2

La couche Applications

Les applications de réseau constituent *la raison d'être* de tout réseau orienté vers l'informatique communicante. S'il n'existait pas d'applications utiles, il n'y aurait pas besoin de mettre au point des protocoles de réseau pour les transporter. Or, justement, ces trente dernières années ont vu naître une profusion d'applications de réseau, toutes plus ingénieuses les unes que les autres, parmi lesquelles on recense toute la gamme d'applications textuelles développées dans les années 1980, telles que les applications de messagerie électronique, l'accès à des terminaux distants, le transfert de fichiers, les groupes de discussion, le bavardage en ligne (ou chat), ainsi que l'application phare des années 1990 : le Web. On remarque également le développement de toute une gamme d'applications multimédias, telles que la vidéo en streaming, les programmes radiophoniques, la téléphonie sur internet et les visioconférences sur internet. À ceci, il convient d'ajouter les deux applications incontournables de la fin du XX^{e} siècle : la messagerie instantanée et le partage de fichiers musicaux MP3 de poste-à-poste.

Dans ce chapitre, nous nous attacherons à l'étude des aspects conceptuels et pratiques liés à la mise en œuvre des différentes applications de réseau. Nous commencerons par la définition des concepts-clés associés à la couche Applications, à savoir, la description des protocoles de cette couche, les notions de client et de serveur, de processus, d'interfaces de connexion et d'interfaces de la couche Transport. Nous examinerons ensuite dans le détail de différents protocoles d'applications : HTTP (Web), SMTP et POP3 (pour le courrier électronique), FTP (pour le transfert de fichiers) et DNS (pour la conversion des noms de serveurs en adresses IP).

Puis, nous nous pencherons sur les aspects liés au développement des applications de réseau, sur TCP et sur UDP. Nous examinerons l'interface API et nous essaierons de programmer quelques applications client-serveur simplifiées en Java. Nous verrons notamment comment créer un serveur Web à l'aide de ce langage.

Nous conclurons ce chapitre par une analyse relativement approfondie des éléments constitutifs d'un système de distribution de contenu : serveurs cache, serveurs proxy, réseaux de distribution de contenu (CDN) et applications de partage de fichiers de

poste-à-poste. Dans ce contexte, nous nous attacherons tout particulièrement à comprendre le fonctionnement des réseaux de cœur d'infrastructure P2P, c'est-à-dire des réseaux virtuels en P2P, le protocole de partage de fichiers P2P résidant au sein de la couche Applications et reposant sur l'internet.

2.1 Principes des protocoles de la couche Applications

Bien qu'il existe une multitude d'applications de réseau très diverses et se caractérisant par l'interaction d'éléments disparates, presque toutes partent d'un noyau logiciel. Au chapitre 1, nous avons vu que le logiciel d'une application se répartit sur plusieurs systèmes (c'est-à-dire sur différents terminaux, qu'il s'agisse d'ordinateurs ordinaires ou de serveurs). L'application Web, par exemple, est le fruit de deux logiciels communiquant l'un avec l'autre : le navigateur du poste de l'utilisateur (PC, Macintosh, PDA, etc.) et le programme correspondant au sein d'un serveur Web. Telnet repose également sur deux logiciels exploités sur deux serveurs différents : le programme Telnet du serveur local et le programme Telnet du serveur distant. Une application de visioconférence fait appel à autant de logiciels qu'il y a de participants.

Dans le jargon des systèmes d'exploitation, les entités interlocutrices ne sont pas réellement des éléments logiciels (des programmes), mais plutôt des **processus** qui communiquent. Un processus peut être considéré comme un programme travaillant au sein d'un système terminal. Lorsque différents processus de communication sont exploités sur un même terminal, ils interagissent au moyen d'un mode de communication interprocessus, dont les règles sont définies par le système d'exploitation du terminal. Dans cet ouvrage, nous ne nous intéresserons pas aux échanges entre divers processus exploités sur un même poste, mais à la manière dont ceux-ci interagissent d'un terminal à un autre (ceux-ci pouvant être dotés de systèmes d'exploitation différents). Les processus qui se déroulent sur deux terminaux différents communiquent entre eux par l'envoi de **messages** à travers le réseau. Un processus d'expédition génère et envoie des messages sur le réseau, tandis que le processus de réception reçoit ces messages et y répond éventuellement. Les applications de réseau disposent de protocoles de couche d'application définissant le format et l'ordre des messages échangés entre les processus, ainsi que les étapes à respecter lors de la transmission ou de la réception d'un message. À la figure 2.1, différents processus communiquent au moyen de la couche Applications de la pile de protocoles à cinq couches.

La couche Applications est tout indiquée pour débuter l'étude des protocoles de réseau. En effet, c'est la couche la plus facile à cerner. Nous sommes tous familiarisés avec la plupart des applications ayant recours aux protocoles que nous passons en revue. Ce chapitre va nous permettre de saisir la notion de protocole dans toute sa dimension et nous confrontera à plusieurs thèmes récurrents de cet ouvrage, qui ont trait aux protocoles de la couche Transport, de la couche Réseau et de la couche Liaison de données.

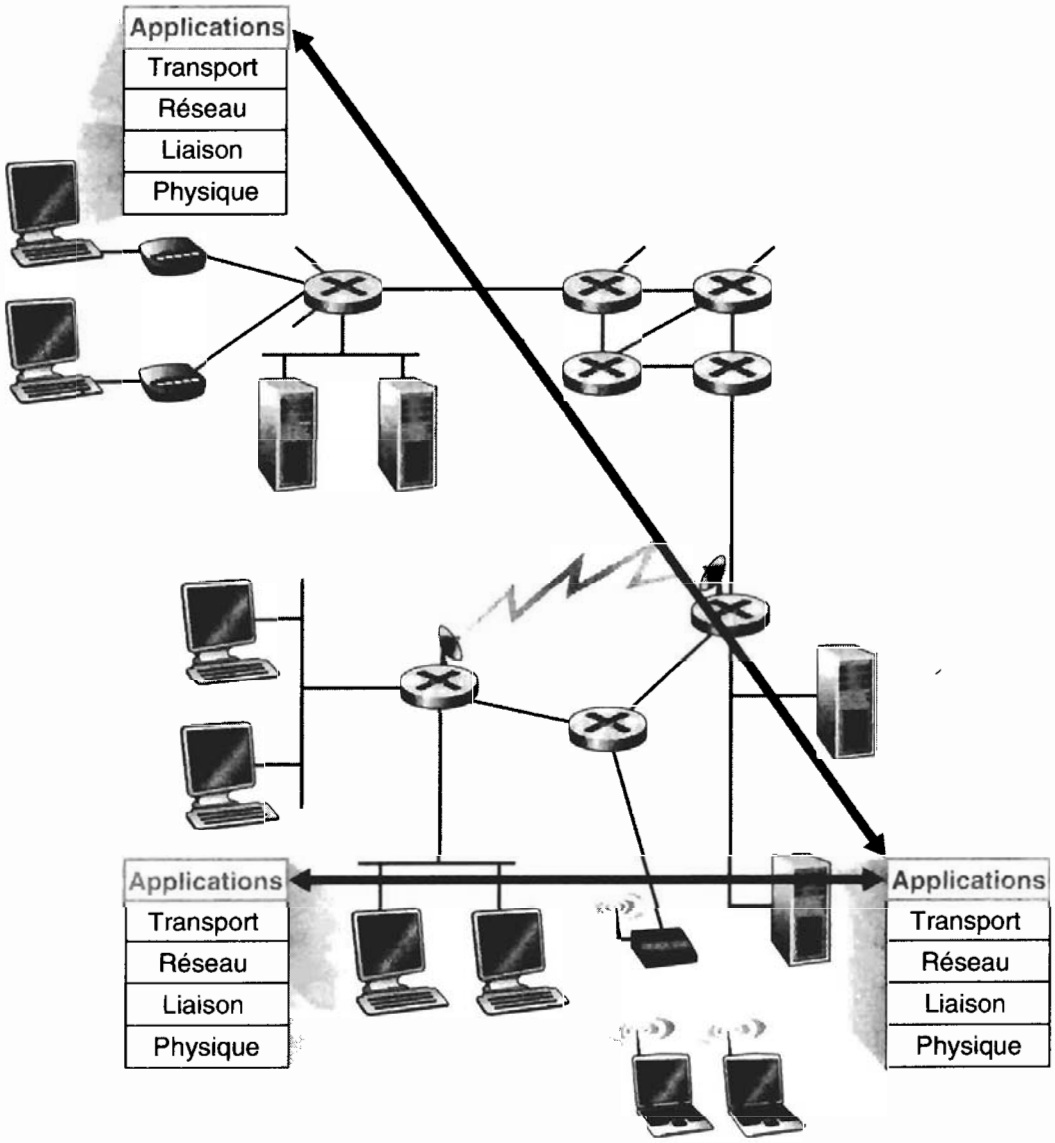

Figure 2.1 • Les fonctions de communication d'une application de réseau prennent place dans la couche Applications.

2.1.1 Protocoles de la couche Applications

Il est important de bien faire la distinction entre la notion d'**application de réseau** et celle de **protocoles de la couche Applications**, à savoir que ces derniers ne sont que des éléments (quel que soit leur rôle) d'une application de réseau. Illustrons donc ceci au moyen de quelques exemples. Le Web est une application de réseau qui permet aux utilisateurs d'obtenir sur demande des « documents » hébergés sur des serveurs Web. L'application Web est constituée de divers composants, parmi lesquels figurent les normes définissant le format des documents échangés (HTML), des navigateurs Web (notamment Netscape Navigator et Microsoft Internet Explorer), des serveurs Web (par exemple, les serveurs Apache, Microsoft ou Netscape) et un protocole de couche d'application. Ce dernier protocole, utilisé pour le Web (HTTP, Hypertext Transfer Protocol [RFC 2616]), définit le format et la séquence des messages échangés entre navigateur et serveur Web. Ainsi, HTTP, protocole de couche Applications pour

le Web n'est qu'un élément de l'application Web. Prenons, comme autre exemple, les applications de messagerie électronique en ligne. Celles-ci aussi sont composées de nombreux éléments parmi lesquels des serveurs de messagerie qui hébergent les boîtes aux lettres des utilisateurs, des programmes permettant de lire et de rédiger des messages électroniques, une norme définissant la structure de ces messages et un protocole de couche Applications déterminant la manière dont ceux-ci sont échangés entre les différents serveurs, comment ils passent des serveurs aux logiciels de lecture et de composition de messages, et comment le contenu de certaines parties du message (notamment l'en-tête) doit être interprété. Le principal protocole de couche Applications actuellement employé pour l'envoi de messages électroniques est SMTP (*Simple Mail Transfer Protocol* [RFC 2821]), qui n'est ainsi lui-même qu'une partie d'un système de messagerie électronique.

Un protocole de couche Applications définit donc la façon dont les processus d'une même application mise en œuvre sur des systèmes d'exploitation différents échangent mutuellement des messages. Plus précisément, un protocole de couche Applications définit :

- le type de messages échangés, par exemple des messages de demande et de réponse ;
- la syntaxe adoptée par les différents types de messages, soit les différents champs qu'il contient et leur délimitation ;
- la sémantique des différents champs, c'est-à-dire le sens des informations qu'ils renferment ;
- les règles utilisées pour déterminer quand et comment un processus doit envoyer ou répondre à un message.

Certains protocoles de la couche Applications sont définis par des RFC. Ils appartiennent donc au domaine public. C'est par exemple le cas de HTTP. Tout navigateur respectant les normes de ce RFC sera en mesure de solliciter et d'obtenir des pages Web de n'importe quel serveur Web conçu conformément à ces normes. Mais de nombreux protocoles de la couche Applications relèvent du domaine privé et sont inaccessibles au domaine public. Actuellement, une grande partie des systèmes de téléphonie sur l'internet, par exemple, utilise des protocoles propriétaires.

Pôles client et serveur d'une application

Comme indiqué à la figure 2.2, une application de réseau comprend généralement deux parties, ou « pôles », en l'occurrence un **pôle client** et un **pôle serveur**. D'une manière générale, les communications s'établissent entre le pôle client d'un système terminal et le pôle serveur d'un autre système terminal. Dans le cas du Web, par exemple, le logiciel navigateur met en œuvre le pôle client de HTTP tandis que le pôle serveur est mis en œuvre par un serveur Web de HTTP. Dans le cas d'une application de messagerie électronique, le serveur source du message met en œuvre le pôle client de SMTP et le serveur destinataire le pôle serveur de SMTP.

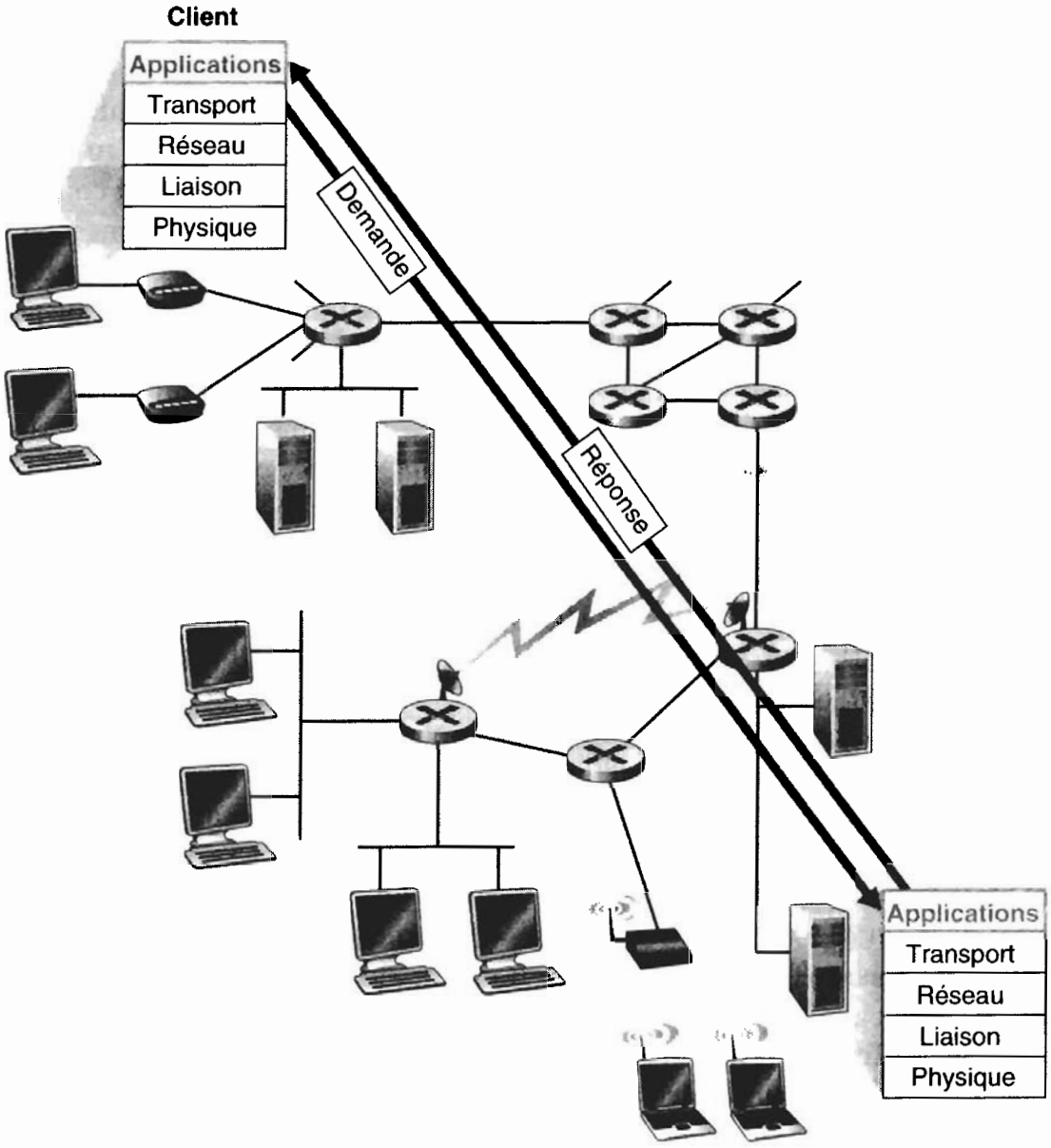

Figure 2.2 • Interaction client-serveur.

De nombreuses applications voient leurs pôles client et serveur mis en œuvre par le même serveur. Considérez par exemple une session Telnet entre deux systèmes terminaux A et B. (Telnet est une application de connexion à distance très répandue.) Si c'est le serveur A qui ouvre la session (en d'autres termes si un utilisateur tente de se connecter à B à partir de A), alors le serveur A gère le pôle client de l'application et le serveur B gère le pôle serveur. Si, en revanche, c'est le serveur B qui ouvre une session Telnet, alors, le serveur B gère le pôle client de l'application. Prenons l'exemple de FTP, application employée pour transférer des fichiers entre deux serveurs. Lorsqu'une session FTP est établie entre deux serveurs, chacun d'eux peut jouer le rôle de source ou de destination de fichiers. Toutefois, comme c'est le cas de la plupart des applications de réseau, c'est le serveur qui a ouvert la session qui est considéré comme le client.

Communication des processus à travers un réseau

Comme nous l'avons vu ci-dessus, beaucoup d'applications de réseau impliquent une communication entre deux processus engagés sur deux postes différents. Les deux processus interagissent en s'échangeant des messages, qu'ils font passer au travers de leurs interfaces de connexion (sockets). Une interface de connexion agit comme une porte : lorsqu'un processus souhaite envoyer un message à un autre processus opérant sur un serveur différent, il le fait passer par sa porte (interface). Ce processus d'envoi part bien entendu du principe qu'il existe une infrastructure de transport de l'autre côté de cette porte, prête à prendre en charge le message et à l'emmener jusqu'à la porte du processus destinataire *via* l'internet. Lorsque le message atteint le serveur de destination, il passe la porte (interface de connexion) du processus, qui prend le relais pour exécuter l'action désirée.

La figure 2.3 illustre les échanges entre deux processus communiquant *via* l'internet. Le protocole de transport choisi est TCP, mais cela aurait tout aussi bien pu être UDP. Une interface de connexion joue un rôle de liaison entre la couche Applications et la couche Transport d'un même serveur. On utilise également l'expression **interface API** (*Application Programmers' Interface*) entre l'application et le réseau, dans la mesure où le **socket** est l'interface de programmation servant à la mise en place des applications de réseau au sein de l'internet. Les développeurs d'applications ont un contrôle absolu sur le pôle de l'interface de connexion associé à la couche Applications, mais ils en ont très peu sur le pôle associé à cette couche, à savoir (1) le choix du protocole de transport et (2), dans certains cas, la possibilité de définir quelques paramètres de la couche Transport, tels que les tailles maximales des tampons et des segments. Une fois qu'un développeur a choisi un protocole de transport pour son application (en admettant qu'il ait le choix), celle-ci se met en place à l'aide des services de couche Transport fournis par ce protocole. Nous examinerons les interfaces de connexion en détail dans les sections 2.6 et 2.7.

Processus d'adressage

Pour qu'un processus travaillant sur un serveur puisse envoyer un message à un processus opérant sur un autre serveur, le premier doit pouvoir identifier le second, ce qui fait appel à deux types d'informations : (1) le nom ou l'adresse du serveur et (2) un identifiant spécifiant la nature du processus de réception chez le destinataire.

Voyons tout d'abord les adresses des serveurs. Dans les applications internet, le serveur de destination est identifié au moyen de son **adresse IP**, qui sera détaillée au chapitre 4. Pour l'instant, il suffit de savoir qu'une adresse IP, longue de 32 bits, identifie uniquement le serveur (elle n'identifie en réalité que l'interface réseau connectant ce serveur au réseau internet). Étant donné que l'adresse IP de tout poste connecté à l'internet public doit être absolument unique, l'assignation des adresses IP doit répondre à une gestion rigoureuse, comme indiqué au chapitre 4.

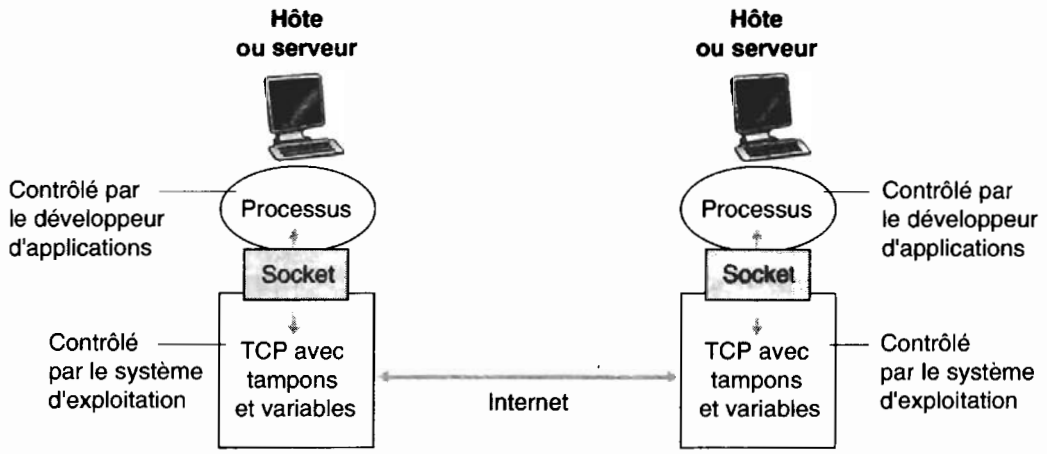

Figure 2.3 • Processus d'applications, interfaces de connexion et protocoles de transport sous-jacents.

L'application expéditrice doit non seulement connaître l'adresse du destinataire d'un message, mais aussi fournir une information permettant à ce dernier, une fois arrivé, de conduire le message vers le processus approprié. Dans le cadre de l'internet, cette information est un **numéro d'accès** ou de porte (*port number*). Les protocoles de la couche Applications les plus courants ont des numéros d'accès spécifiques. Par exemple, le numéro d'accès d'un processus de serveur Web (qui utilise le protocole HTTP) est systématiquement 80. Celui d'un processus de serveur de messagerie électronique (protocole SMTP) est 25. Une liste des numéros d'accès des protocoles les plus courants figure dans le RFC 1700 (un peu ancien) ou en ligne à l'adresse suivante : http://www.iana.org [RFC 3232]. Lorsqu'un développeur met au point une nouvelle application de réseau, celle-ci doit obligatoirement disposer d'un numéro d'accès unique. Ce sujet est évoqué au chapitre 3.

Agents utilisateurs

Avant de décrire les protocoles de la couche Applications, il est judicieux de définir la notion d'agent utilisateur, qui ne cesse de revenir au cours des prochaines sections. Ce nom s'applique à l'interface reliant l'utilisateur à l'application. Dans le cas du Web, par exemple, l'agent utilisateur n'est autre que les navigateurs Netscape ou Microsoft Internet Explorer. Ceux-ci permettent de visionner des pages Web, de surfer, de remplir des formulaires en ligne, d'interagir avec des applettes Java, etc. Le navigateur est également responsable de la mise en œuvre du pôle client du protocole HTTP. Le navigateur est ainsi un processus envoyant et recevant des messages au travers d'une interface de connexion, tout en fournissant une interface à l'utilisateur. Pour prendre un autre exemple, considérons l'application de messagerie électronique. Dans ce cas, l'agent utilisateur est un « éditeur de messages » qui permet à l'utilisateur de composer et de lire des messages. Les éditeurs modernes (tels que Microsoft Outlook, Eudora, AOL et Netscape Communicator) proposent des interfaces graphiques facilitant l'utilisation. Comme évoqué à la section 2.4, ces logiciels ont fréquemment recours à deux protocoles : SMTP pour l'envoi de messages et un protocole d'accès, tel que POP3 ou IMAP, pour la réception.

2.1.2 Services nécessaires à une application

Dans le jargon des réseaux, l'interface de connexion (socket) est celle qui existe entre le processus d'application et le protocole de transport. L'application située au niveau du pôle expéditeur envoie des messages par cette porte, qui sont dès lors pris en charge par le protocole de transport et véhiculés à travers le réseau jusqu'à la porte du processus de réception. De nombreux réseaux, y compris le réseau internet, proposent plusieurs types de protocole de transport, et un développeur doit bien évidemment choisir parmi les protocoles disponibles. Selon quels critères ? En général, il étudie les services proposés par chacun des différents protocoles et opte pour celui qui correspond le mieux aux besoins de son application. Il en va de même lorsque nous devons faire un choix entre l'avion et le train pour un trajet de plusieurs centaines de kilomètres (comme Paris-Nice). Nous devons nous décider pour l'un ou l'autre de ces modes de transport, sachant que chacun propose des services différents. L'avantage du train est qu'il vous amène directement en centre ville, alors que l'avion se distingue par sa rapidité.

Mais comment déterminer les services dont les applications de réseau auront réellement besoin ? Pour se faire une première idée, il est possible de classer les besoins d'une application selon trois grands critères : la fiabilité du transfert de données, le débit et la synchronisation.

Transfert de données fiable

Certaines applications, telles la messagerie électronique, la messagerie instantanée, le transfert de fichiers, la connexion à un serveur distant, le transfert de documents Web et même les applications financières exigent un transfert de données d'une fiabilité à toute épreuve, n'accusant aucune perte de données, aussi infime soit-elle. La perte d'un fichier ou de certaines données d'une transaction financière peut en effet avoir des conséquences désastreuses pour les personnes concernées (autant pour la banque que pour le client !). Les **applications à tolérance d'erreurs**, en particulier, les applications multimédias, telles que le transfert de fichiers audio/vidéo en temps réel, tolèrent la perte d'une quantité limitée de données. Avec ces applications, cette perte se traduit le plus souvent par une petite irrégularité dans le morceau de musique ou le film en lecture, ce qui passe souvent inaperçu ou du moins est loin de constituer un défaut crucial. Les effets d'une perte de données sur la qualité de l'application, et par conséquent, la quantité maximale de paquets pouvant être perdus avant de dépasser le seuil de tolérance, dépendent fortement de la nature de l'application et du système de codage utilisé.

Débit

Pour remplir les fonctions qui leur incombent, certaines applications requièrent un débit minimal disponible en permanence. Si, par exemple, une application de téléphonie par l'internet effectue un codage de la voix à 32 kbit/s, elle doit être en mesure d'envoyer des données et de s'assurer de leur bonne arrivée à ce même débit.

Si le débit requis n'est pas disponible, l'application doit alors coder à un rythme différent et disposer de suffisamment de ressources en ligne pour maintenir cette nouvelle vitesse de codage. Il faut purement et simplement abandonner la connexion lorsque le débit n'atteint pas la valeur requise pour assurer une **application sensible au débit**. La plupart des applications multimédias actuelles sont de ce type, mais les prochaines applications devraient bientôt avoir recours à des techniques de codage plus souples et capables d'adapter la vitesse de codage au débit disponible. Les applications flexibles (messagerie électronique, transfert de fichiers et transfert sur le Web) peuvent s'adapter aux débits disponibles, même s'ils sont très variables. Bien entendu, il est préférable de disposer d'un débit important.

Contraintes de temps

Le dernier critère de bon fonctionnement d'une application est lié au temps. L'efficacité et le confort d'utilisation d'applications interactives de type temps réel, telles que la téléphonie sur l'internet, les environnements virtuels, les conférences téléphoniques et les jeux en réseau, dépendent du respect de strictes contraintes de temps dans le transfert des données. La plupart de ces applications reposent par exemple sur des temps de transmission de bout-en-bout de l'ordre de la centaine de millisecondes, ou moins. (Voir chapitre 6, [Gauthier 1999 ; Ramjee 1994].) Un temps trop long entre la prononciation d'un mot et son écoute à l'autre bout d'une ligne en téléphonie sur internet entraîne une gêne dans la conversation. Avec les jeux en réseau ou au sein d'environnements virtuels interactifs, un temps de réponse au-dessus de la normale après une action (par exemple, de la part d'un joueur à l'autre extrémité d'une connexion de bout-en-bout) affecte le réalisme de l'application. Même s'il est toujours préférable d'avoir des temps d'attente limités, les applications qui ne sont pas de type temps réel n'ont pas de contrainte sur les délais.

Le tableau de la figure 2.4 résume les conditions de fiabilité, de débit et de contraintes de temps des applications internet les plus courantes. Ce tableau synthétique ne cherche pas à réaliser un inventaire complet, mais simplement à identifier les conditions de fonctionnement des différentes applications de réseau.

Figure 2.4. • Conditions de fonctionnement des applications de réseau les plus courantes.

Application	Perte de données	Débit	Sensibilité au temps
Transfert de fichiers	Interdite	Flexible	Non
Courrier électronique	Interdite	Flexible	Non
Pages Web	Interdite	Flexible (quelques kbit/s)	Non

Figure 2.4. • Conditions de fonctionnement des applications de réseau les plus courantes. *(Suite)*

Application	Perte de données	Débit	Sensibilité au temps
Fichiers audio/vidéo en temps réel	Acceptable	Audio : quelques kbit/s à 1 Mbit/s Vidéo : 10 kbit/s à 5 Mbit/s	Oui : quelques centaines de ms
Fichiers audio/vidéo enregistrés	Acceptable	Idem	Oui : quelques secondes
Jeux interactifs	Acceptable	Quelques kbit/s à 10 kbit/s	Oui : quelques centaines de ms
Messagerie instantanée	Interdite	Flexible	Oui et non

2.1.3 Services fournis par les protocoles de transport d'internet

L'internet et, d'une manière générale, les réseaux TCP/IP, proposent deux protocoles de transport aux applications : **UDP** (*User Datagram Protocol*) et **TCP** (*Transmission Control Protocol*). Un développeur doit très tôt opter pour l'un ou l'autre de ces protocoles, sachant qu'ils proposent deux modèles de services très différents.

Services TCP

Le modèle de services TCP comprend un service orienté connexion, et un service de transfert de données fiable. Les applications ayant recours à TCP bénéficient donc des deux services suivants :

- *Service orienté connexion.* Le protocole TCP implique l'échange d'informations de contrôle de couche Transport entre le client et le serveur avant l'échange de messages de couche d'applications. Cette procédure d'échange de présentation avertit à la fois le client et le serveur de l'arrivée de paquets. Une fois l'étape de présentation achevée, on dit qu'une **connexion TCP** a été établie entre les interfaces de connexion des deux processus. La connexion est une connexion dite « duplex », dans le sens où les deux processus peuvent s'envoyer des messages mutuellement et simultanément. Lorsque l'application a envoyé son dernier message, elle ferme la connexion. On parle de service « orienté connexion » plutôt que de service « connexion » (ou de service à « circuits virtuels ») parce que la connexion entre les deux processus est très lâche. Le chapitre 3 présente ce service et son mode d'application.
- *Service de transport fiable.* Les processus de communication peuvent compter sur TCP pour assurer le transfert de leurs données sans erreur et dans l'ordre

souhaité. Lorsque le pôle d'une application expédie un flux d'octets au travers de l'interface de connexion, on peut être certain que ce même flux de données arrive à l'interface de connexion du destinataire, sans aucun octet perdu ou dupliqué.

TCP comprend également un mécanisme de contrôle de congestion, qui est plus destiné au bon fonctionnement de l'internet en général qu'à celui des processus de communication impliqués. Ce mécanisme tend à brider les processus d'expédition (client ou serveur) à la détection de la moindre saturation sur le chemin entre la source et la destination. Comme le montre le chapitre 3, le contrôle d'encombrement effectué par TCP a pour fonction de limiter les connexions TCP au débit le plus adapté.

Cette limitation du débit peut avoir des effets indésirables pour les applications de transfert de fichiers audio et vidéo en temps réel qui exigent un débit minimal. Qui plus est, les applications de type temps réel tolèrent un certain volume de pertes de données et elles n'ont donc pas besoin d'un service de transport d'une fiabilité à toute épreuve. Pour ces raisons et pour d'autres, les développeurs de ce type d'applications ont tendance à préférer UDP à TCP.

Après cette rapide présentation des services TCP, examinons les services que TCP ne propose pas. Tout d'abord, TCP n'établit aucune garantie de débit minimum. En d'autres termes, un processus d'envoi n'a pas la possibilité de transmettre au débit qu'il souhaite, et il est astreint au contrôle d'encombrement de TCP, qui peut dans certains cas fortement limiter le taux de transfert. Deuxièmement, TCP n'établit aucune garantie en ce qui concerne les temps de transfert. Si un processus d'envoi expédie des données au travers d'une interface de connexion TCP, celles-ci arriveront bien à l'interface du processus destinataire, mais personne ne sait en combien de temps. Comme beaucoup d'entre nous l'ont constaté, avec le « World Wide Wait », il faut parfois plusieurs minutes, voire même plusieurs dizaines de minutes, pour qu'un message (contenant par exemple un simple fichier HTML) s'ouvre dans notre navigateur. En bref, TCP ne garantit que la bonne livraison des données, mais pas le débit auxquelles elles sont transmises, ni le temps qu'il leur faudra pour arriver à destination.

Services UDP

UDP est un protocole de transport extrêmement simple et léger, doté d'un modèle de services minimum. Il s'agit tout d'abord d'un service sans connexion, qui ne nécessite donc aucun échange de présentation avant l'établissement d'une communication entre deux processus. De plus, le service de transfert de données procuré par UDP est dit non fiable, ce qui signifie que lorsqu'un processus expédie un message au travers d'une interface de connexion UDP, UDP n'assure aucune garantie sur la remise à bon port. En outre les données n'arrivent pas toujours dans le bon ordre.

UDP n'inclut aucun mécanisme de contrôle d'encombrement, si bien qu'un processus d'envoi peut expédier des données au travers d'une interface de connexion UDP au débit qui lui convient (bien que certaines données risquent de ne jamais arriver à destination). Étant donné que les applications en temps réel ont besoin d'un débit minimum, mais qu'elles tolèrent généralement un certain degré de perte de données,

les développeurs de ce type d'application préfèrent souvent UDP à TCP. Le protocole UDP ne confère pas plus de garantie que TCP sur le temps de transmission des données.

Le tableau de la figure 2.5 recense les protocoles de transport utilisés par certaines des applications internet les plus courantes. Le Web, les applications de messagerie électronique, d'accès des terminaux distants et le transfert de fichiers ont recours à TCP. Ce choix est principalement dû au fait que TCP procure un service de transfert de données fiable qui garantit que toutes les données arriveront bien à destination. En revanche, la téléphonie sur internet a généralement recours à UDP. Les deux pôles de cette application doivent en effet envoyer des données à un débit minimum (voir figure 2.4), ce qui est plus probable avec UDP qu'avec TCP. De même, la tolérance de ce type d'application vis-à-vis des pertes de paquets les affranchit du besoin du service de transfert de données fiable procuré par TCP.

Comme indiqué précédemment, ni TCP ni UDP ne donnent de garantie par rapport aux contraintes de temps. Cela signifie-t-il que les applications sensibles au facteur temps soient exclues de l'internet ? Bien sûr que non : voilà des années que l'internet héberge des applications de ce genre. Celles-ci fonctionnent même souvent relativement bien, car elles ont été conçues pour faire face, dans la mesure du possible, à cette absence de garantie. Certaines des astuces développées à cet effet sont présentées au chapitre 6. Quoi qu'il en soit, même le système le plus ingénieux doit faire face à des temps de réponse excessifs, qui sont fréquents sur internet. En bref, même si l'internet actuel est souvent en mesure de proposer un service acceptable aux applications sensibles au facteur temps, il n'en demeure pas moins dans l'incapacité de fournir des garanties de débit et de contraintes de temps. Le chapitre 6 présente également quelques modèles de services innovants pour l'internet, dont certains garantissent un temps de transmission maximal pour les applications sensibles au facteur temps.

Figure 2.5 • Applications internet les plus utilisées, protocoles de couche d'application et protocoles de transport sous-jacents.

Application	**Protocole de couche Applications**	**Protocole de transport sous-jacent**
Courrier électronique	SMTP [RFC 2821]	TCP
Accès d'un terminal distant	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Transfert de fichiers	FTP [RFC 959]	TCP
Serveur de fichiers distant	NFS [McKusik 1996]	UDP ou TCP
Multimédia en streaming	Souvent propriétaire (par exemple Real Networks)	UDP ou TCP
Téléphonie sur internet	Souvent propriétaire (par exemple Dialpad)	Généralement UDP

2.1.4 Applications de réseau abordées dans cet ouvrage

L'internet voit chaque jour paraître de nouvelles applications qui relèvent du domaine public ou d'initiatives d'industriels. Plutôt que de procéder à un inventaire exhaustif des milliers d'applications disponibles sur le réseau, nous avons préféré présenter quelques applications à grand succès. Dans ce chapitre, nous en détaillerons cinq : le Web, le transfert de fichiers, la messagerie électronique, le service d'annuaire et le partage de fichiers de poste-à-poste. Nous commencerons tout naturellement par le Web, de loin l'application la plus en vogue et dont le protocole d'application, HTTP, est à la fois simple et illustratif d'un grand nombre des principes clés des protocoles de réseau. Nous passerons ensuite à l'application de transfert de fichiers, qui forme un contraste intéressant par rapport à HTTP et qui permet d'aborder certains principes supplémentaires. Puis nous nous pencherons sur la messagerie électronique, la toute première application phare de l'internet, née il y a une trentaine d'années. Le courrier électronique moderne met en jeu plusieurs protocoles de couche d'application. La quatrième application abordée dans ce chapitre, le DNS (*Domain Name System*, Système de noms de domaine) offre un service d'annuaire, auquel peu d'utilisateurs accèdent directement, mais qui se consulte *via* une application tierce (telle que l'une des trois précédentes). Le DNS constitue un très bon exemple de base de données distribuée. Enfin, dans le contexte de la distribution de contenu, nous examinerons aussi le système de partage de fichiers de poste-à-poste (P2P), l'application sur laquelle repose une profusion de sites de téléchargement de fichiers MP3 au succès incontestable.

2.2 Le Web et HTTP

Jusqu'au début des années 1990, l'internet était essentiellement l'affaire de quelques universitaires, chercheurs et étudiants, qui utilisaient le réseau pour se connecter à des serveurs distants, pour transférer des fichiers vers des postes éloignés géographiquement, pour recevoir ou envoyer des informations en tout genre et s'échanger des messages électroniques. Ces applications avaient beau être très utiles, peu de gens en dehors de ces initiés avaient eu vent de ce nouveau moyen de communication. Puis, ce furent les débuts d'une application qui n'allait pas tarder à bouleverser le monde des télécommunications : le World Wide Web [Berners-Lee 1994], souvent appelé Web. Le Web est en effet l'application qui aura finalement attiré l'attention du grand public sur l'internet. Le Web a profondément changé la façon dont les gens communiquent, que ce soit depuis leur lieu de travail ou maintenant de plus en plus depuis leur domicile ou depuis d'autres points d'accès. Ce succès a projeté l'internet du rang de banal réseau d'ordinateurs parmi d'autres réseaux (aux côtés de Prodigy, America Online, CompuServe, de réseaux nationaux, tels que le Minitel/Transpac en France, de réseaux X.25 privés et de réseaux à relais de trames) à celui de réseau majeur, bientôt seul et unique réseau de données.

Histoire de...

La saga Netscape

En avril 1994, Marc Andreesen, informaticien déjà connu pour avoir supervisé le développement du navigateur Mosaic à l'Université d'Illinois à Urbana-Champagne, et Jim Clark, fondateur de Silicon Graphics à l'issue d'une carrière de professeur à l'Université Stanford, créèrent la Netscape Communication Corporation. Une grande partie de l'équipe Netscape était constituée d'anciens collaborateurs du projet Mosaic. La version bêta du Navigator 1.0 fut disponible dès le mois d'octobre 1994. De nombreuses améliorations furent par la suite apportées au logiciel de navigation, parallèlement au développement de serveurs Web, de serveurs commerciaux, de serveurs de messagerie électronique, de serveurs d'informations, de serveurs proxy, de lecteurs de messagerie et de bien d'autres logiciels relevant de la couche d'application. Netscape a probablement été l'une des sociétés internet les plus innovantes et les plus chanceuses de cette période charnière. En janvier 1995, Jim Barksdale fut nommé PDG de Netscape et en août de cette même année, l'introduction en bourse de la société mis son nom sur toutes les bouches.

Microsoft, plus lent à s'engager dans la ruée vers l'internet, ne commercialisa son premier navigateur, Internet Explorer 1.0, qu'en août 1995. Sa conception laissait grandement à désirer, mais Microsoft choisit finalement d'investir en masse dans ce développement et dès fin 1997, Internet avait rejoint Netscape dans la course aux navigateurs. Le 11 juin 1997, Netscape inaugura la version 4.0 de son Navigator, moins de trois mois avant la parution d'Internet Explorer 4.0 (30 septembre). N'arrivant pas à se démarquer sur le plan technique, Microsoft joua de son monopole en matière de système d'exploitation Windows pour gagner d'importantes parts de marché. De son côté, Netscape fit cette année là plusieurs erreurs stratégiques : il sous-estima le potentiel de son site internet pour attirer et fidéliser la clientèle et précipita le développement d'un navigateur tout-Java (à une période où le Java n'était pas encore prêt à assumer cette tâche) [Cusumano 1998]. L'année suivante vit le déclin progressif du navigateur et des autres produits logiciels de Netscape sur le marché. À la fin de 1998, Netscape fut racheté par America Online. Depuis, Marc Andreesen et la majeure partie de l'équipe qui avait participé à la création de Netscape ont quitté le navire.

L'histoire est jalonnée d'avancées technologiques dans le domaine des télécommunications qui a eu un impact très important sur les modes de vie. Tout a commencé avec l'invention du téléphone en 1870 qui, du jour au lendemain, a permis aux gens de communiquer en temps réel sans avoir à se trouver dans le même lieu.

L'impact du téléphone a eu une grande répercussion sur la société, avec des conséquences positives et négatives. La percée technologique suivante fut celle de la radiodiffusion, suivie de la télévision, qui virent le jour respectivement dans les années 1920 et 1930. La diffusion de programmes de radio et de télévision permit aux foyers d'avoir accès à une profusion d'informations audiovisuelles. À nouveau, ces développements ont eu un impact sur la société, à la fois positif et négatif. La dernière technologie de communication en date s'avère tout simplement être le Web, dont l'attrait caractéristique est d'être un service d'informations à la demande, c'est-à-dire fournissant aux utilisateurs les données qu'ils souhaitent au moment où ils le désirent, à la différence des technologies précédentes. Le Web propose en outre toute une gamme d'autres services très appréciés du public. Il permet par exemple à tout un chacun de mettre des informations en ligne, n'importe qui pouvant désormais se déclarer auteur ou éditeur à peu de frais. Un système de liens hypertexte et un large éventail de moteurs de recherche facilitent l'accès à des données qui, autrement, demeureraient introuvables au cœur d'un océan de sites Web. Les environnements graphiques stimulent nos sens et aiguisent notre curiosité. Les animations JavaScript, applettes Java et composants Active X, pour ne citer que ceux-ci, nous offrent une possibilité de contacts interactifs avec les pages et les sites. Et de plus en plus, le Web propose sur l'internet un choix inépuisable de fichiers audio et vidéo, soit autant de données multimédias accessibles sur simple demande.

2.2.1 Description générale de HTTP

Le protocole HTTP (*HyperText Transfer Protocol*), protocole de couche d'applications utilisé par le Web, est défini par les normes [RFC 1945] et [RFC 2616]. HTTP est mis en œuvre par l'interaction de deux logiciels : un logiciel client et un logiciel serveur. Exploités sur des systèmes terminaux différents, ceux-ci communiquent entre eux au moyen de messages HTTP. HTTP définit la structure de ces messages et leurs modalités d'échange entre client et serveur. Avant d'étudier ce protocole plus en détail, il est important de définir certains points de vocabulaire propres au domaine du Web.

Une **page Web** (aussi appelée document Web) est constituée de différents objets. Un **objet** est tout simplement un fichier — un fichier HTML, une image compressée au format GIF ou JPEG, une applette Java, un clip audio, etc. — accessible au moyen d'une adresse propre (URL, *Uniform Resource Location*, ou Localisateur normalisé de ressource). La plupart des pages Web se composent d'un fichier HTML de base et d'un certain nombre d'objets référencés. Si une page Web contient par exemple du texte HTML et cinq images JPEG, elle se compose de six objets : le **fichier HTML de base** et les cinq images. Le premier permet de référencer les autres objets dans la page au moyen de leurs adresses URL respectives. Chaque URL comprend deux éléments : le nom du serveur hébergeant l'objet et le chemin d'accès de l'objet. Par exemple, dans l'URL

```
www.universitedauphine.edu/departementY/image.gif
```

`www.universitedauphine.edu` est le nom de serveur et `/departementY/image.gif` le chemin d'accès. Un **navigateur** est un agent utilisateur pour le Web. Il affiche les pages Web sollicitées et propose de nombreuses fonctions de navigation et de configuration. Les navigateurs Web mettent en œuvre également le pôle client de HTTP, si bien qu'en parlant du Web, nous aurons alternativement recours aux termes « navigateur » et « client ». Les navigateurs Web les plus répandus sont Netscape Communicator et Microsoft Internet Explorer.

Un **serveur Web** héberge des objets Web, tous accessibles au moyen d'une adresse URL. Les serveurs Web mettent également en œuvre le pôle serveur de HTTP. Les serveurs Web les plus utilisés sont les serveurs Apache et Microsoft Internet Information Server. (Netcraft propose une analyse intéressante du marché des serveurs Web [Netcraft 2002].)

HTTP définit les modalités de requête et de transfert de pages Web entre un client (par exemple, un navigateur) et un serveur. Cette interaction client-serveur est examinée plus loin en détail. La figure 2.6 en présente l'idée générale. Lorsqu'un utilisateur sollicite une page Web (en cliquant par exemple sur un hyperlien), le navigateur envoie au serveur des messages de demande HTTP concernant les objets contenus dans la page. Celui-ci y répond au moyen de messages HTTP contenant les objets sollicités. Jusqu'en 1997, presque tous les navigateurs et les serveurs Web reposaient sur HTTP version 1.0, défini dans le RFC 1945. Puis l'on vit apparaître HTTP/1.1, défini dans le RFC 2616 et compatible avec la version précédente. Serveurs Web et navigateurs exploités en version 1.1 peuvent « parler » avec leurs homologues en 1.0 et *vice versa*.

Ces deux versions de HTTP ont recours au protocole de transport TCP. Le client HTTP établit dans un premier temps une connexion avec le serveur, puis navigateur et serveur accèdent à TCP *via* leurs interfaces de connexion. Comme précisé à la section 2.1, cette interface agit au niveau du pôle client comme une « porte » entre le processus client et la connexion TCP, puis au niveau du pôle serveur comme une autre « porte » entre la connexion TCP et le processus serveur. Le client envoie des messages de demande HTTP au travers de l'interface de connexion et reçoit des messages de réponse HTTP de sa part. Il en va de même pour le serveur HTTP, qui reçoit des messages de demande de la part de son interface de connexion et s'en sert pour expédier ses messages de réponse. Une fois que le client a envoyé son message, celui-ci est considéré comme « hors de sa portée » et placé « sous le contrôle de TCP » qui procure un service de transfert de données fiable (voir section 2.1). Ceci signifie que tout message de demande HTTP émis par un processus client arrive intact au serveur. De la même manière, tous les messages de réponse HTTP émis par le processus serveur parviennent intacts au client. C'est ici que réside l'un des grands avantages de l'architecture en couches : HTTP n'a pas besoin de s'inquiéter des données perdues ou de la façon dont TCP se remet d'une perte ou encore de la réorganisation des données au sein du réseau, car cette tâche incombe à TCP seul et aux protocoles des couches inférieures de la pile.

Avec http, le serveur envoie les fichiers au client sans stocker d'information d'état relative au client. En conséquence, si un client sollicite le même objet deux fois de

suite dans un délai relativement court, n'étant pas informé du fait que l'opération vient de se produire, l'objet sera renvoyé une seconde fois, et ainsi de suite. Dans la mesure où un serveur HTTP ne conserve aucune information sur ses clients, on parle de **protocole sans mémoire** (*stateless protocol*).

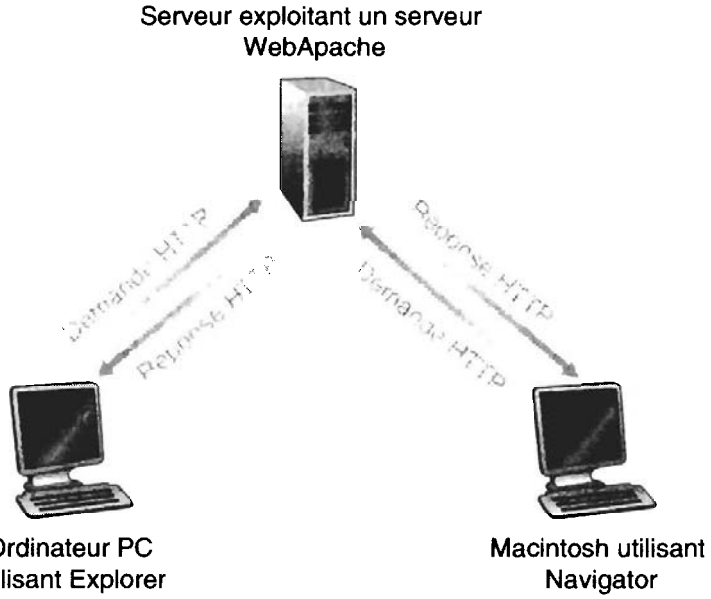

Figure 2.6 • Procédure de demande/réponse du HTTP.

2.2.2 Connexions persistantes et non-persistantes

HTTP peut mettre en œuvre des connexions persistantes et des connexions non-persistantes. Par exemple, la version HTTP 1.0 utilise des connexions non-persistantes. En présence de connexions non-persistantes, un seul objet Web peut être transféré à la fois sur une connexion TCP. La version 1.1 utilise des connexions persistantes par défaut, mais elle peut aussi bien être configurée pour utiliser des connexions non-persistantes. Dans le cas de connexions persistantes, il n'est pas nécessaire d'établir une nouvelle connexion pour le transfert de chaque nouvel objet.

Connexions non-persistantes

Parcourons donc brièvement les étapes se succédant lors du transfert d'une page Web entre un serveur et un client dans le cas de connexions non-persistantes. Supposons que la page comprenne un fichier de base HTML et dix images au format JPEG, et que tous ces objets résident dans le même serveur. Soit :

▶ `www.universitedauphine.edu/departementY/home.index`

l'URL du fichier de base HTML.

Voici la chronologie des différentes étapes :

1. Le client HTTP établit une connexion TCP avec le serveur `www.universite-dauphine.edu` sur l'accès 80, qui est le numéro d'accès par défaut de HTTP.

2. Le client HTTP envoie un message de demande HTTP au serveur *via* l'interfac de connexion associée à la connexion TCP établie dans l'étape précédente. C message contient le nom du chemin d'accès `/departementY/home.index`. (L contenu des messages HTTP est examiné plus loin dans cette section.)

3. Le serveur HTTP reçoit le message de demande *via* l'interface de connexion associé à la connexion, retire l'objet `/departementY/home.index` de son lieu de stockage (qu est une mémoire RAM ou un disque dur), l'intègre dans un message de répons HTTP et envoie le tout au client, toujours *via* l'interface de connexion.

4. Le serveur HTTP ordonne à TCP d'interrompre la connexion. (Mais TCP n s'exécute pas avant d'avoir obtenu la confirmation de la réception correcte d message de réponse par le client.)

5. Le client HTTP reçoit le message de réponse. La connexion TCP est interrompue Le message indique que l'objet qu'il transporte est un fichier HTML. Le clien extrait le fichier HTML du message reçu, l'examine et y trouve les références de dix objets JPEG.

6. Les quatre premières étapes sont alors répétées pour chacun des objets référencés

Une fois que la page est parvenue au navigateur, celui-ci l'affiche sur l'écran de l'utili sateur. Il faut savoir que des navigateurs différents peuvent interpréter (et donc affi cher) une même page Web de façon différente. HTTP ne doit pas être mis en cause ce propos. Les spécifications de HTTP ([RFC 1945] et [RFC 2616]) ne portent qu sur le protocole de communication entre client et serveur.

On dit que la procédure précédente a recours à des connexions non-persistantes dan la mesure où les connexions TCP sont rompues après le passage d'un objet et ne son donc pas maintenues pour d'autres. Notez que les différentes connexions TCP n transportent qu'un message de demande et un message de réponse. Ainsi, dan l'exemple donné, lorsqu'un utilisateur sollicite la page Web, ce ne sont pas moins d 11 connexions qui doivent être établies.

En décrivant cette opération, nous sommes restés volontairement vagues sur un question : le client recevait-il les 10 images JPEG au moyen de dix connexions succes sives ou certaines lui parvenaient-elles *via* des connexions TCP parallèles ? En effet les navigateurs récents peuvent être configurés pour maîtriser le degré de paral lélisme. Dans les procédures par défaut, la plupart d'entre eux établissent de cinq dix connexions en parallèle, chacune d'elles étant limitée à un seul échange de messa ges de demande/réponse. S'il le souhaite, l'utilisateur peut également interdire l parallélisme. Dans ce cas, les 10 connexions de l'exemple ci-dessus ont lieu l'un après l'autre. Le chapitre 3 montre que l'utilisation de connexions en parallèle s traduit par une réduction du temps de réponse.

Avant d'aller plus loin, essayons brièvement d'estimer le temps qu'il faut à un clien pour obtenir le fichier qu'il sollicite. Dans ce but, nous définissons la durée du traje aller-retour (RTT, *Round-trip Time*), comme le temps nécessaire à un petit paque pour faire un aller-retour entre le client et le serveur. Ce RTT est la somme du temp de propagation, du temps d'attente au niveau des divers routeurs et commutateur

intermédiaires et du temps de traitement des paquets (voir description à la section 1.6). Considérez maintenant ce qui se produit lorsque l'utilisateur clique sur un hyperlien. Comme l'indique la figure 2.7, ceci conduit le navigateur à établir une connexion TCP avec le serveur Web, opération qui implique un « échange de présentation en trois étapes », au cours duquel le client envoie un petit segment TCP au serveur, attend un accusé de réception de sa part (sous la forme d'un autre segment TCP) et l'informe finalement de la bonne réception de la réponse. Le RTT se limite aux deux premières phases de la procédure d'échange de présentations. À l'issue de ces deux phases, le client envoie le message de demande HTTP associé à la troisième étape (l'accusé de réception) sur la connexion TCP. Une fois ce message arrivé au serveur, ce dernier répond au moyen du fichier HTML sollicité. Cette procédure de demande/réponse implique un nouveau RTT, ce qui porte le temps d'attente total à environ deux RTT plus le temps de transmission du fichier HTML depuis le serveur.

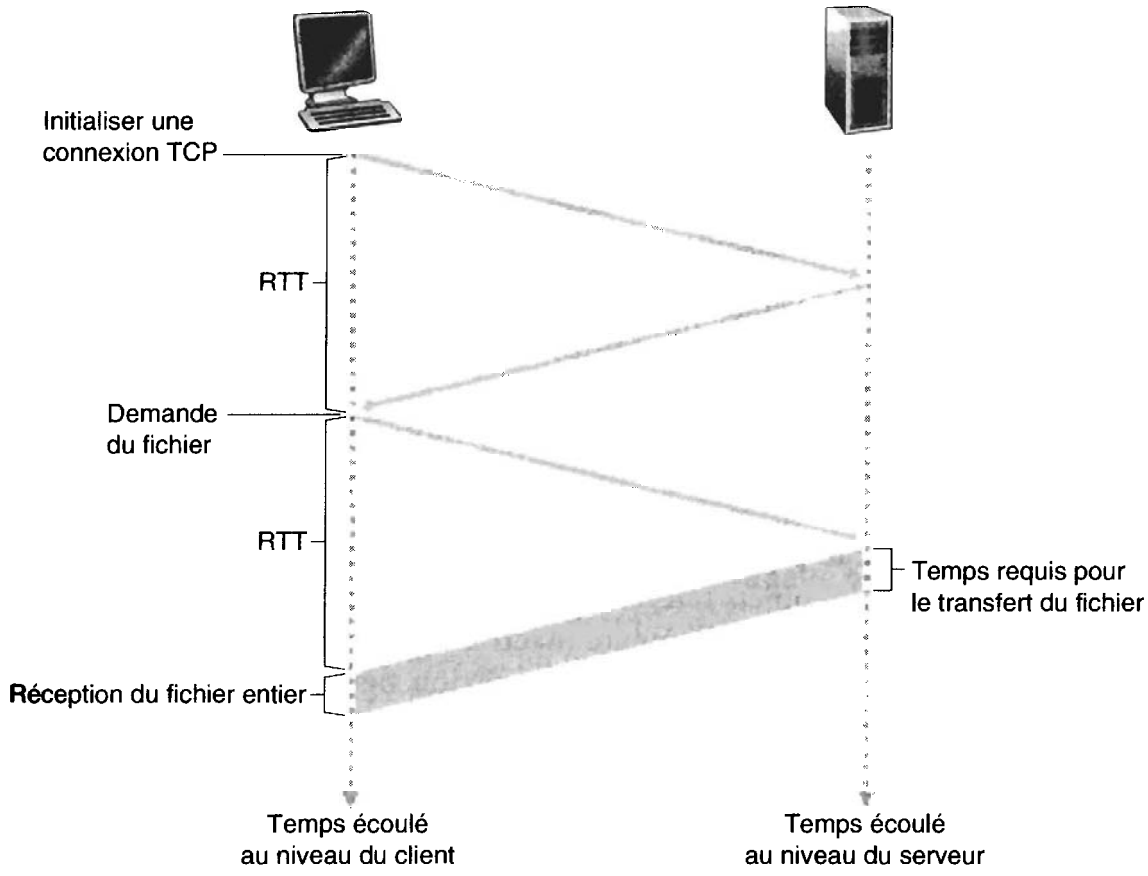

Figure 2.7 • Estimation du temps d'attente associé à la requête d'un fichier HTML.

Connexions persistantes

Les connexions non-persistantes présentent certains inconvénients. Tout d'abord, une nouvelle connexion doit être établie et maintenue pour chaque objet sollicité. Chacune d'entre elles implique l'allocation de tampons TCP qui doit être effectuée au niveau du client et du serveur. Ceci peut représenter une charge importante pour le serveur Web, qui peut parfois être appelé à satisfaire des requêtes provenant de centaines d'utilisateurs simultanément. De plus, la livraison de chaque objet implique un

temps d'attente de deux RTT : un premier lors de l'établissement de la connexion TCP et un second pour la requête et la réception de l'objet.

Avec des connexions persistantes, la connexion TCP est maintenue par le serveur après l'envoi de la réponse. Par conséquent, tous les échanges entre un client et un serveur peuvent emprunter la même connexion. Si nous reprenons l'exemple de la page Web constituée de 10 images, l'ensemble peut être envoyé le long d'une seule connexion TCP persistante. Qui plus est, plusieurs pages Web hébergées sur le même serveur peuvent se succéder sans interrompre la connexion. D'une manière générale, le serveur HTTP interrompt la connexion lorsque celle-ci est restée inactive pendant un certain temps (qui fait l'objet d'une temporisation configurable).

Il existe deux types de connexions persistantes : les connexions dites **avec** ou **sans pipelinage**. En l'absence de pipelinage, le client émet une nouvelle requête uniquement après la réception de la réponse à sa requête précédente. Dans ce cas, il subit un RTT pour la requête et la réception de chaque objet contenu dans la page qu'il sollicite (par exemple, les 10 images de notre page Web). Bien que ceci représente déjà un progrès sensible par rapport aux deux RTT par objet associés aux connexions non-persistantes, la méthode du pipelinage permet encore de réduire le RTT. Autre inconvénient de la connexion sans pipelinage : une fois qu'un objet a été envoyé le long de la connexion TCP persistante, celle-ci se retrouve inactive, dans l'attente d'une autre requête. Or, toute inactivité constitue un gaspillage de ressources du serveur.

Le mode par défaut de la version 1.1 de HTTP utilise des connexions persistantes avec pipelinage. Dans cette configuration, le client HTTP émet une requête dès qu'il rencontre une référence à un objet. Il est ainsi en mesure d'émettre des requêtes successives dans le cas où la page sollicitée compterait plusieurs objets, c'est-à-dire qu'il peut émettre une nouvelle requête avant même d'avoir reçu la réponse à sa requête précédente. Le serveur, lui, envoie les objets au rythme auquel lui arrivent les requêtes. Avec le pipelinage, il est donc possible de ne générer qu'un seul RTT pour l'ensemble des objets référencés (au lieu d'un RTT par objet). De plus, la connexion TCP avec pipelinage demeure inactive beaucoup moins longtemps. Des exercices confrontant les deux modes de connexion figurent à la fin de ce chapitre et au chapitre suivant. Les lecteurs curieux peuvent consulter [Heidemann 1997 et Nielsen 1997].

2.2.3 Format des messages HTTP

Les spécifications des versions 1.0 ([RFC 1945]) et 1.1 ([RFC 2616]) du protocole HTTP définissent un format à respecter pour les messages HTTP. Il existe deux types de messages HTTP : les demandes et les réponses, dont la description figure ci-après.

Messages de demande HTTP

Voici un message de demande HTTP classique :

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
```

```
Connection: close
User-agent: Mozilla/4.0
Accept-language:fr
```

Examinons à présent ce message. Tout d'abord, il est rédigé en format ASCII, si bien qu'il est compréhensible pour toute personne connaissant un peu l'informatique. Ensuite, il est composé de cinq lignes, chacune terminée par un retour chariot. Un message de demande peut contenir beaucoup plus de lignes, tout comme se limiter à une ou deux seulement. La première ligne est appelée **ligne de demande**, les suivantes **lignes d'en-tête**. La ligne de demande se compose de trois champs qui traitent respectivement la méthode, l'adresse URL et la version du protocole HTTP utilisée. Le champ relatif à la méthode peut prendre différentes valeurs : `GET`, `POST` ou `HEAD`. La grande majorité des messages de demande HTTP a recours à la méthode `GET`. C'est celle qui est utilisée lorsque le navigateur sollicite un objet et dont l'URL est mentionnée dans le champ prévu à cet effet. Dans l'exemple ci-dessus, le navigateur sollicite l'objet `/somedir/page.html` et fait référence à la version 1.1 de HTTP.

Passons maintenant à l'analyse des lignes d'en-tête de l'exemple ci-dessus. La première `Host: www.someschool.edu` désigne le serveur qui héberge l'objet. Au moyen de la ligne `Connection: close`, le navigateur informe le serveur qu'il ne veut pas utiliser de connexion persistante, mais qu'il souhaite que la connexion soit interrompue après l'envoi de l'objet sollicité. Bien que le navigateur responsable de ce message de demande mette en œuvre la version 1.1 de HTTP, il ne semble pas vouloir s'encombrer de connexions persistantes pour réaliser cette opération. La ligne suivante, `User-agent: Mozilla/4.0`, spécifie l'agent utilisateur utilisé, c'est-à-dire le type de navigateur à l'origine de la demande. Dans le cas présent, il s'agit d'un navigateur Netscape. L'utilité de cette ligne d'en-tête réside dans le fait que le serveur peut envoyer différentes versions du même objet à différents types d'agents utilisateurs. (On accède à chacune des versions au moyen de la même URL.) Enfin, la ligne `Accept-language:fr` indique que l'utilisateur souhaite recevoir la version française de l'objet, si elle existe. Dans le cas contraire, le serveur lui fait parvenir la version par défaut. Nous sommes ici en présence de l'un des nombreux en-têtes de négociation du protocole HTTP.

Après l'étude de cet exemple, passons maintenant au format général d'un message de demande, tel qu'illustré à la figure 2.8. Ce format est très proche de celui de l'exemple précédent, à ceci près que les lignes d'en tête sont suivies d'un « squelette ». Alors que ce dernier n'est pas utilisé par la méthode `GET`, la méthode `POST` s'en sert pour tous les champs et formulaires à remplir en ligne, tels que l'entrée de mots clés dans un moteur de recherche. Avec un message de type `POST`, l'utilisateur sollicite toujours une page Web auprès d'un serveur, mais son contenu spécifique dépend des informations contenues dans le squelette.

Notez que la formulation d'une demande à l'aide d'un formulaire HTML n'a pas uniquement recours à la méthode `POST`, mais également à la méthode `GET`. Dans ce cas, les données de recherche entrées par l'utilisateur sont transposées dans l'URL. Par exemple, un formulaire utilisant la méthode `GET` et comprenant deux champs contenant les

mots clés `singes` et `bananes` présente l'URL suivante : `www.siteweb.com/recherche-animaux?singes&bananes`. En surfant sur le Web, vous avez probablement souvent pu observer ce genre d'adresse.

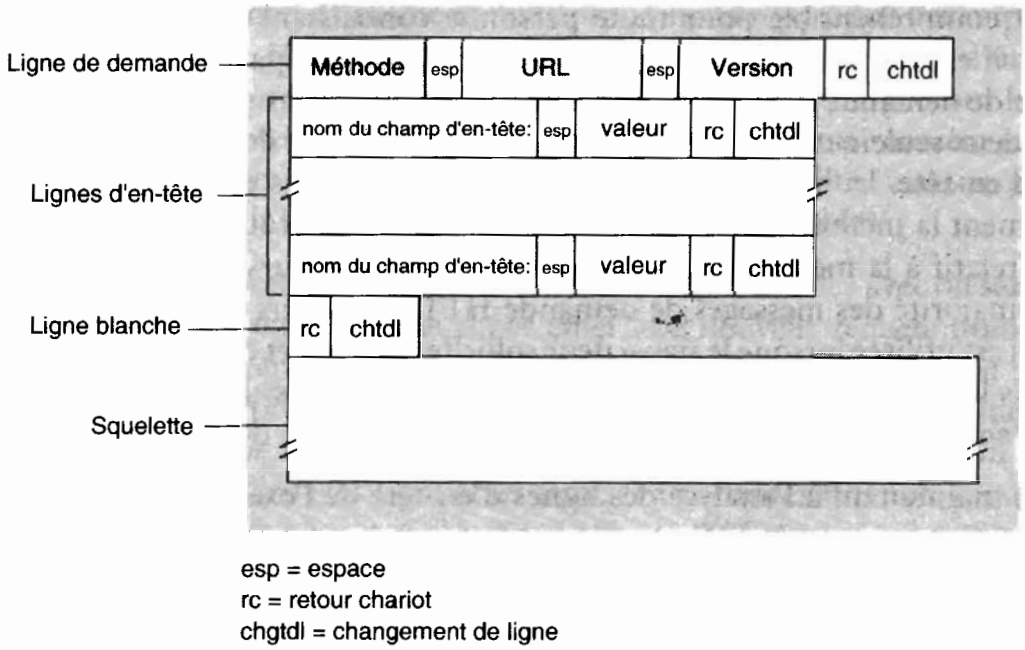

Figure 2.8 • Format général d'un message de demande HTTP.

La méthode `HEAD` est très proche de la méthode `GET`, à ceci près que lorsqu'un serveur reçoit une demande de type `HEAD`, le message HTTP qu'il génère en réponse ne contient pas l'objet sollicité. Les développeurs utilisent souvent la méthode `HEAD` lors du débogage.

Alors que HTTP/1.0 ne propose que les trois méthodes décrites ci-dessus (`GET`, `POST` et `HEAD`), la version 1.1 en autorise plusieurs autres, dont les méthodes `PUT` et `DELETE`. La première, qui permet à un utilisateur de charger un objet vers un chemin (ou un répertoire) précis au sein d'un serveur donné, est spécifique aux outils de création Web. La seconde permet à un utilisateur ou à une application d'effacer un objet hébergé sur un serveur Web.

Messages de réponse HTTP

Voici un message de réponse typique, qui peut venir en réponse au message de demande précédent :

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
(données données données données...)
```

Ce message est composé de trois sections : une **ligne d'état**, six **lignes d'en-tête** et le **squelette**. Ce dernier contient la partie la plus importante du message, à savoir l'objet sollicité (représenté ici par la ligne `données données données données...`). La ligne d'état est constituée de trois champs : la version de HTTP, un code d'état et un message d'état correspondant. Dans notre exemple, cette ligne indique que le serveur utilise http 1.1 et que tout est en ordre (c'est-à-dire que le serveur a trouvé l'objet et que l'envoi est en cours).

Le serveur utilise la formule d'en-tête `Connection: close` pour informer le client que la connexion TCP sera coupée après l'envoi du message. La ligne `Date:` indique le jour et l'heure de la création et de l'envoi de la réponse HTTP par le serveur. Notez qu'elle ne correspond pas à la date de création ou de dernière modification de l'objet sollicité, mais plutôt à l'heure à laquelle le serveur a retiré l'objet de son système de fichiers, intégré l'objet dans le message de réponse et procédé à l'envoi. La ligne `Server:`, qui informe que le message provient d'un serveur Web Apache, est analogue à la ligne `User-agent:` du message de demande HTTP. La ligne `Last-Modified:` indique le jour et l'heure de la création ou de la dernière modification de l'objet en question. Cette information se révèle capitale pour la mise d'objets en mémoire cache, à la fois au sein des postes clients et des serveurs cache de réseau (aussi connus sous le nom de serveurs proxy). La ligne `Content-Length` indique le nombre d'octets contenu dans l'objet envoyé. L'en-tête `Content-Type` indique que l'objet dans le squelette est du texte HTML. Le type de l'objet est officiellement donné par l'en-tête `Content-Type:` et non par l'extension du fichier.

Notez que si le serveur reçoit une demande de type HTTP 1.0, les connexions utilisées seront des connexions non-persistantes, même s'il s'agit d'un serveur HTTP 1.0. En d'autres termes, la connexion TCP sera coupée après l'envoi de l'objet. Ceci est obligatoire parce qu'un client HTTP 1.0 s'attend à ce que la connexion soit coupée après réception de la réponse.

Après avoir considéré un exemple spécifique, passons maintenant au format général d'un message de réponse HTTP, en nous aidant du schéma présenté à la figure 2.9. Ce format général s'apparente fortement à notre cas de figure. Le code d'état et les formules associées rendent compte des résultats de la réponse. Voici quelques codes d'état et les phrases qui leur sont associées employés couramment :

- `200 OK`: La requête est réussie et l'information est contenue dans la réponse.
- `301 Moved Permanently`: L'objet sollicité a été définitivement déplacé ; une nouvelle URL est spécifiée dans la ligne d'en-tête `Location`: du message de réponse. Le navigateur du client se dirigera automatiquement vers la nouvelle adresse.
- `400 Bad Request`: Code d'erreur générique indiquant que la demande n'a pas été comprise par le serveur.
- `404 Not Found`: Le document recherché est introuvable sur le serveur.
- `505 HTTP Version Not Supported`: La version du protocole HTTP requise n'est pas disponible sur le serveur.

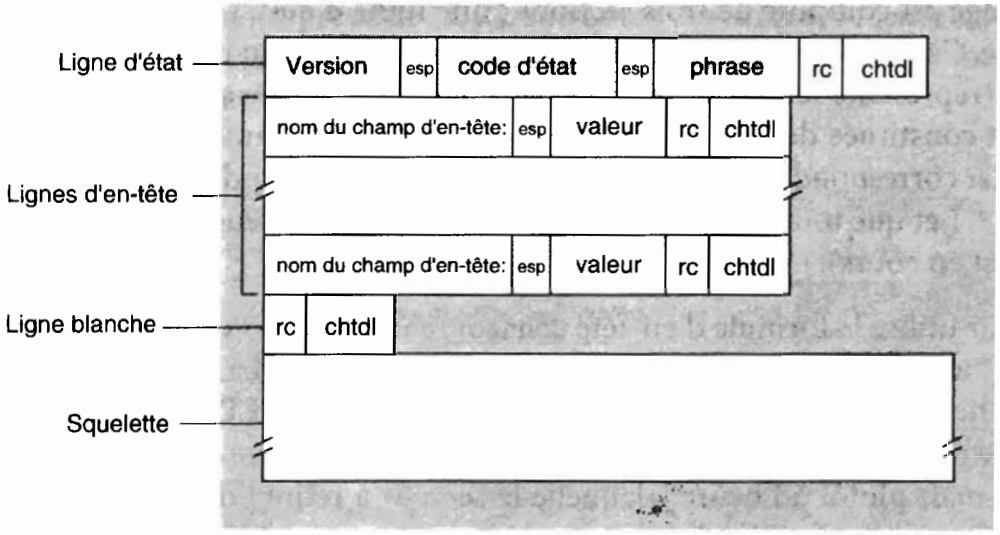

Figure 2.9 • Format général d'un message de réponse.

Aimeriez-vous voir un véritable message de réponse HTTP ? C'est un jeu d'enfant et nous vous le recommandons ! Ouvrez une session Telnet avec le serveur de votre choix, puis lancez une requête concernant un objet qu'il héberge. Par exemple, si vous avez accès à un ordinateur sous UNIX, entrez :

```
telnet www.eurecom.fr 80
GET /~ross/index.html HTTP/1.0
```

(Faites deux retours chariot après avoir tapé la seconde ligne). Ceci établit une connexion TCP avec l'accès 80 du serveur `www.eurecom.fr` et transmet la commande `HTTP GET`. Vous devez voir apparaître un message de réponse contenant le fichier de base HTML de la page d'accueil du site du professeur Ross. Si vous souhaitez seulement voir les lignes du message HTTP sans consulter l'objet sollicité, remplacez `GET` par `HEAD`. Enfin, remplacez `/~ross/index.html` par `/~ross/banana.html` et observez la réponse obtenue.

Vous connaissez maintenant quelques lignes d'en-tête des messages de demande et de réponse HTTP. Mais les spécifications de HTTP (et surtout de la version 1.1) définissent de nombreuses variétés d'en-têtes, tant en provenance des navigateurs et des serveurs Web que des serveurs cache. D'autres sont évoquées avec la mise en mémoire cache sur le Web à la fin de ce chapitre. Une présentation très limpide et très complète du protocole HTTP/1.1, incluant ses principaux codes d'état et en-têtes, figure dans [Krishnamurty 2001]. [Luotonen 1998] permet de découvrir le point de vue du développeur. Une excellente introduction au Web est disponible dans [Yeager 1996].

Comment un navigateur décide-t-il des lignes d'en-tête à inclure dans un message de demande ? Comment un serveur Web choisit-il les lignes que doit contenir sa réponse ? Les lignes d'en-tête du message de demande dépendent du type et de la version du navigateur (un navigateur HTTP/1.0 ne génère par exemple que des lignes

d'en-tête de type 1.0), de sa configuration par l'utilisateur (par exemple en ce qui concerne la langue souhaitée) et de la présence ou non d'une version (éventuellement périmée) de l'objet sollicité dans la mémoire cache. Les serveurs Web procèdent de la même manière : ils existent sous différents modèles, versions et configurations, autant de paramètres influant sur les lignes d'en-tête qui sont incluses dans les messages de réponse.

2.2.4 Échanges utilisateur–serveur : autorisation et cookies

Nous avons indiqué ci-dessus qu'un serveur HTTP est sans mémoire et qu'il ne conservait pas d'informations d'état. Cette conception simplifiée a permis aux ingénieurs de développer des serveurs hautement performants pouvant traiter des milliers de connexion TCP simultanément. Toutefois, il est souvent dans l'intérêt d'un site Web de pouvoir identifier ses visiteurs, soit pour limiter l'accès à une certaine catégorie d'utilisateurs, soit afin d'adapter le contenu des pages Web à l'identité de l'utilisateur. HTTP procure au serveur deux mécanismes d'identification : la procédure d'autorisation et les cookies.

Procédure d'autorisation

Vous avez probablement remarqué que de nombreux sites exigent un nom d'utilisateur et un mot de passe avant de donner accès aux documents contenus sur le serveur. Cette procédure, dite **procédure d'autorisation**, repose sur l'utilisation d'en-têtes HTTP et de codes d'état particuliers, dont nous vous proposons de découvrir un exemple ci-après.

Supposez qu'un client demande un objet à un serveur configuré pour appliquer une procédure d'autorisation. Le client envoie tout d'abord un message de demande ordinaire, c'est-à-dire libre de toute ligne d'en-tête particulière, auquel le serveur répond par un corps d'entité vide contenant le code d'état `401 Authorization Required`. À cette réponse, le serveur ajoute l'en-tête `WWW-Authenticate:`, qui spécifie les détails de la procédure d'authentification. En général, celle-ci consiste à inviter l'utilisateur à donner son nom et son mot de passe.

Sur réception du message de réponse, le client (c'est-à-dire ici, son navigateur) invite l'utilisateur à s'identifier, puis renvoie le message de demande accompagné d'une ligne d'en-tête `Authorization:` suivie du nom et du mot de passe entrés par l'utilisateur. Après l'obtention du premier objet et au cours des demandes suivantes au même serveur, le client continue de préciser ces données d'utilisateur, désormais placées dans la mémoire cache. Ceci se poursuit généralement jusqu'à ce que le client ferme son navigateur, de sorte que celui-ci n'a besoin d'entrer ses données personnelles qu'une seule fois par session. De cette façon, le site est en mesure d'identifier l'auteur de chaque demande.

Le chapitre 7 montre que HTTP propose une procédure d'autorisation relativement vulnérable, c'est-à-dire facile à contourner. Des systèmes d'autorisation plus robustes et donc beaucoup plus fiables y sont présentés.

Cookies

Les cookies, définis dans le RFC 2109, constituent un autre système permettant aux sites Web de rassembler des données sur leurs visiteurs. Tous les sites sont loin d'en faire usage, mais la plupart des grands portails (par exemple Yahoo), les sites de commerce électronique (par exemple Amazon) et les publicitaires (par exemple DoubleClick) sont adeptes de cette pratique.

La technologie des cookies suppose le déroulement des quatre étapes suivantes : (1) l'insertion d'une ligne d'en-tête particulière dans le message de réponse HTTP ; (2) la réitération du message de demande avec la ligne d'en-tête correspondante ; (3) l'envoi d'un fichier témoin qui est conservé dans le poste de l'utilisateur et est activé par le navigateur ; (4) l'intégration des informations dans une base de données située sur le serveur du site Web. Prenons un exemple typique d'implantation de cookie. Imaginez une personne accédant toujours au Web à l'aide du navigateur Internet Explorer installé sur son ordinateur et visitant pour la première fois un site de commerce électronique ayant recours aux cookies. À l'arrivée de la première demande au niveau du serveur, le site Web choisit un numéro d'identification unique et crée une entrée au sein de sa base de données. Le serveur envoie alors un message de réponse comprenant l'en-tête `Set-cookie:` suivi du numéro d'identification. Par exemple :

```
Set-cookie: 1678453
```

À réception de ce message, le navigateur prend connaissance de l'en-tête et ajoute une ligne au fichier cookie qu'il administre. Cette ligne contient le nom du serveur ainsi que le numéro d'identification dans l'en-tête `Set-cookie:`. Au cours de sa visite à ce site de commerce électronique, le navigateur accompagne chaque message de demande du numéro d'identification contenu dans le cookie correspondant, au moyen de l'en-tête suivant :

```
Cookie: 1678453
```

Le serveur sera ainsi en mesure de suivre les activités de l'utilisateur sur le site Web. Bien qu'il ne connaisse pas forcément son véritable nom, il sait exactement quelles pages l'utilisateur N° 1678453 a visitées, dans quel ordre et à quelle heure de la journée ! Fort de ce type d'informations, le site de vente en ligne peut par exemple édifier un service dit de « panier », qui consiste à maintenir une liste de tous les articles choisis par l'utilisateur en vue de leur paiement à la fin de la session.

Le navigateur de l'utilisateur intègre la ligne d'en-tête `Cookie: 1678453` aux messages de demande à chaque nouvelle visite de ce site, permettant à ce dernier de lui proposer des produits correspondant aux pages Web visitées au cours des sessions précédentes. Si le client décide également de remplir un profil utilisateur, généralement constitué de ses nom, prénom et adresse (de courrier électronique et postale) ainsi que de son numéro de carte de crédit, le site pourra intégrer ces informations dans sa base de données et associer le numéro d'identification au nom de l'utilisateur (et toutes les pages que celui-ci a visitées sur ce site !). Voici comment les sites de commerce électronique mettent en œuvre le fameux système d' « achat en un clic », qui permet

à l'utilisateur de retourner sur le site et d'acheter un article en ligne sans passer par une procédure d'enregistrement.

Cet exemple montre que les cookies peuvent servir à l'authentification des utilisateurs. La première fois qu'une personne visite un site Web, elle est invitée à s'identifier (en tapant son nom, par exemple). À partir de ce moment, le navigateur accompagnera chaque message de demande pour ce site d'une ligne d'en-tête de cookie, permettant l'identification de l'utilisateur. Il ressort également de cet exemple que les cookies peuvent être utilisés pour superposer une couche de session d'utilisateur à HTTP. Lorsqu'un utilisateur se connecte par exemple à une application de messagerie électronique, son navigateur envoie continuellement des informations de cookie au serveur permettant son identification tout au long de la session.

Bien que la vertu simplificatrice des cookies dans le contexte des achats en ligne soit reconnue, ces fichiers fureteurs n'en demeurent pas moins au centre d'un intense débat sur la protection de la vie privée des utilisateurs. En effet, au moyen d'une combinaison de cookies et d'informations de profilage, un site Web peut apprendre beaucoup de choses sur un utilisateur particulier et fournir ces informations à une tierce partie. Qui plus est, les cookies peuvent également servir à analyser le comportement d'un utilisateur sur différents sites. En effet, les pages Web accompagnées d'annonces publicitaires utilisent des messages de demandes HTTP pour obtenir ces bannières (des images GIF ou JPEG) à partir du serveur HTTP de l'agence publicitaire responsable de leur diffusion, sachant que chacun d'entre eux peut contenir un cookie administré par l'agence. Dans la mesure où de nombreuses bannières sont diffusées par la même agence, il est souvent possible d'analyser le comportement d'un utilisateur sur plusieurs sites différents.

Pour une introduction très complète et très abordable de l'univers des cookies, nous vous conseillons le livre Persistent Client State HTTP Cookies [Netscape Cookie 1999]. Nous vous recommandons également l'ouvrage Cookie Central [Cookie Central 2002], qui donne de nombreuses informations sur la controverse provoquée par ces « espions ».

2.2.5 GET conditionnel

En enregistrant les objets déjà demandés une fois, la mise en mémoire cache sur le Web tend à réduire les temps de réponse et à diminuer le trafic Web encombrant l'internet. Les caches Web peuvent se trouver chez le client (administrés par le navigateur de l'utilisateur) ou au sein d'un serveur cache intermédiaire du réseau. La mise en mémoire cache est traitée à la fin de ce chapitre.

Si le procédé de mise en mémoire cache réduit considérablement le temps de réponse perçu par l'utilisateur, il pose cependant un problème. En effet, la version d'un objet résidant dans la mémoire cache d'un serveur peut être périmée. L'objet hébergé sur le serveur Web a pu être modifié depuis sa mise en mémoire cache au sein du poste client. Heureusement, HTTP est doté d'un mécanisme permettant au client de bénéficier du système de mise en mémoire cache tout en s'assurant de l'actualité des objets soumis au navigateur. Ce mécanisme est connu sous le nom de **GET conditionnel**.

Un message de demande HTTP est considéré comme un message conditionnel s'il
recours à la méthode `GET` et s'il contient une ligne d'en-tête `If-Modified-Since`
`(ligne d'en-tête)`.

Illustrons le mode d'opération de la méthode GET à l'aide d'un exemple concret
tout d'abord, un navigateur sollicite auprès d'un serveur Web quelconque un obje
pour la première fois, ce qui signifie que ce dernier est absent de la mémoire cache d
système.

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
```

Après quoi, le serveur Web envoie au client un message de réponse contenant l'objet

```
HTTP/1.0 200 OK
Date: Wed, 12 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24
Content-Type: image/gif
(données données données données données ...)
```

Le client affiche l'objet sur l'écran de l'utilisateur et le place aussi dans sa mémoir
cache locale. Notez que le client enregistre également la date de dernière modificatio
de l'objet, qui lui permettra de comparer sa version à celle disponible sur le serveur a
moment de la requête suivante. Puis, une semaine plus tard, le client sollicite le mêm
objet, qui se trouve toujours dans la mémoire cache. Étant donné que celui-ci a pu
être modifié entre-temps, le navigateur effectue un contrôle de mise à jour au moyer
d'un GET conditionnel. Le message envoyé est le suivant :

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
If-modified-since: Mon, 22 Jun 1998 09:23:24
```

Remarquez que la valeur de la ligne d'en-tête `If-modified-since: (ligne d'en-tête)` est la même que celle de l'en-tête `Last-Modified:` contenue dans le message de
réponse de la semaine précédente. Ce GET conditionnel demande au serveur de
n'envoyer l'objet que s'il a été modifié depuis la date de dernière consultation. Suppo-
sons que ce dernier n'ait pas été modifié depuis le 22 juin 1998 à 09:23:24. Alors, le
serveur Web envoie au client le message de réponse suivant :

```
HTTP/1.0 304 Not Modified
Date: Wed, 19 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
```

(squelette vide)

Nous voyons qu'en réponse au GET conditionnel, le serveur Web envoie bel et bien
un message de réponse, mais sans inclure l'objet sollicité. Ceci consommerait de la
bande passante inutilement et ne ferait qu'augmenter le temps d'attente de l'utili-
sateur, surtout si l'objet en question est volumineux. Remarquez que ce dernier

message de réponse contient dans sa ligne d'état la mention `304 Not Modified`, qui informe le client qu'il peut utiliser la version de l'objet stockée dans la mémoire cache.

2.2.6 Contenus HTTP

Dans ce chapitre, nous sommes pour l'instant partis du principe que les données portées par les messages de réponse HTTP étaient systématiquement constituées d'éléments de pages Web, en d'autres termes de fichiers HTML, GIF, JPEG, d'applettes Java, etc. Nous avons présenté HTTP appliqué au Web afin d'accompagner sa description d'un exemple tangible et familier, celui de la navigation Web. Mais ce serait une erreur de croire que HTTP ne sert qu'au transfert de ces types de fichiers.

En réalité, HTTP est souvent employé par les applications de commerce électronique pour transférer des fichiers XML d'un poste à un autre en l'absence de tout navigateur ou utilisateur. Les banques ont souvent recours à XML pour structurer leurs données (telles que les informations associées aux comptes de leurs clients) et les distributeurs de billets utilisent HTTP pour échanger ces informations structurées par XML. Un document XML classique procure des données structurées associées et des instructions sur la manière de les interpréter, sans fournir en général d'indications de formatage comme le fait HTML. HTTP sert également à transférer des documents VoiceXML, WML (le langage de balisage du WAP), et d'autres types de documents XML. Et HTTP est en outre le protocole de transfert de prédilection des systèmes de partage de fichiers de poste-à-poste. Le chapitre 6 montre que HTTP est fréquemment utilisé pour le transfert de données audio et vidéo en streaming.

2.3 Transfert de fichiers : FTP

Au cours d'une session FTP classique, l'utilisateur dispose d'un terminal (appelé serveur local) et il transfère des fichiers depuis ou vers un serveur distant. Pour accéder au compte distant, l'utilisateur doit entrer des données d'identification (nom et mot de passe). Après avoir fourni ces informations, il est libre de transférer des fichiers entre le disque dur (serveur) local et le système distant. Comme l'indique la figure 2.10, l'utilisateur interagit avec le protocole FTP par le biais d'un agent utilisateur FTP. Il doit tout d'abord préciser le nom du serveur distant, ce qui conduit à l'établissement d'une connexion TCP entre le poste local et le serveur FTP, puis donner son nom et son mot de passe, lesquels sont envoyés sur la connexion en tant que commandes FTP. Dès que la session de l'utilisateur est autorisée par le serveur distant, le transfert de fichiers peut commencer.

HTTP et FTP, qui sont tous deux des protocoles de transfert de fichiers, partagent un grand nombre de points communs, dont le recours à TCP. Ces deux protocoles de la couche Applications diffèrent cependant par certains aspects fondamentaux ;

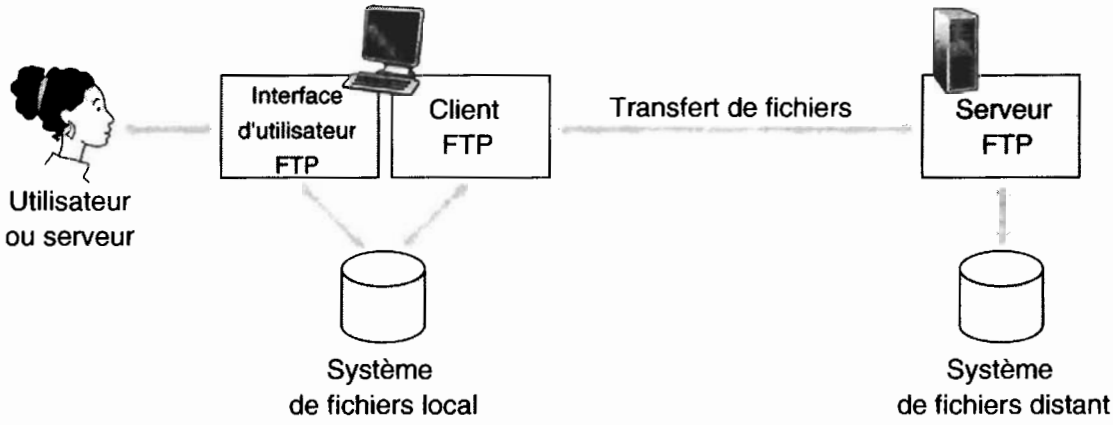

Figure 2.10 • FTP permet le transfert de fichiers entre systèmes de fichiers local et distant.

ainsi FTP utilise deux connexions TCP en parallèle pour transférer un fichier, une **connexion de contrôle** et une **connexion de données**. La première sert à l'envoi d'informations de contrôle entre les deux serveurs, telles que les données d'identification de l'utilisateur, son mot de passe, les commandes de changement de répertoire distant et les commandes permettant de télécharger des fichiers. La seconde sert à l'envoi du fichier sollicité par l'utilisateur, dans un sens ou dans l'autre. Comme FTP utilise une connexion de contrôle indépendante, on dit qu'il envoie ses commandes **hors bande**. Le chapitre 6, montre que le protocole RTSP, qui sert au contrôle du transfert de fichiers audio et vidéo en continu, envoie également des informations de contrôle hors bande. En revanche, HTTP transmet ses entêtes de messages de demande et de réponse sur la même connexion que celle empruntée par le fichier sollicité. Pour cette raison, on dit que HTTP envoie ses informations de contrôle **dans la bande**. SMTP, protocole privilégié des applications de messagerie électronique, utilise également un mode de transfert dans la bande. Les connexions de contrôle et de données FTP sont illustrées à la figure 2.11.

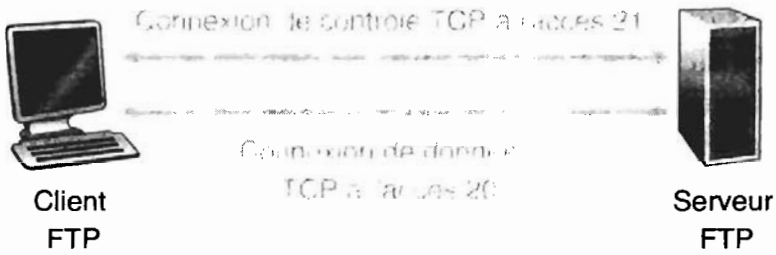

Figure 2.11 • Connexions de contrôle et de données.

Lorsqu'un utilisateur ouvre une session FTP avec un serveur distant, le pôle client de FTP (l'utilisateur) établit tout d'abord une connexion de contrôle TCP avec le pôle serveur (serveur distant) sur l'accès 21. Le pôle client envoie les données d'identification et le mot de passe de l'utilisateur sur cette connexion de contrôle, de même que les commandes de changement de répertoire distant. Sur réception

de ces informations et de cet ordre de transfert, le pôle serveur établit une connexion de données en sens inverse. FTP interrompt la connexion dès l'envoi du premier fichier. Si un utilisateur souhaite transférer d'autres fichiers durant la même session, il s'établit une nouvelle connexion de données. Ainsi, seul le contrôle de connexion demeure ouvert tout au long d'une session, tandis que les connexions de données s'achèvent après le transfert de chaque fichier (en d'autres termes, les connexions de données sont de type non-persistant).

Au cours de la session, le serveur FTP doit conserver des informations d'**état** sur l'utilisateur. Notamment le serveur associe la connexion de contrôle à un compte utilisateur spécifique et il garde trace des changements dans le répertoire de l'utilisateur tout le long de son exploration du répertoire distant. Le maintien de ces informations d'état pour chaque session d'utilisateur limite sérieusement le nombre de sessions pouvant avoir lieu simultanément. En revanche, FTP est un protocole **sans mémoire**, qui n'a donc pas besoin de garder trace de ce genre d'informations.

2.3.1 Commandes et réponses FTP

Concluons cette section par un bref aperçu des commandes FTP les plus courantes. Les commandes (transmises de client à serveur) et les réponses (transmises de serveur à client) empruntent la connexion de contrôle. Leur format ASCII à sept bits les rend, à l'image des commandes HTTP, compréhensibles à tous. Chaque commande est composée d'un groupe de quatre majuscules, accompagnées ou non de paramètres, selon le schéma suivant :

- `USER username`: Transmet le nom d'utilisateur au serveur.
- `PASS password`: Transmet le mot de passe au serveur.
- `LIST`: Demande au serveur de transmettre une liste de tous les fichiers contenus dans le répertoire distant actuel. Cette liste est envoyée sur une nouvelle connexion de données (non-persistante) et non sur la connexion de contrôle TCP.
- `RETR filename`: Télécharge (obtient) un fichier à partir du répertoire distant. Ordonne au serveur distant d'établir une connexion de données et d'envoyer le fichier sollicité sur la connexion de données.
- `STOR filename`: Enregistre un fichier dans le répertoire distant.

Il y a généralement une correspondance exacte entre la commande générée par l'utilisateur et la commande FTP transmise sur la connexion de contrôle. Chaque commande donne lieu à une réponse du serveur vers l'utilisateur. D'une structure volontairement comparable au code d'état et à la phrase de la ligne d'état des messages HTTP, ces réponses se présentent sous la forme de numéros à trois chiffres suivis d'un message optionnel.

- `331 Username OK, password required`
- `125 Data connection already open; transfer starting`

- `425 Can't open data connection`
- `452 Error writing file`

Le RFC 959 donne le détail des autres commandes et réponses FTP.

2.4 Le courrier électronique sur l'internet

Le courrier électronique existe depuis l'origine de l'internet. De fait, ce fut pendant longtemps son application la plus utilisée [Segaller 1998]. N'ayant pas cessé d'évoluer, le courrier électronique continue à figurer parmi les applications « phare » de l'internet.

Comme pour le courrier normal, le courrier électronique constitue un moyen de communication de type asynchrone, les utilisateurs pouvant consulter leur boîte aux lettres au moment qui leur convient. À la différence du courrier postal, le courrier électronique est rapide, facile à distribuer et gratuit. Avec une liste de destinataires, un même message peut être envoyé à des milliers de personnes à la fois. Qui plus est, les systèmes de messagerie actuels permettent l'envoi de pièces jointes. Ils peuvent inclure des hyperliens et du texte au format HTML et véhiculer des images, du son, des vidéos et des applettes Java. Dans cette section, nous allons étudier les protocoles de couche d'application utilisés pour l'échange de courrier électronique sur internet. Mais avant de nous lancer dans l'étude approfondie de ces protocoles, examinons le système de messagerie proposé sur internet et ses principaux éléments.

La figure 2.12 présente ce système sous la forme d'un schéma, dans lequel nous reconnaissons trois éléments fondamentaux : les **agents utilisateurs**, les **serveurs de messagerie** et le **protocole SMTP** (*Simple Mail Transfer Protocol*). Ces trois éléments sont présentés à travers l'envoi d'un courrier électronique d'Alice (l'expéditrice) à Bob (le destinataire). Les agents utilisateurs sont constitués des matériels et des logiciels qui permettent aux protagonistes de lire, répondre, rediriger, sauvegarder les messages reçus et composer de nouveaux messages. Les agents utilisateurs appliqués au courrier électronique sont parfois appelés « lecteurs de messagerie », mais nous éviterons d'employer ce terme dans cet ouvrage. Lorsque Alice a fini de composer son message, son agent utilisateur l'envoie au serveur de messagerie qui le place dans sa file de messages sortants. Lorsque Bob veut consulter son courrier électronique, son agent utilisateur retire le message de sa boîte aux lettres virtuelle située dans le serveur de messagerie. À la fin des années 1990, les agents utilisateurs GUI (*Graphical User Interface*) connurent un grand essor, permettant aux utilisateurs de consulter et de créer des messages multimédias. Les agents GUI les plus utilisés actuellement sont Eudora, Microsoft Outlook et Netscape Messenger. Il existe également des interfaces de messagerie textuelle disponibles en ligne, telles que le mail, le pine et le elm.