino.com,* il sert à identifier le client. Ce cookies sera valide jusqu'en 2036 sauf si le client décide de la supprimer. Le second a été placé par le serveur *joes-store.com,* il recense les articles qui sont dans le panier du client. Le client flâne sur le site et regarde ce qu'il peut acheter. Chaque fois, qu'il clique sur un article, le serveur lui renvoie le cookie mis à jour. Lorsque le client clique sur le bouton facturation, le navigateur renvoie le dernier cookie contenant tous les articles sélectionnés. Ce cookie sera valide durant 1 heure.

L'emploi des cookies a été largement détourné. Détournement de cookies par des pirates qui essaient de récupérer des numéros de cartes de crédit stockées par le serveur chez son client. Collecter secrètement des informations sur les habitudes de navigation des clients. *Pour vous en convaincre, visualisez vos cookies*.

Le format MIME RFC 2045 à 2049

La page ou corps du message sont actuellement codés au format **MIME** (Multi Purpose Internet Mail Extensions). Ce format n'existait pas initialement dans Http. En effet, dans les débuts de l'ARPANET, la messagerie électronique ne consistait qu'en messages écrits en anglais et codés au format ASCII (RFC 822). Aujourd'hui, ce format ne convient plus:

- Messages écrits dans des langues différentes contenant des accents (français, allemand),
- Messages écrits dans des alphabets non latins (russes, hébreu..)
- Messages écrits sans alphabets (idéogrammes chinois ou japonais)
- Messages sans texte composés de son et d'images.

Une solution a été proposée comme une modification au format de base, c'est le MIME.
L'idée est de continuer à utiliser le format de base mais de rajouter au corps de message des **extensions** qui définissent le codage pour les messages non ASCII. Ces nouveaux messages peuvent être envoyés comme les anciens mais ils seront traités par les programmes utilisateurs. Le format MIME définit des nouveaux entêtes de messages:

- **MIME-Version :** version du format utilisé. Champ absent = anglais standard
- **Content-Description:** chaîne lisible par le destinataire et décrivant le contenu du message. Cet entête est nécessaire pour indiquer au destinataire du message si cela vaut la peine de décoder le message.
- **Message-ID**: identification unique du message,
- **Content-transfer-encoding :** méthode d'encapsulation du message contenu dans le corps: type de codage le plus simple est le code ASCII.

A titre d'exercice: vous pouvez décrypter les entêtes MIME de vos mails.

MIME reconnaît les types et sous-types de documents:

- Le type Texte inclut les sous-types: plain (texte on formaté, HTML (texte avec Markup HTML) et XML (texte avec markups XML).
- Le type image inclut les sous-types GIF et JPEG.
- Le type audio inclut le basic audio 8bits CM échantillonné à 8000hz et Tone (spécifique).
- Le type application inclut les sous-types Octets stream, Postscript et PDF.

2 GESTION DES CONNEXIONS ENTRE CLIENT HTTP ET SERVEUR HTTP

2.1 Gestion des connexions persistantes.

La méthode habituelle que suit le navigateur du poste client pour communiquer avec un serveur est d'établir une connexion TCP sur le port 80 de la machine serveur.
L'avantage de cette connexion avec TCP est que ni le serveur ni le navigateur n'ont à se préoccuper des messages perdus, reçus en double, trop longs ou d'acquittements: tous ces aspects sont gérés par le protocole TCP.
Le header Connection permet à l'expéditeur (client) de signaler que la connexion sera fermée après l'exécution complète de la requête: c'est à dire la délivrance de la réponse

Attention : Un serveur HTTP1.1 **suppose** que le client HTTP va entretenir une connexion jusqu'à ce que le General_header "connection" (avec close) soit inclus dans une requête qui est considérée (de ce fait) comme étant la dernière. Dans ce cas, si le serveur décide de fermer la connexion et il inclut également, le General_header **connection** :**close** dans la réponse qu'il renvoie au client. Une fois qu'un close a été signalé, le client ne doit plus envoyer de nouvelles requêtes sur cette connexion.

Chapitre 5. Les serveurs basés sur les documents, les serveurs multimedia

1 SYSTEMES DISTRIBUES BASES SUR LES DOCUMENTS (WEB)

Le plus célèbre est le Web. Il est basé sur des fichiers (documents). Un autre est LotusNotes basé sur des bases de données.

1.1 Les processus clients et serveurs Web

Le web utilise 2 sortes de processus: les clients et les serveurs.
Le web client est représenté par un logiciel appelé WEB navigateur (ou browser). Ce navigateur permet au client de rechercher les documents WEB et de les afficher sur son écran. Les hyperliens affichés lui permettent par un simple clic de souris de rechercher les documents suivants.

Coté client

Les browsers Web sont, à la base, des programmes simples. Toutefois, comme ils doivent permettre l'affichage de multiples types et sous-types de documents, ils sont finalement complexes. Pour permettre au navigateur d'évoluer et de prendre en compte de multiples formats de fichiers, il utilise des facilités, les **plug-ins**.
Un plug-in est un petit programme qui peut être chargé et exécuté dynamiquement pour traiter un type de document spécifique.
Quand un navigateur rencontre un type de document pour lequel, il doit utiliser un plug_in, il télécharge ce fichier localement et crée une instance pour l'exécuter. Lorsqu'il reste longtemps inutilisé, le plug in est supprimé du navigateur.
Un autre processus client side qui est souvent utilisé est le Web proxy. Actuellement, ces proxys sont populaires parce qu'ils fournissent un cache pour les pages partageables entre plusieurs navigateurs.

Coté serveur

Un serveur web est un programme qui réceptionne les requêtes http et retourne les réponses au client. Le noyau (core) http réceptionne les requêtes entrantes http. Le serveur est organisé selon une série de modules qui chacun prennent en charge une partie du traitement de la requête. Le noyau construit un Request Record avec les informations contenues dans la requête et il le passe aux différents modules, organisés en pipeline. Chaque module a un rôle dans le traitement de la requête :

1. Résolution de la référence un document. On utilise l'URL du document ou les informations nécessaires ua lancement du programme CGI,
2. Authentification du client: vérification de l'identité du client sans contrôle d'accès.
3. contrôle d'accès client: on contrôle que le client a les droits d'accès suffisants (bon groupe d'utilisateurs),
4. accès du client: est-ce que le client a le droit de faire cette requête?
5. détermination du type MIME de la réponse,
6. lancement des traitements supplémentaires,
7. transmission de la réponse,
8. Le dernier module passe le document réponse au noyau.

Les différentes phases du serveur sont définies au moment du paramétrage du serveur.
Prenons l'exemple du serveur Apache qui est le serveur dominant des environnements Unix (utilisation possible sous Windows). Le serveur Apache est constitué de modules qui sont contrôlés par un seul module noyau (Apache core). Le serveur Apache est très configurable.

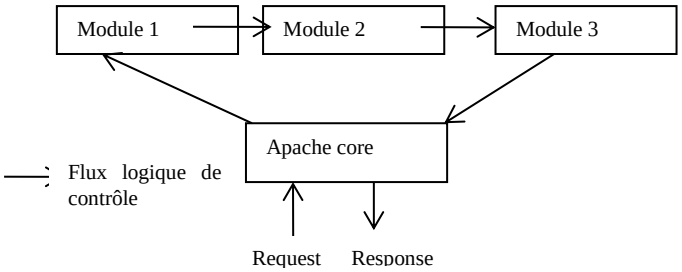


Figure 9 Organisation générale d'un serveur Apache

1.2 Les serveurs dupliqués (cluster server)

Un serveur web peut être très vite saturé. Une solution pratique consiste à installer le serveur sur un ensemble de stations de travail pilotées par une machine d'entrée (front-end). Le problème vient souvent de ce que la machine frontale (front-end) joue le rôle de goulot. La différence entre la machine frontale et les autres se situe au niveau du protocole de transport.
En général, un client qui envoie une requête http est connecté au serveur par une connexion TCP. Un switch fonctionnant au niveau transport passe les données de la connexion à une des machines possédant le serveur web. Le switch ne tient pas compte du contenu de la requête ni de la réponse. Il se base uniquement sur la charge des serveurs.

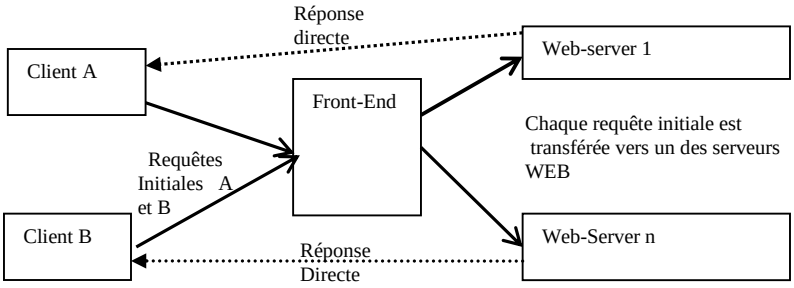


Figure 10 Transfert de connexion entrante

Une autre possibilité plus intéressante est de déployer une **content aware request distribution.** Dans ce cas, la machine Front-end analyse la requête Http initiale entrante et décide ensuite vers quel serveur web (1 à n serveurs) elle l'achemine.
Un mécanisme a été mis en place pour gérer les connexions TCP avec le client. Premièrement, quand la requête arrive, la machine front-end doit choisir le serveur qui va la traiter. Deuxièmement, la machine front end transfère la requête initiale à ce serveur ainsi que toutes les demandes associées à cette connexion.

1.3 Amélioration des performances de http.

La popularité du WEB a provoqué des problèmes d'attentes et d'embouteillages. Aussi des techniques ont été développées pour améliorer les performances. Nous en citons deux:

- La réplication de serveurs (mirroring). Le client se connecte sur le site central puis on lui demande de choisir un site plus proche qui sera son serveur. (Ex: site par pays, ou un site par contient.
- La mise en cache des pages (§ proxy). Un moyen simple d'améliorer les performances est de conserver en mémoire les pages qui ont été demandées au cas où elles seraient à

nouveau réclamées par le client. Cette technique est très efficace dans le cas des pages très consultées (FAI, yahoo.com). Le processus d'accumulation des pages est appelé **mise en cache**.

La procédure habituelle est d'avoir chez le client un mandataire appelé **proxy** qui se charge de la gestion des pages du cache. **Un proxy est un agent actif**, recevant les requêtes destinées à un serveur recomposant tout ou partie du message, et réémettant la requête transformée à destination du serveur si cela est nécessaire.

Pour employer cette technique, un navigateur **doit être configuré pour envoyer toutes les requêtes au proxy et non au serveur hébergeant la page**. La figure 10 résume les 3 cas possibles.

- (1) Si le proxy détient la page il va s'interroger sur sa validité. Si c'est le cas, il l'envoie
- (2) si la page n'est pas valide ou bien si le serveur ne connait pas la date de validité, il pose la question au serveur. Le Request_header **if modified_since** permet au proxy d'envoyer au serveur une requête de page qui ne sera exécutée que dans le cas où la condition est vérifiée. Il peut aussi obtenir la date de dernière modification d'une page avec le Request_header HEAD. Si la page a été modifiée (3) , le serveur l'envoie au proxy qui l'ajuste au cache en prévision des autres demandes puis la transmet au client.

Des mécanismes permettent de signaler que certaines pages actives (php) ne doivent pas être stockées dans des caches.

Plusieurs questions importantes

1. **Combien de temps doit-on garder les pages?** Le but est de ne pas délivrer des pages périmées au client: la page est là mais la page d'origine a été modifiée. D'où la question suivante
2. **Quelle est la validité des données délivrées?** On peut se servir d'heuristiques: par exemple la date de dernière modification de la page: si elle a été modifiée il y a 1 mois, on peut la garder 1 mois; si elle a été modifiée il y a une minute, on ne la gardera pas.
3. **A qui s'adresse cette technique?** Sur un LAN d'entreprise, le proxy est installé sur une machine partagée par toutes les machines du réseau. Les Fournisseurs d'Accès Internet (FAI) installent aussi des proxy sur leurs machines pour accélérer la distribution des pages les plus demandées par leurs clients. Plusieurs caches fonctionnent en même temps sur une connexion.

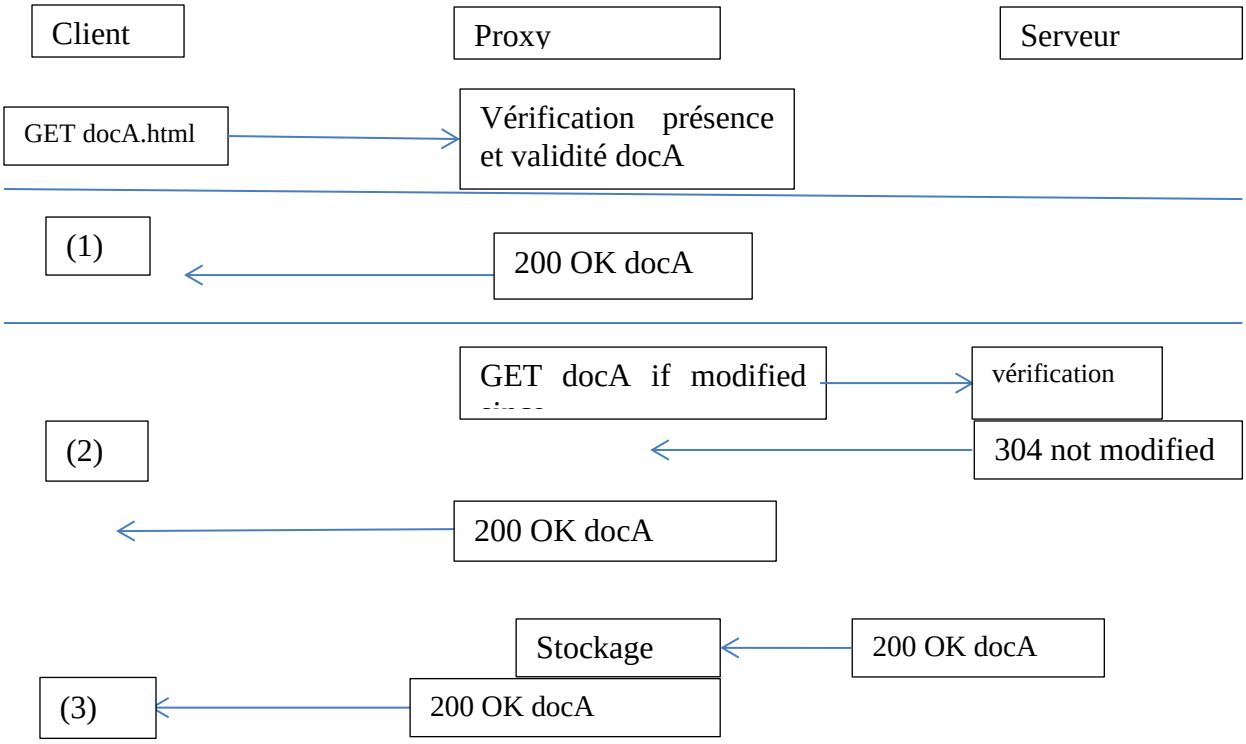


Figure 11 Exemple d'une requête de page traité par proxy

Les principales évolutions du WEB.
A partir du web initial qui peut se résumer au mécanisme de requête d'un document sur le serveur et délivrance du document en guise de réponse, de nombreuses évolutions sont apparues:

- L'interface CGI (Common Gateway Interface).
- Exécution d'un script coté serveur (Server-side script)
- Utilisation des applets et servlets
- Le transfert de documents multimédia (serveur multimédia)

1.3.1 L'interface CGI (Common Gateway Interface).

A partir du serveur WEB, l'évolution principale vise à permettre une interaction de l'utilisateur par le moyen du CGI (Common Gateway Interface).
CGI est un moyen standard qui permet à un serveur WEB d'exécuter un programme en utilisant en entrée des données saisies par le client.
Généralement ces données d'entrée viennent d'un formulaire HTML. Sont spécifiés dans la requête: le programme à exécuter ainsi que la valeur des données saisies par le client
Quand le serveur reçoit la requête, il lance le programme spécifié en lui passant les paramètres. Ces programmes peuvent être très sophistiqués et utiliser une base de données. (Voir figure 11)
Lorsque les données ont été traitées, le programme peut générer un document HTML qui sera passé par le serveur au client sous forme d'une réponse. Un aspect intéressant vient de ce que tout se passe pour le client comme "s'il avait été chercher ce document" sur le serveur. Le principe du WEB reste le même.

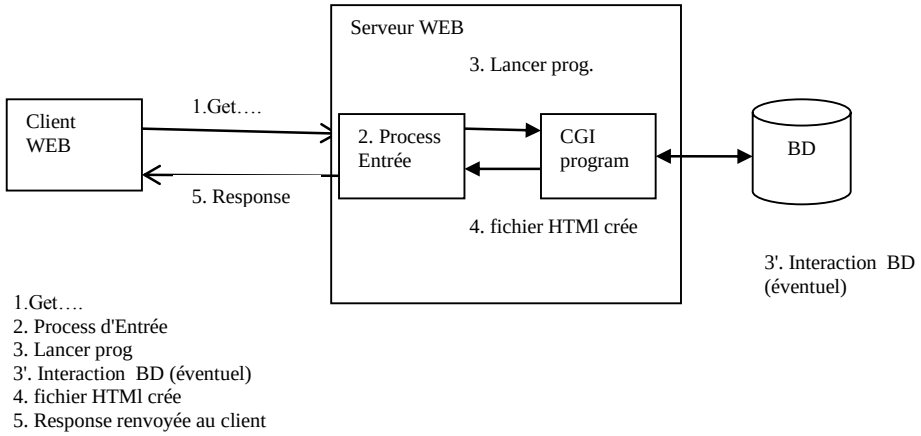


Figure 12 Le principe des applications CGI

1.3.2 Le server-side script

Une autre amélioration importante est que le serveur peut **exécuter un document** qu'il vient de rechercher avant de le passer au client. Le document recherché contient un script (**server-side script**). Le résultat d'exécution est renvoyé au client avec le reste du document recherché. Toutefois, le script lui-même n'est pas envoyé. En d'autres termes, le server-side script remplace le code exécutable dans le document par le résultat d'exécution.
Le server-side script est reconnu grâce à une étiquette du type <SERVER >. N'appartenant pas à HTML, l'étiquette dépend des serveurs.
Le serveur renvoie le résultat d'exécution du script (à la place de celui-ci) compris entre les tags <SERVER>

1.3.3 Les applets et servlets

Il est aussi possible de passer des programmes pré compilés à un client sous la forme d'applets. En général, un **applet** est une petite application indépendante qui est envoyée vers le client sous forme de fichier et exécutée dans l'espace adresse du client. La forme la plus courante d'applets se présente sous forme de code Java pré compilé prêt à être interprété dans la machine du client.
Par exemple la commande suivante inclut un applet java dans un document HTML :
<OBJECT codetype=application /java"classid= "java:welcome.class">
Les **applets** sont exécutés sur la partie client, les **servlets** sont eux exécutés sur la partie serveur.
Comme un applet, un servlet est un programme précompilé, exécuté dans l'espace adresse du serveur. La plupart des servlets sont codés en Java mais il n'y a pas de restriction pour les coder avec d'autres langages.
Lorsqu'un serveur reçoit une requête http adressée à un servlet, il lance la méthode correspondante au servlet. Celui-ci exécute la requête et retourne le résultat d'exécution sous forme d'un document HTML.
La différence importante entre un script CGI et un servlet vient de ce que le servlet est directement exécuté par le serveur alors que le programme CGI fait appel à un processus séparé
Nous pouvons donner une architecture fonctionnelle d'un serveur Web dans la figure suivante.

Lorsque le serveur reçoit la requête http du client (1), il peut faire 3 tâches selon le contenu de la requête:

1. Il peut tout simplement rechercher le document dans la base de données (2a, 3b)

2. Il peut exécuter le servlet. Celui-ci durant l'exécution pourra utiliser la BD, puis créer la réponse pour le serveur (2b,3b,4c)
3. Il peut exécuter un programme CGI. Celui-ci peur accéder à la BD puis créer le document réponse (2c, 3c, 4c).

Ensuite le post processeur crée la réponse à la requête (5), le navigateur peut exécuter les éventuels **client_side script** et afficher le document à l'utilisateur.

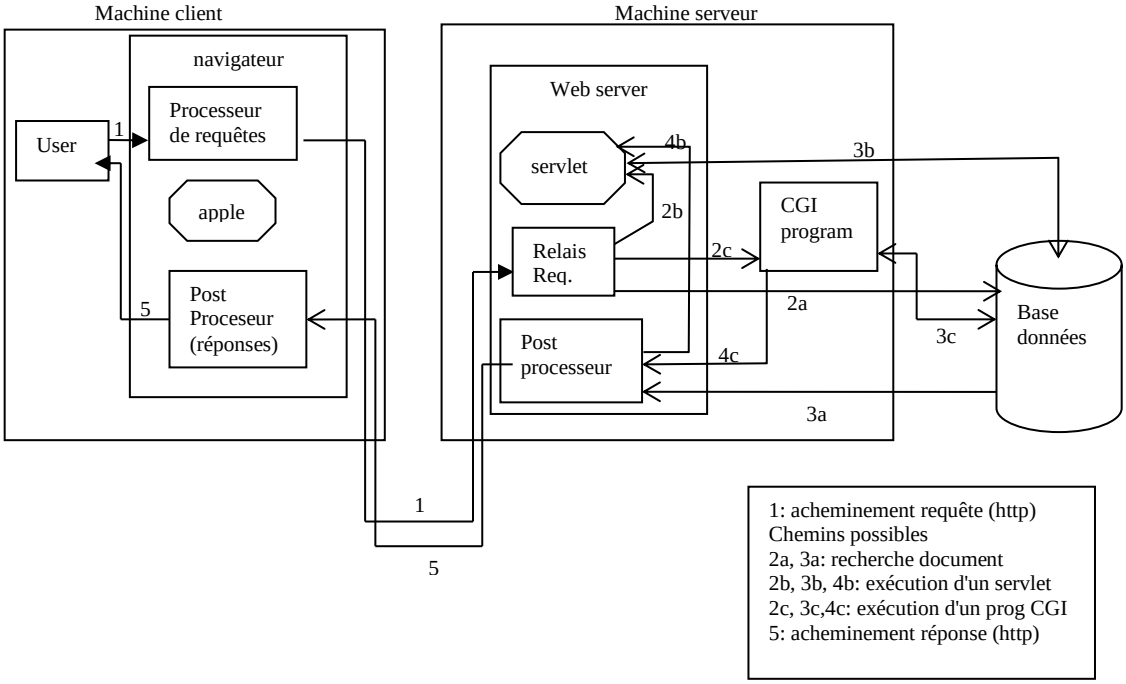
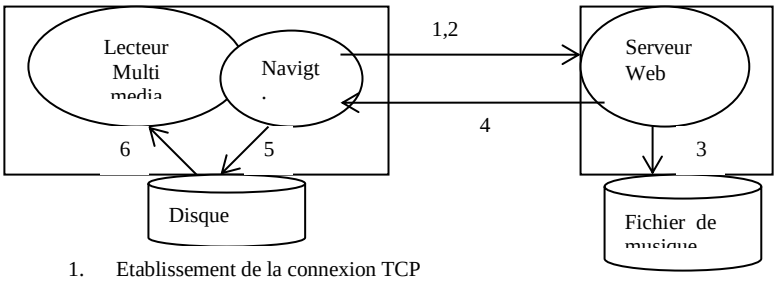


Figure 13 Architecture fonctionnelle d'un serveur WEB

2 LES APPLICATIONS MULTIMEDIA (AUDIO) DE L'INTERNET

2.1 Le principe de base

L'Internet regorge de sites Web consacrés à la musique (streaming audio). Le principe est le suivant: un serveur WEB contient les fichiers de musique, Le client équipé d'un navigateur et d'un lecteur multimedia tel que Real Player, Windows Media Player ou WinamP se connecte pour télécharger pour écouter un document musical. Le processus démarre lorsque l'utilisateur clique sur le titre du morceau de musique. Le navigateur fait un GET du document musical sur le serveur. Lorsqu'il récupère ce document, il lit le type MIME du fichier. Généralement c'est un fichier **audio/mp3**.



1. Etablissement de la connexion TCP
2. Envoi de la requête http GET
3. Récupération du fichier audio (MP3) sur le disque
4. Envoi du fichier au client web (réponse du Get
5. Ecriture du fichier sur disque par le navigateur
6. Récupération et lecture du fichier par le lecteur

Figure 14 Le principe du web audio

Pour communiquer avec l'application auxiliaire (le lecteur multimedia), le navigateur écrit le fichier dans un fichier de travail (scratch file) (5). Il invoque ensuite le lecteur multimédia et lui passe le nom du fichier scratch. A l'étape 6, le logiciel lit le fichier bloc par bloc et commence à jouer de la musique.
Le problème vient de ce que l'on ne peut commencer à jouer de la musique que lorsque le morceau est complètement téléchargé. Pour une chanson, la taille moyenne du fichier MP3 est d'environ 4Mo (32 Megabits).
Le transfert peut prendre 1 à 2 mn (2Mbits/s).

2.2 Les serveurs audio, streaming

Pour résoudre ce problème sans changer de mode de fonctionnement du navigateur, les sites web de musique ont adopté une approche commune.
Le fichier lié au document demandé et retourné dans la réponse du GET n'est plus le fichier lui-même mais un petit fichier contenant l'URL du document (la localisation). C'est un **métafichier.**

Par exemple: le contenu est le suivant:
rtsp://joes-audio-server/song-0025.mp3

De la même façon que précédemment lors de l'étape 5, le navigateur écrit le nom sur le disque, lance le logiciel multimedia et lui passe le nom du fichier comme dans le premier cas.
Le lecteur lit le fichier, voit qu'il contient une URL.
Il contacte alors directement le serveur *joes-audi-server* qui peut ne pas être un serveur WEB (FTP, http).

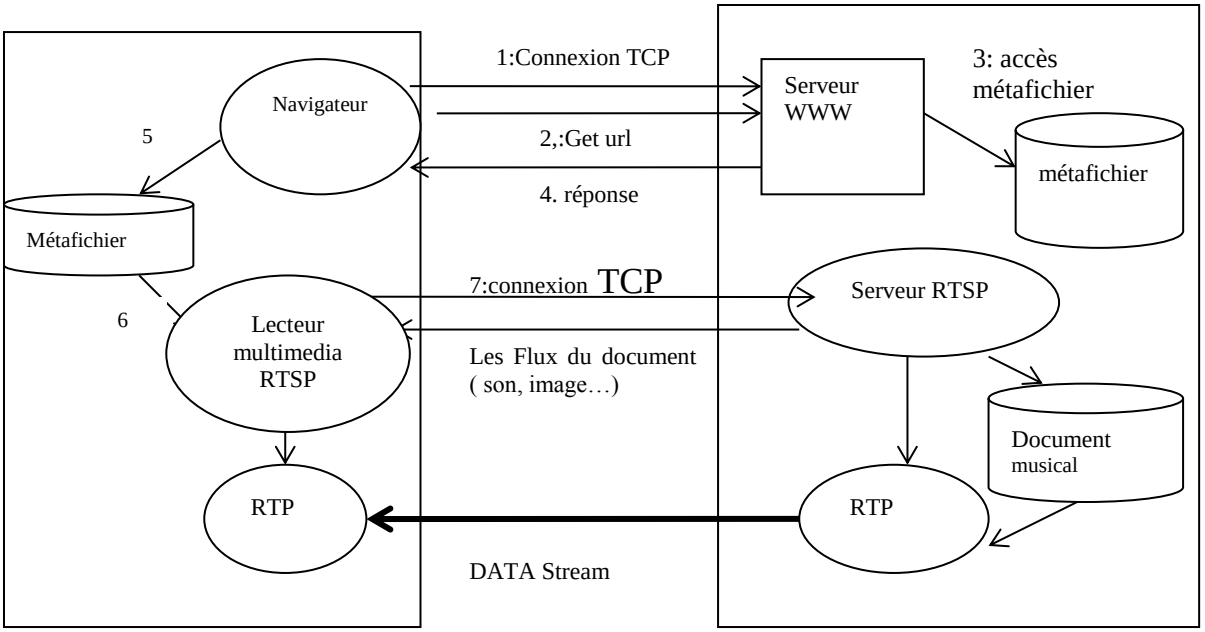


Figure 15 Le cas le plus courant des web audio

Dans l'exemple, c'est un serveur de contenu spécialisé qui emploie le protocole RSTP (Real Time Streaming Protocol). (RFC 2326)
Le logiciel multimedia gère donc le transfert de musique en temps réel et plus généralement l'application multimedia composée de plusieurs flux: son, video, texte…. Ce transfert se fait rarement en utilisant TCP, car une retransmission suite à une erreur risquerait d'introduire un délai inacceptable dans la lecture. En pratique, on utilise soit le protocole Real-time Transport Protocol (RTP (RFC 1889), soit le protocole Real Time Streaming Protocol (RTSP - RFC 2326).
Le protocole RTP permet de transmettre des documents audio. Il se trouve dans l'espace utilisateur avec l'application Sa position est étonnante puisque *RTP est un protocole de transport implanté dans la couche application* et utilisant UDP. RTP multiplexe les flux et les encode en paquets RTP qu'il envoie ensuite vers le client à l'aide de plusieurs sockets UDP. Les paquets RTP sont numérotés par l'émetteur et contrôlés par le récepteur. En cas de perte de paquets RTP, le destinataire pallie au défaut par interpolation temporelle (t1, t2 abs ,t3). Un système de retransmission ne serait pas intéressant en temps réel. … RTP peut envoyer les paquets soit vers un destinataire unique (unicast) soit vers plusieurs destinataires (multicasting).
Le protocole RTSP permet de transmettre des applications multimedia composées de plusieurs flux: son, video, texte. Il utilise 2 liaisons au moins : une connexion TCP et une ou plusieurs autres avec UDP. De manière pratique, le client contacte le serveur de streaming grâce à la connexion TCP. En réponse à cette requête, le serveur retourne une description de la session de streaming qu'il va ouvrir : une session de streaming est composée d'un ou plusieurs flux (stream), par exemple audio ou vidéo. Le serveur informe le client du nombre de flux. Il donne aussi des informations décrivant les flux comme le type du média et le codec de compression. Les flux sont quant à eux diffusés séparément via le protocole RTP qui utilise UDP.

Conclusion

Ce document est un support de cours. Les exemples et TD permettront d'éclaircir certains points que je n'ai volontairement pas traités. Enfin deux projets vous permettront de mettre en pratique ces notions. Afin d'améliorer ce document, n'hésitez pas à me faire parvenir vos remarques.

Bibliographie

1. Ana Cavalli, Ingénierie des Protocoles et Qualité de Services, Hermès Sciences, 2001
2. Douglas Comer. TCP/IP- Architecture, protocoles et applications, 5° édition, Pearson Education.
3. Estelle Information URL http://www.estelle.org/
4. Forum d'entraide Linux professionnel, URL
5. <http://coredump.developpez.com/nfs/principes/>
6. P. Rolin, Réseaux locaux, Normes et protocoles 5° Edition, Hermès.
7. RFC Documents accessible à l'URL http://www.frameip.com/rfc/
8. Andrew S. Tanenbaum, Réseaux, 4° édition, Pearson Education.
9. Andrew S. Tanenbaum, Maarteen van Steen, Distributed systems principles and paradigms, International Edition, Practice Hall, 2002.
10. Toutain L., Réseaux locaux et Internet, 2° Edition, Hermès Science.